

Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Real time textrendering

Examensarbete utfört i Informationskodning
vid Tekniska högskolan vid Linköpings universitet
av

Gustav Adamsson

LiTH-ISY-EX--YY/NNNN--SE

Linköping 2015



Linköpings universitet
TEKNISKA HÖGSKOLAN

Real time textrendering

Examensarbete utfört i Informationskodning
vid Tekniska högskolan vid Linköpings universitet
av

Gustav Adamsson

LiTH-ISY-EX-YY/NNNN--SE

Handledare: **Jens Ogniewski**
ISY, Linköpings universitet
Mattias Wingstedt
Visiarc

Examinator: **Ingemar Ragnemalm**
ISY, Linköpings universitet

Linköping, 9 april 2015



Avdelning, Institution
Division, Department

informationskodning
Department of Electrical Engineering
SE-581 83 Linköping

Datum
Date

2015-04-09

Språk

Language

 Svenska/Swedish Engelska/English _____**Rapporttyp**

Report category

 Licentiatavhandling Examensarbete C-uppsats D-uppsats Övrig rapport _____**ISBN**

—

ISRN

LiTH-ISY-EX--YY/NNNN--SE

Serietitel och serienummer

Title of series, numbering

ISSN

—

URL för elektronisk version<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-XXXXX>**Titel**Realtidstextrendering
Real time textrendering**Författare**Gustav Adamsson
Author**Sammanfattning**

Abstract

Det här som vi har hållit på med är jätteviktigt faktiskt och det vi gjort blev bara sååå bra.
Kanske inte helt otippat, men det glass är sååå gott!

Förresten har vi blivit bäst på att skriva rapporter, så nu ska ska vi inte gå in närmare
på några detaljer såhär i sammanfattningen.

Nyckelord

Keywords textrendering, distance field, distance map, distance transform

Abstract

If your thesis is written in English, the primary abstract would go here while the Swedish abstract would be optional.

Acknowledgments

We think alla har varit så himla goa hela den här långa och tuffa tiden i våra liv.

Linköping, Januari 2020
N N och M M

Contents

Notation	ix
1 Introduction	1
2 Background	3
2.1 Computer graphics	3
2.2 Basic text rendering	3
2.3 Distance fields	5
2.4 Beziér curves	6
2.5 Polygon filling	8
3 Related work	11
3.1 Early EDT algorithms	11
3.2 Exact EDT algorithms	12
3.3 Improved distance measure for EDT algorithms	13
4 Method	15
4.1 Distance field generation initial attempt	15
4.2 Fast and approximate distance field generation	16
4.2.1 Bezier to line segment approximations	16
4.2.2 Drawing the glyph	18
4.2.3 Drawing distance gradients	18
5 Resultat	23
5.1 Ditten	23
5.2 Framtiden	23
5.A Ett par långa bevis	26
6 Avslutande kommentarer	27
Bibliography	29
Index	32

Notation

NÅGRA MÄNGDER

Notation	Betydelse
\mathbb{N}	Mängden av naturliga tal
\mathbb{R}	Mängden av reella tal
\mathbb{C}	Mängden av komplexa tal

FÖRKORTNINGAR

Förkortning	Betydelse
ARMA	Auto-regressive moving average
PID	Proportional, integral, differential (regulator)

1

Introduction

Redan de gamla grekerna och rommarna trodde att...

2

Background

This chapter includes relevant background information about the main topics of this report. The purpose of the chapter is to give the reader some basic knowledge and understanding about the touched topics to facilitate further reading of the report.

2.1 Computer graphics

Computer graphics is good.

2.2 Basic text rendering

Text rendering is a non trivial subject in computer graphics and has been for a long time. There are many different ways to render text to a screen. A common way is to pre render all the glyphs of a font to an image. Characters of this font can then be rendered by using a part of the image as texture for a polygon. Another way is to use some library to render the text to an image and use it to texture a quad [FreeType, 2014a]. Both of the above mentioned methods has some drawback, for example the text can not be scaled up without losing the smooth edges and the images takes a lot of memory if you want to have high resolution text.

When rendering a text it is really important that every glyph is positioned to each other as specified in the font file. Every glyph in a font has data stored in the font with information. For example the OpenType and the TrueType™ font format has information about of advance, kerning, height, width etc [Microsoft Corporation, 2015, Apple Inc., 2015].

Without kerning it might be hard to get a good looking text if the font is not build in a way that kerning is not a factor. Kerning is displacement along the

axis of the advancing direction of the text. A kerning value can be positive and negative and is a function of two parameters, the previous letter and the current letter. A negative kerning value is the most common and it means that the current glyph will be moved back in the advancing direction of the text and a positive value means that it will be moved forward. To get a better understanding of what kerning is take the strings “AV” and “AA” as an example. It is obvious that the left part of “V” is hanging above the “A” but that is not the case in the second string where “A” is followed by an “A”. This is because the kerning table in the font has a negative entry for the combination “AV” but not for the combination “AA”. [FreeType, 2014b]

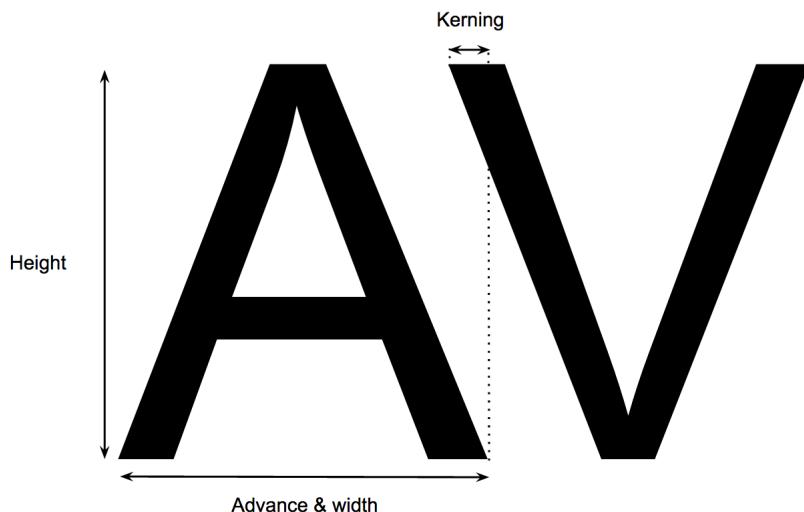


Figure 2.1: Illustration of how kerning works

Another important concepts when rendering text is baseline. A text can be placed on several different baselines on different heights. The baselines used in this thesis work is top, hanging, middle, alphabetic, ideographic and bottom. The most commonly used baseline is the alphabetic baseline which is used when for example writing english text on a piece of paper.

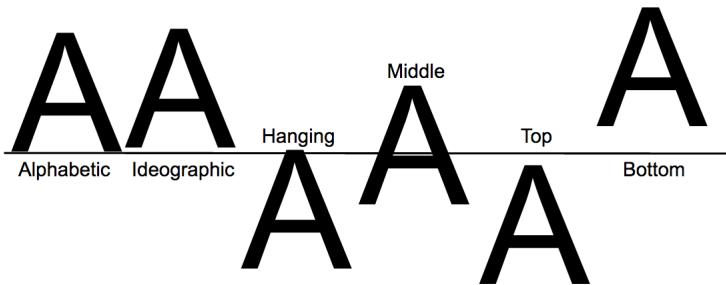


Figure 2.2: Some baselines

2.3 Distance fields

Binary images is used frequently in image processing [Ragnemalm, 1993]. A binary image is an image with only one binary value for each pixel. This will limit the image to only contain two values or colors. One example of an image that could be represented as a binary image is a picture of a white letter on a black background. In this case white would be represented as 0 and black as 1 or vice versa.

A distance transform is the transform of an input binary image to an output image called distance field or distance map [Rosenfeld and Pfaltz, 1966]. The values of the distance field pixels represents the closest distance by some distance metric to an arbitrary shape in the input image. The distance metric can differ between different applications but the most commonly used distance metric is the euclidean distance metrics which is also called real distance. If euclidean distance metrics is used when doing the distance transform it is called euclidean distance transform(EDT). Other distance metrics that can be used in distance transforms is the city-block distance metric and the chessboard distance metric. When using city-block distances you are only allowed to travel horizontally or vertically in a grid and with the cost 1. Chessboard distance is an extension of city-block distance where you are also allowed to travel diagonally with the cost of 1.

A distance field can be signed or unsigned. An unsigned distance field only maps the distances either inside the shape or outside the shape while a signed distance field maps the pixels inside the shape with negative distances and the outside of the shape with positive distances or vice versa.

Distance fields has many good advantages compared to other methods for storing shapes like boundary representation. One of the most obvious advantages is that a distance field represents more than just the boundary. For example a signed distance field also represents the environment around the boundary, both the interior and the exterior. This makes it easy to check if a point is inside or outside a shape by checking the value of that point in the distance field. It is also easy to move the boundary of the object by just changing the threshold value

determining where the boundary is located. [Jones et al., 2006]

2.4 Beziér curves

A beziér curve is defined in space or the plane as two endpoints and a number of control points which are blended together with one blending functions per point. The number of control points depends on the degree of the beziér curve. The most commonly used beziér curves is the cubic beziér with four points and the quadratic beziér with three points. A cubic beziér is defined as:

$$B(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3,$$

where $t \in [0, 1]$, P_0 and P_3 are the endpoints of the beziér and P_1 and P_2 are the control points of the beziér. The sum of the blending functions is always equal to 1 for any t in the function domain.[Ragnemalm, 2008]

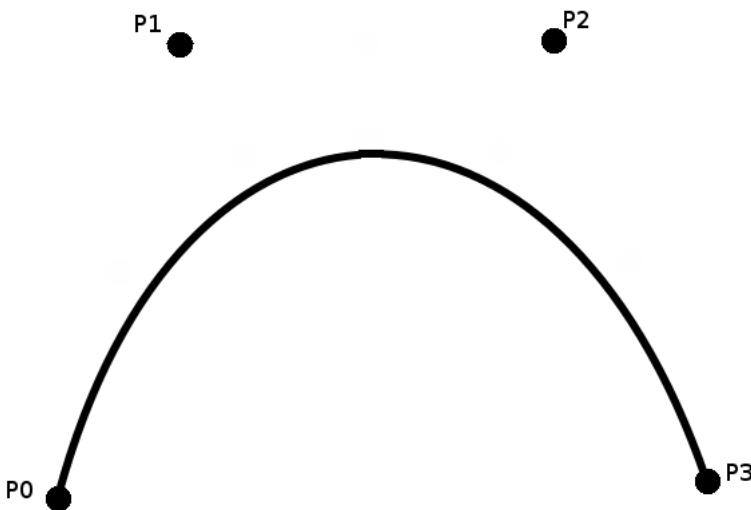


Figure 2.3: A cubic beziér curve

A Bezier curve can not represent arbitrary curves, therefore it is important to be able to connect several Beziers to a path to be able to describe more complex shapes. When connecting curves, continuity between the curves is an important concept to have in mind. Parametric continuity is a measure that will have grave importance of the appearance of the curve. There are three levels of parametric continuity.

C^0 = The curves meet

C^1 = The derivative for the both curves is equal where the curves meet

C^2 = The second derivative for the both curves is equal where the curves meet

Another measure than can be used for continuity is geometric continuity which is almost the same as parametric continuity. The only difference is that geometric continuity only require the derivatives of G^1 and G^2 to be proportional and not equal. Given two cubic beziér curves p defined by p_1, p_2, p_3 and p_4 and q defined by q_1, q_2, q_3 and q_4 . Assume that p and q is placed in a way that $p_4 = q_1$. This will trivially fulfill the requirement for C^0 because they share one endpoint and $p(1) = q(0)$. C^1 continuity, $p'(1) = q'(0)$ will be fulfilled if p_3, p_4 and q_1 is located on the same line. C^2 continuity $p''(1) = q''(0)$ will be fulfilled if p_3, p_4 and q_1 is located on the same line and $|p_4 - p_3| = |q_2 - q_1|$ meaning that the distance from p_3 to p_4 must be equal to the distance from p_4 to q_2 .

An example of an application of beziérs is fonts. Every glyph in a font is stored as a number of straight lines and bezier curves, either cubic or quadratic.[Phinney, 2001] Even though beziérs is used in fonts it is not a trivial problem to draw a beziér to the screen. Beziér curves is hard to draw because at the lowest level graphics hardware can only draw polygons and line segments. This is solved by approximating smooth curves to line segments before drawing. [Shreiner et al., 2009] Another problem with beziér curves is that it time consuming to find the closest point on the beziér curve given an arbitrary point. To solve this problem $(p - q(t))q'(t) = 0$, where $t \in [0, 1]$, p is a point in space and q is a beziér curve, has to be solved[Chen et al., 2007]. This implies that a quintic polynomial needs to be solved to find the closest point on a cubic beziér curve given a point in space. This is another reason why another representation like line segments is much easier to work with in applications like computer graphics.

De Casteljau's algorithm can be used to subdivide a bezier curve recursively by using the properties of bezier curves to calculate new beziér points for both the left and the right part of the old beziér.[Fischer, 2000] A special case of the algorithm is to divide the curve at $t = 0.5$ which gives a first curve $t \in [0, 0.5]$ and a second curve $t \in [0.5, 1]$. An example of this subdivision is presented in figure XXX below, where the first curve is described by the points p_0, p_{01}, p_{012} and p_{0123} and the second curve is described by $p_{0123}, p_{123}, p_{23}$ and p_3 is derived from the initial curve described by p_0, p_1, p_2 and p_3 . The following calculations shows how the points is computed.

$$P_{01} = \frac{P_0 + P_1}{2}$$

$$P_{23} = \frac{P_2 + P_3}{2}$$

$$P_{12} = \frac{P_1 + P_2}{2}$$

$$P_{012} = \frac{P_{01} + P_{12}}{2}$$

$$P_{123} = \frac{P_{12} + P_{23}}{2}$$

$$P_{0123} = \frac{P_{012} + P_{123}}{2}$$

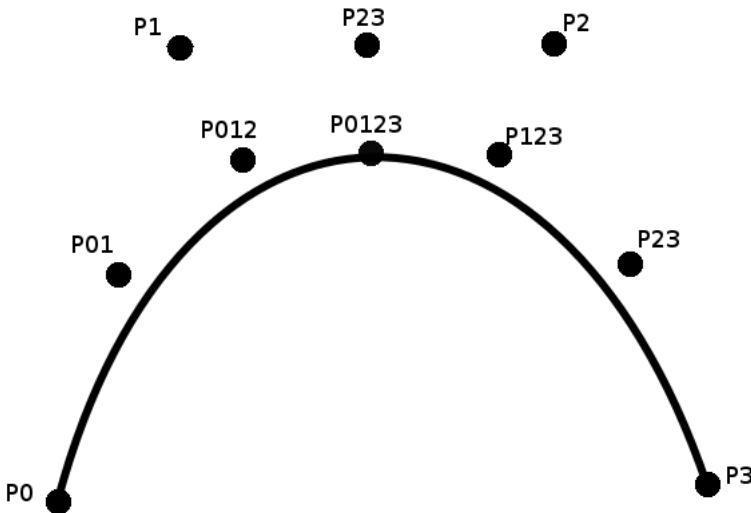


Figure 2.4: An illustration of the casteljau's algorithm splitting a cubic bezier at $t = 0.5$

2.5 Polygon filling

A polygon is a shape bound together by a finite number of line segments. Drawing polygons is what computer graphics is about but this does not make it a trivial problem. There are several algorithms for doing this where each one of them have their advantages and drawbacks. One algorithm used a lot is the scan-line polygon fill algorithm. The algorithm contains two main components. The first component is a sorted edge table which is an array of linked lists where each linked list corresponds to a row in the image. The second component is an active edge list which is an initially empty linked list of edge references. The x value for the lowest y value of each line segment is inserted into the edge table. The next step in the algorithm is to iterate through the edge table row by row. All edges in the active edge list that ended on the previous line are removed. All remaining values of the active edge list are updated to the intersection between the line segment and the current row. All values of the current row in the edge table are then inserted into the active edge list and the active edge list is then sorted by x value. The image can now be filled for the current row by using some fill rule. The odd even fill rule is easy to use by just iterating through the active edge list and filling the pixels between every pair of edges starting with an odd edge.

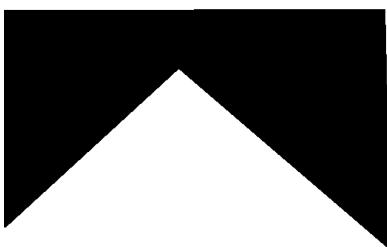


Figure 2.5: A case managed by odd even fill

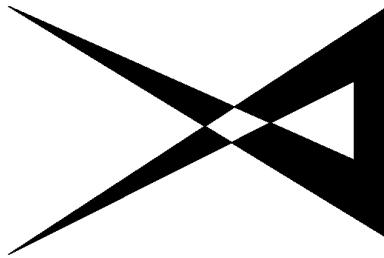


Figure 2.6: A case not managed by odd even fill

The odd even fill rule only work if edges can't cross eachother. If edges cross eachother there might be filled where it should not be filled and vice versa. An example of this can be seen in figureXXX where the odd even fill rule fails to fill the area in the middle. To handle this another fill rule can be used like the non-zero winding number rule. The property of this rule that makes it handle the case in the figure is that it also considers which direction the edges of the polygon are moving and increments a variable with different sign depending on that direction when crossing an edge.

3

Related work

3.1 Early EDT algorithms

In a very often mentioned article Danielsson [1980] proposed an improved way to generate distance maps by representing the output of the distance transform as a vector, separating the distance of the different dimensions. Danielsson also proposed two sequential algorithms 4SED (4-point Sequential Euclidean Distance mapping) and 8SED (8-point Sequential Euclidean Distance mapping) for calculating the EDT using his representation of distance. Both the 4SED algorithm and the 8SED algorithm consists of 2 consecutive picture scans where they incrementally update pixel values depending on nearby pixels. The 8SED algorithm is described in the following pseudocode.

Algorithm 1 First scan of the 8SED algorithm

```
for j = 1 to N - 1 step 1 do
    for i = 0 to M - 1 step 1 do
        L(i, j) = min(L(i, j), L(i-1, j-1)+(1, 1), L(i, j-1)+(0+1), L(i+1, j-1)+(1,1))
    end for
    for i = 1 to M - 1 step 1 do
        L(i, j) = min(L(i, j), L(i-1, j)+(1, 0))
    end for
    for i = M - 2 to 0 step 1 do
        L(i, j) = min(L(i, j), L(i+1, j)+(1, 0))
    end for
end for
```

Algorithm 2 Second scan of the 8SED algorithm

```

for  $j = N - 2$  to  $0$  step  $1$  do
    for  $i = 0$  to  $M - 1$  step  $1$  do
         $L(i, j) = \min(L(i, j), L(i-1, j+1)+(1, 1), L(i, j+1)+(0, 1), L(i+1, j+1)+(1,$ 
         $1))$ 
    end for
    for  $i = 1$  to  $M - 1$  step  $1$  do
         $L(i, j) = \min(L(i, j), L(i-1, j)+(1, 0))$ 
    end for
    for  $i = M - 2$  to  $0$  step  $1$  do
         $L(i, j) = \min(L(i, j), L(i+1, j)+(1, 0))$ 
    end for
end for

```

The two scans are very similar. The only difference is that the first scan is done top down evaluating the pixels below and on the sides of the current pixel and the second scan is done bottom up evaluating the pixels above and on the sides of the current pixel[Ragnemalm, 1993]. The 4SED algorithm and the 8SED algorithm are not error free as Danielsson [1980] proves in his article but he also claims that the errors are rare and small and is negligible for practical purposes. In 8SED and 4SED every pixel is visited a constant number of times. This makes the time complexity of 8SED and 4SED trivially $\mathcal{O}(mn)$, if m and n are the width respectively the height of the input image.

3.2 Exact EDT algorithms

Distance transforms has been used in different applications in many years. The article from Danielsson [1980] was not the first on the subject but the fact that the algorithms he proposed in his article are some of the most widely used[Fabbri et al., 2008] makes it easy to call Danielsson one of the pionjeers on the subject. As mentioned earlier in this report, 8SED and 4SED is not exact. Exact algorithms for EDT has only been around since about the 1990s. In an article Fabbri et al. [2008] compares the execution time of exact EDT algorithms. The 6 algorithms compared in the test is Meijster et al. [2000], Maurer Jr et al. [2001], Eggers [1998], Lotufo and Zampirolli [2001], Cuisenaire and Macq [1999] and Saito and Toriwaki [1994]. The conclusion of the comparison is that the algorithms from Meijster and Maurer are the fastest but Meijster's algorithm is preferred due to slightly better performance and the fact that it is easier to implement than Maurer's algorithm.

Meijster's algorithm consist of two separate phases. The first phase iterates through each column performing a distance propagation in both directions. This will create a new image $g(i, j)$ which used in the second phase. The second phase iterates through each row left to right and right to left applying a function $DT(x, y)$ to calculate the output value of each pixel. The function depends

on what distance metric is used. The function used for euclidean distance transform follows.

$$DT(x, y) = \min_{0 \dots m} ((x - i)^2 + g(i)^2)$$

With the same motivation for 4SED and 8SED the time complexity of Meijster's algorithm is $\mathcal{O}(nm)$, if m and n are the width respectively the height of the image.

3.3 Improved distance measure for EDT algorithms

Calculating a distance transform of large size images can be very time consuming. There is a significant difference in computation time between creating a distance map from a 4096x4096 image and creating a distance map from a 64x64 image. For example, the 8SED and the 4SED algorithms both has time complexity $O(n)$ where n is the number of pixels. Time complexity $O(n)$ means that every pixel is visited a constant number of times when transforming the image. Assuming every pixel is visited once and one calculation is done per visit the larger image would require 4096 times more calculations than the smaller image. In an article from Green [2007], distance fields was generated by using a 4096x4096 binary image of a glyph as input to a distance transform. The output from the distance transform was then downsampled to a 64x64 texture. Generating a distance field using a large input image makes the discrete set of possible boundary pixels more dense compared to if a smaller input image is used. The increased density of the possible pixel set helps decrease the calculation error of the distance field assuming subpixel distance measures is not used.

In an article, Gustavson and Strand [2011] showed that the calculation errors of distance transforms can be decreased by using information about the subpixel boundary between the foreground and background pixels. In the article they use an anti-aliased greyscale input image to be able to locate the boundary in the pixels to get more precise distance measures between the pixel and the boundary.

They show that they get approximately the same amount of errors using their method on a 16x16 times smaller input image than by using the method Chris Green proposed in his article. The smaller input image increases the execution time and decreases the memory consumption by a factor of approximately 30 times according to Gustavson and Strand.

Skriv lite om hur algoritmen funkar....

4

Method

This chapter will give a detailed description on which how the problem real-time text rendering on mobile devices was solved in this thesis. This chapter is divided into two different parts. The first part gives a detailed description about the implementation of the distance transform module. This part has been reimplemented several times due to poor performance. The second part gives a detailed description about the implementation of the distance field rendering module.

4.1 Distance field generation initial attempt

There are many EDT algorithms for creating distance fields. Most of the EDT algorithms used today runs in $\mathcal{O}(nm)$, where n and m are the width respectively the height of the image. Examples of algorithms running in $\mathcal{O}(nm)$ are Danielsson [1980] and Meijster et al. [2000].

The initial implementation of distance field generation in this thesis was built around Gustavson and Strand [2011] article on the subject. A signed version of the 8SED algorithm was used along with the anti-aliased sub-pixel distance measure proposed in the article. A signed distance field has several advantages compared to an unsigned distance field. The most obvious advantage is the increased flexibility it gives and that it facilitates the implementation of proper anti-aliasing on the border of the shape due to the increased information in an environment around the border pixels[Gustavson, 2012]. To generate a signed distance field the distance transform was run twice with inverted input representation. This will make one of the transforms create a distance field on the inside of the glyph and the other transform on the outside of the glyph. The resulting distance field was then calculated with the following formula.

$$result(i, j) = inside(i, j) - outside(i, j), \forall (i, j), i \in \{0, \dots, m - 1\}, j \in \{0, \dots, n - 1\}$$

4.2 Fast and approximate distance field generation

Fast distance field generation on mobile devices requires some approximations and optimizations to decrease the transformation time. This can be seen in many of the distance field generation implementations done lately. Two examples of this is GLlyphy by Behdad Esfahbod and the implementation of distance field generation in QT. Both the implementations approximate the bezier curves from the font file to some different representation of a glyph that is easier to work with. GLlyphy approximates the bezier curves to circular arcs and QT approximates the bezier curves to line segments. This makes it possible to draw distance fields locally over the outlines of the glyph and a distance transform of the whole image is not needed.

Because the initial implementation of the distance field generation was too slow to meet the requirement another faster implementation was necessary. A variant of the implementation done by Qt was implemented. The implementation was divided into four steps in the following order, extracting beziers from the given font file, approximating the beziers to line segments, drawing the glyph in an image and drawing distance gradients over the line segments to create a distance field locally over the outlines of the glyph. Extraction of bezier curves from font files can easily be done with different libraries for example freetype and a description of how this is done is not relevant for the proceeding of this report.

4.2.1 Bezier to line segment approximations

Bezier to line segment approximations is done alot in computer graphics and can be done by using the properties of the de Casteljau's algorithm. A naive way to solve the beziér to line segment problem is to split the Beziér curve into n smaller curves with a constant length and approximate them to line segments. This would make the line density constant over the whole curve meaning that both the strongly bent parts and the almost straight parts will be represented by equally many line segments. A smarter way to solve this is proposed in an article by Fischer [2000] which involves using a recursive function dividing the bezier into two parts until a stop condition is met. This will make the line segment density higher where the curve is strongly bent and lower where the curve is almost straight. The following pseudocode is taken from the article by Fischer.

Algorithm 3 Function for approximating beziér to line segment

```

procedure flattenCurve(Curve c)
  if isSufficientlyFlat(c) then
    output(c);
  else
    Curve l,r
    subdivide(c,l,r)
    flattenCurve(l)
    flattenCurve(r)
  end if
end procedure

```

In the above code the type Curve is a beziér curve for example a cubic bezier curve represented by 4 points in the plane or a cubic beziér represented by 3 points in the plane. The 2 functions isSufficientlyFlat and subdivide is not defined in the above code because the implementation of them differs depending on the curve type used. The function isSufficientlyFlat checks the difference between the curve and the line segment from the start point to the endpoint of the curve is small enough to meet the stop condition. This function can be implemented in many different ways. In his article Fischer propose solution for cubic beziérs initially developed by Roger Willcocks. The pseudocode for this solution follows.

Algorithm 4 Stop condition for cubic beziér subdivision

```

function isSufficientlyFlat(Curve c)
  double ux = 3.0*c.b1.x - 2.0*c.b0.x - c.b3.x; ux *= ux
  double uy = 3.0*c.b1.y - 2.0*c.b0.y - c.b3.y; uy *= uy
  double vx = 3.0*c.b2.x - 2.0*c.b3.x - c.b0.x; vx *= vx
  double vy = 3.0*c.b2.y - 2.0*c.b3.y - c.b0.y; vy *= vy
  if ux<vx then
    ux = vx
  end if
  if uy<vy then
    uy = vy
  end if
  return (ux+uy ≤ tolerance)
end function

```

There are other ways to check if a curve is flat enough to approximate it to a line segment. One solution is proposed by Shemanarev [2005]. The stop condition in his article only depends on the distance between the two points P_1 and P_2 and the line between P_0 and P_4 .

The function subdivide in the algorithm for approximating beziers to line segments proposed by Fischer is implemented using the special case of de Casteljau's algorithm splitting at $t = 0.5$.

4.2.2 Drawing the glyph

The distance gradients in the distance field will only be drawn locally within a predefined distance from each outline. This creates a need to draw the glyph in the image before generating the distance field. If the glyph is not drawn before the distance gradients are drawn there will most certainly be parts of the glyph that have the outside color but is located on the inside of the glyph. For example the middle of the 'I' might be the same color as pixels outside the glyph. This would lead to a hole in the middle of the 'I' when rendering it which is not desired. The input to this step of the distance field generation is a glyph represented as connected line segments in the shape of a polygon. As mentioned in the background the scan-line algorithm together with the odd even fill rule or the non-zero winding rule can be used to fill a polygon. Because is a very common problem in computer graphics and there are many open source libraries doing this very fast and optimized there was no need to implement this. To solve this the function `drawPath` in the open source 2D graphics library Skia was used. The Skia library was already installed and used in other places within Visiarc's products.

4.2.3 Drawing distance gradients

Drawing gradients can be done in many different ways. The initial implementation of this step had real measured distances as values in each pixel in the gradient. Description of how the pixel to line segment distance was measured follows. The pixel coordinate was projected onto the line. If the projection hit the line the distance was equal to the projection distance. Otherwise the distance was equal to the shortest distance to one of the endpoints of the line.

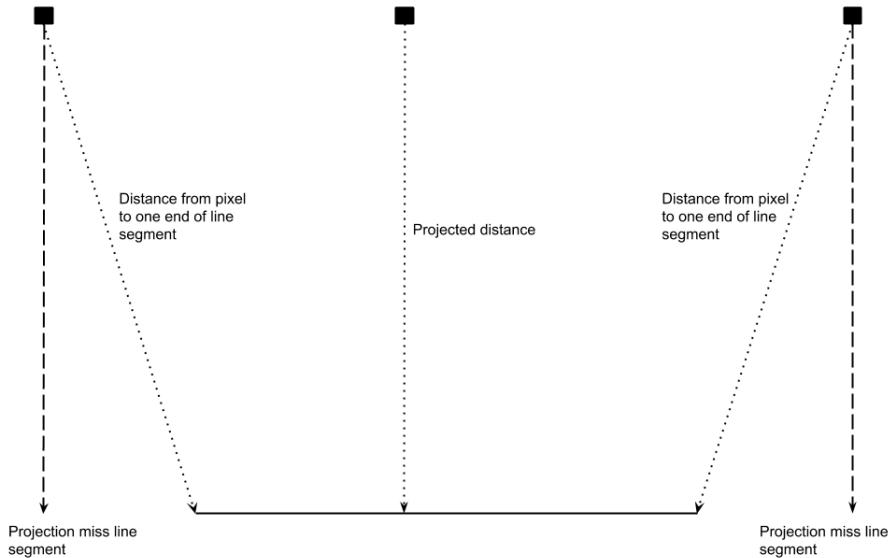


Figure 4.1: Shortest distance from pixel to line segment

The code for projecting a pixel to a line segment made the distance fields look good because there was few sources of errors drawing the gradients. Every pixel was calculated independently from eachother and if the new distance was not better than the distance already in the pixel the pixel was not updated. The only problem with this approach was the execution time. The code for pixel to line segmen distance calculation was located in the innermost loop which means that the code was executed very many times and performance improvements was neccessary.

In the distance field generation made by Qt no pixel to line projection is done. They draw the gradient directly using trigonometry stepping through the pixels row by row. This influenced how the gradients is drawn in the final result of this thesis. Every line segment will result in two gradients drawn, one rectangular over the line and one triangular close to the first end points of the line segment. The gradients drawn for line segment A and B is illustrated in figureXXX.

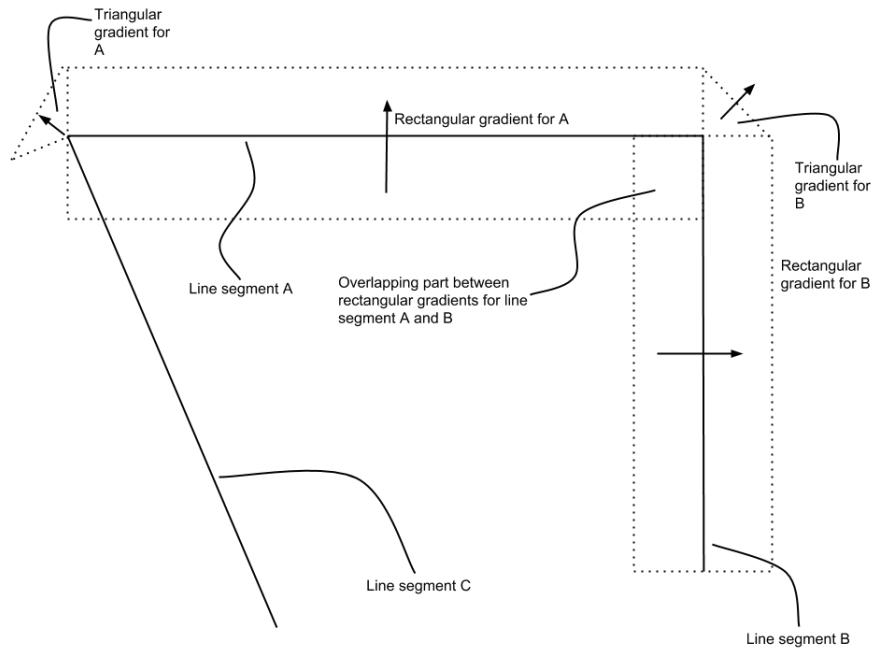


Figure 4.2: Illustration of the drawn gradients for line segment A and B

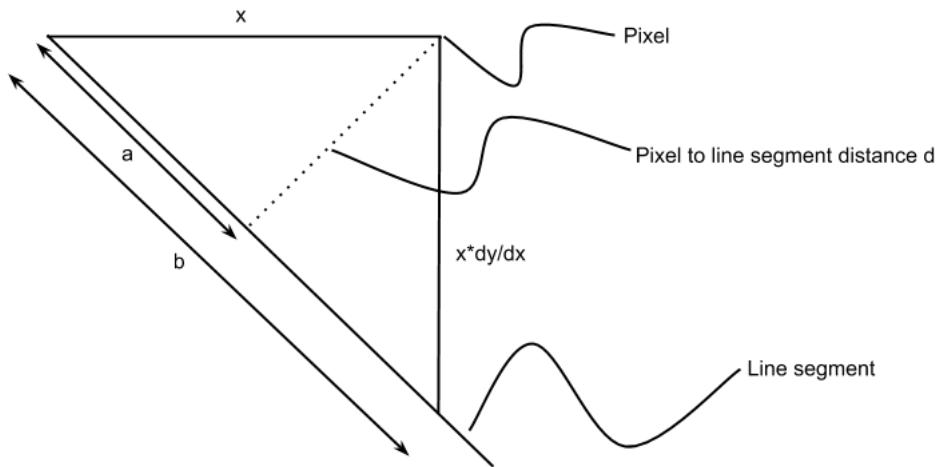


Figure 4.3: Illustration of the calculations done when drawing the gradient

The line segment is defined between two points in the plane p_1 and p_2 . The

derivative of the line can trivially be calculated $\frac{dy}{dx} = \frac{p_{2y}-p_{1y}}{p_{2x}-p_{1x}}$. The projection of the pixel to the line segment will always be perpendicular to the line segment. This implies that the triangle with sides $x, x\frac{dy}{dx}$ and b and the triangle with sides x, a and d are similar triangles which means that $\frac{x}{b} = \frac{a}{x\frac{dy}{dx}} = \frac{d}{x}$ is true. The similarity of the triangles means that d can be calculated in two ways, either $d = \frac{x^2}{b}$ or $d = \frac{a}{x\frac{dy}{dx}}$. The first solution is used in the thesis because b is easier to calculate than a $b = \sqrt{x^2 + (x * \frac{dy}{dx})^2}$. Having b the final formula for calculating d is $d = \frac{x^2}{\sqrt{x^2 + (x * \frac{dy}{dx})^2}} = \frac{x}{\sqrt{1 + (\frac{dy}{dx})^2}}$. The shortest distance from the line segment to the pixel is equal to $|d|$.

5

Resultat

Det här är kapitlet där resultaten presenteras.

5.1 Ditten

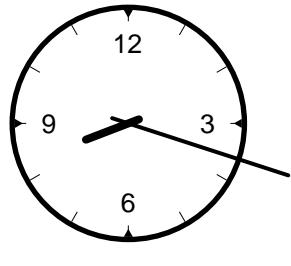
Liksom [?] har vi kommit fram till att glass smakar bäst på sommaren. Kommer
När vi nu går in på hur glass smakar vid olika tidpunkter under dagen hän- att tänka
visar vi till figur 5.1, och speciellt till figur 5.1b. Jämför sedan med figur 5.2 för på en
att se hur det kan bli när man äter glass vid okontrollerade tidpunkter. liten

Veselić, Krešimir (Veselić, Krešimir) skrev en gång en artikel med titeln *Bounds anek-
for exponentially stable semigroups.* dot...

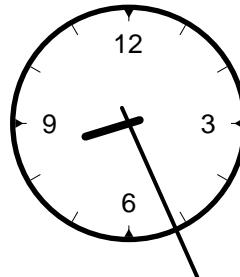
5.2 Framtiden

Sen när glassen är uppåten är det bara till att sätta igång och skriva på exjobbet
igen!

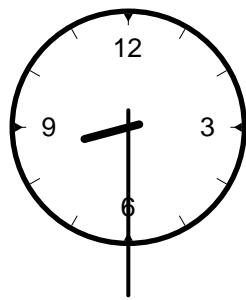
TODO:
Ta bort den löjliga
anekdoten!



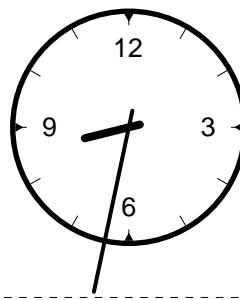
(a) Det här är väl tidigt — din glass hinner smälta innan ditt sällskap dyker upp.



(b) Kiosken stänger snart, men inte nu — perfekt!



(c) Precis i tid — du får in ett finger i luckan just när kiosken ska stänga. Han som jobbar blir sur, och det blir smolk i bågaren.



(d) Du är sen — kiosken är stängd.

Figure 5.1: Illustration av subfloats. Den så kallade bounding boxen visas i (d). Lägg märke till att bounding boxen har satts så att alla bilder har samma storlek, med enhetlig placering av själva innehållet i förhållande till bounding boxen. Antag att du ska träffa en kompis för att äta glass just när kiosken stänger för dagen vid 08:30. När dyker du upp?

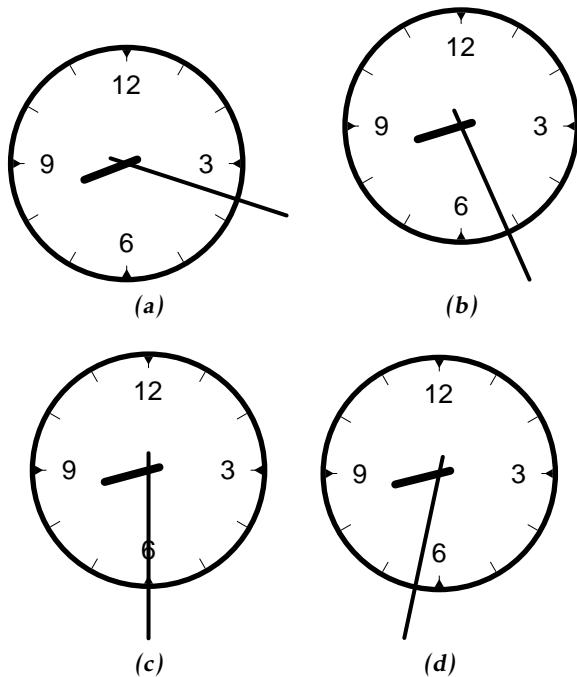


Figure 5.2: En andra illustration av subfloats. Den här gången har bounding boxen gjorts så liten som möjligt runt själva innehållet. Resultatet är stöksiga placeringar på sidan. Samma sak kan hända med vanliga fyrkantiga figurer när man har text som spretar ut åt lite olika håll från själva rutan med kurvor i.

Appendix

5.A Ett par långa bevis

Det här är en appendix-del av det aktuella kapitlet.

6

Avslutande kommentarer

Sätt av ett kort kapitel sist i rapporten till att avrunda och föreslå rikningar för framtida utveckling av arbetet.

Bibliography

- Apple Inc. Truetype™ reference manual, 2015. URL <https://developer.apple.com/fonts/TrueType-Reference-Manual/>. Cited on page 3.
- Xiao-Diao Chen, Yin Zhou, Zhenyu Shu, Hua Su, and J.-C. Paul. Improved algebraic algorithm on point projection for beziércurves. In *Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*, pages 158–163, Aug 2007. doi: 10.1109/IMSCCS.2007.17. Cited on page 7.
- Olivier Cuisenaire and Benoît Macq. Fast euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76(2):163–172, 1999. Cited on page 12.
- Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14(3):227 – 248, 1980. ISSN 0146-664X. doi: [http://dx.doi.org/10.1016/0146-664X\(80\)90054-4](http://dx.doi.org/10.1016/0146-664X(80)90054-4). URL <http://www.sciencedirect.com/science/article/pii/0146664X80900544>. Cited on pages 11, 12, and 15.
- Hinnik Eggers. Two fast euclidean distance transformations in z 2 based on sufficient propagation. *Computer Vision and Image Understanding*, 69(1):106–116, 1998. Cited on page 12.
- Ricardo Fabbri, Luciano Da F. Costa, Julio C. Torelli, and Odemir M. Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1):2:1–2:44, February 2008. ISSN 0360-0300. doi: 10.1145/1322432.1322434. URL <http://doi.acm.org/10.1145/1322432.1322434>. Cited on page 12.
- Kaspar Fischer. Piecewise linear approximation of bezier curves. In *HTTP://HCKLBRRFNN. WORDPRESS. COM/2012/08/20/PIECEWISE-LINEAR-APPROXIMATION-OF-BEZIER-CURVES/*. Citeseer, 2000. Cited on pages 7 and 16.
- FreeType. Freetype outlines, 2014a. URL <http://www.freetype.org/freetype2/docs/glyphs/glyphs-6.html>. Cited on page 3.

- FreeType. Kerning, 2014b. URL <http://www.freetype.org/freetype2/docs/glyphs/glyphs-4.html>. Cited on page 4.
- Chris Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 9–18, New York, NY, USA, 2007. ACM. ISBN 978-1-4503-1823-5. doi: 10.1145/1281500.1281665. URL <http://doi.acm.org/10.1145/1281500.1281665>. Cited on page 13.
- Stefan Gustavson. 2d shape rendering by distance fields. 2012. Cited on page 15.
- Stefan Gustavson and Robin Strand. Anti-aliased euclidean distance transform. *Pattern Recognition Letters*, 32(2):252 – 257, 2011. ISSN 0167-8655. doi: <http://dx.doi.org/10.1016/j.patrec.2010.08.010>. URL <http://www.sciencedirect.com/science/article/pii/S0167865510002953>. Cited on pages 13 and 15.
- M. Jones, J.A. Baerentzen, and M. Sramek. 3d distance fields: a survey of techniques and applications. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):581–599, July 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.56. Cited on page 6.
- Roberto A Lotufo and Francisco A Zampirolli. Fast multidimensional parallel euclidean distance transform based on mathematical morphology. In *Computer Graphics and Image Processing, 2001 Proceedings of XIV Brazilian Symposium on*, pages 100–105. IEEE, 2001. Cited on page 12.
- Calvin R Maurer Jr, Vijay Raghavan, and Rensheng Qi. A linear time algorithm for computing the euclidean distance transform in arbitrary dimensions. In *Information Processing in Medical Imaging*, pages 358–364. Springer, 2001. Cited on page 12.
- Arnold Meijster, Jos BTM Roerdink, and Wim H Hesselink. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its applications to image and signal processing*, pages 331–340. Springer, 2000. Cited on pages 12 and 15.
- Microsoft Corporation. Opentype specification, 2015. URL <https://www.microsoft.com/typography/otspec/default.htm>. Cited on page 3.
- Thomas W Phinney. Truetype, postscript type 1, & opentype: What's the difference, 2001. Cited on page 7.
- Ingemar Ragnemalm. *The Euclidean Distance Transform*. Linköping Studies in Science and Technology, first edition, 1993. ISBN 91-7871-083-9. Cited on pages 5 and 12.
- Ingemar Ragnemalm. *Polygons feel no pain*. First edition, 2008. Cited on page 6.
- Azriel Rosenfeld and John L Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM (JACM)*, 13(4):471–494, 1966. Cited on page 5.

- Toyofumi Saito and Jun-Ichiro Toriwaki. New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications. *Pattern recognition*, 27(11):1551–1565, 1994. Cited on page 12.
- Maxim Shemanarev. Adaptive subdivision of bezier curves. 2005. URL http://antigrain.com/research/adaptive_bezier/. Cited on page 17.
- Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009. Cited on page 7.

Index

ARMA

 abbreviation, ix

greker, 1

PID

 abbreviation, ix



Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innehåller rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>