Antialiasing and shading, an implementation

T. van Rij

Computer Science/Department of Interactive Systems

Report CS-R9427 April 1994

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

Antialiasing and Shading, An Implementation

T. van Rij
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
tanja@cwi.nl

Abstract

A lot of research has been dedicated to representing 3 dimensional images on the screen. When rendering a 3 dimensional image, one wants the image to be as realistic as possible. Another important requirement is that the edges in the image are smooth. A realistic image can be obtained by incorporating shading algorithms. Smooth edges can be obtained by using antialiasing methods.
An experimental architecture for interactive graphics is currently being evaluated at the CWI. For this architecture an implementation of several rendering methods has been developed.
In this paper we will introduce this implementation which contains several rendering algorithms and uses an antialiasing method.

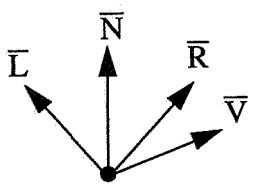
CR Subject Classification (1991): I.3.7: Three-dimensional graphics and realism - *colour, shading.*
Keywords & Phrases: Shading, antialiasing, rendering, scan-line algorithm, hardware architecture

1. INTRODUCTION

In this section we will first give an introduction to shading and antialiasing. Secondly a brief overview of the architecture will be given.

1.2. Shading

Shading is the colouring of a 3 dimensional object, depending on its orientation and the position of one or more light sources.
The intensity or colour at a point of a three dimensional object can be calculated by a formula introduced in [Phong, 75]. This formula makes use of the following parameters:



$\overline{N}$ = Normal of the surface at a certain point

Report CS-R9427
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

$\overline{L}$ = Vector pointing in the direction of a point light source
$\overline{R}$ = Reflection vector (= Light vector mirrored in N)
$\overline{V}$ = Vector pointing in the direction of the viewpoint
I_a = Ambient light intensity
I_l = Intensity of the point light source
k_a = Ambient-reflection coefficient
k_d = Diffuse-reflection coefficient
k_s = Specular-reflection coefficient
n = Specular-reflection exponent represents the shininess of a 3 dimensional object.

The Ambient-reflection coefficient, diffuse-reflection coefficient and the specular-reflection coefficient are surface properties of the material of which the three dimensional object is made of.
The formula for the intensity I at a certain point is as follows:

$$I = (I_a * k_a) + (I_l * ((k_d * (\overline{N}.\overline{L})) + (k_s * (\overline{R}.\overline{V})^n)))$$

Several techniques have been developed for the shading of graphical objects approximated by 3 dimensional flat polygons. When the 3 dimensional polygons approximate an object, then the normals at the vertices of the polygons can be different. The most well known shading techniques are flat shading, Gouraud shading and Phong shading.
When using flat shading, one intensity is calculated for each polygon of the graphical object. Each polygon is then coloured in its own colour.
The Gouraud-shading algorithm described in [Gouraud, 71], calculates an intensity for each of the vertices of a polygon. The intensities are then linearly interpolated between the vertices of the polygon and across the polygon between the edges.
The Phong shading algorithm described in [Phong, 75] interpolates the normal vector $\overline{N}$ instead of the intensities. After each interpolation the intensity at that point is calculated with the above mentioned formula. The implementation of the flat shading and Gouraud shading algorithms will be discussed in Section 3.

1.3. Antialiasing

Images which are generated for graphic displays are generally stored in a frame buffer. This is a two dimensional array which represents the image in screen coordinates. These points on the screen are also called pixels. In computer graphics much work has been done to solve the problem of jagged edges and disappearing detail due to the discrete nature of the frame buffer. The application of techniques to solve the problems caused by the discrete nature of the frame buffer is referred to as antialiasing.
There are roughly three ways of applying antialiasing. The first one is applying a convolution filter on the image in the hope that the inaccuracies are not visible any more. A disadvantage of this approach is that the images may loose their sharpness. The second one is to virtually increase the resolution of the screen. An example of this is dividing each pixel into four. The intensity value of each partial pixel is then averaged to one value. When applying this technique you are actually taking more samples per pixel. The last way is to apply area sampling. When applying area-sampling you look at a pixel as if it represents a finite area in the scene, this has the effect of applying a convolutional filter before the scene is sampled. With this technique you are actually taking an infinite number of samples per pixel. A comparison of these techniques is described in [Crow, 81]. For a good overview of antialiasing techniques

using filters read [Blinn, 89a] and [Blinn, 89b]. An implementation of a filtering tiler is shown in [Crow, 77].
The most common filter used for area sampling is the box filter. This is also the filter which is used in the implementation discussed later. To explain the working of this filter, assume pixels are a non-overlapping two dimensional array of square tiles. The percentage of the polygon filling a pixel is then related to the intensity contribution for that pixel. Although the box filter gives better results than using no filter at all, there are filters which give much better results. A disadvantage of these filters is that they are computationally expensive. An example of such a filter is the cone filter. An algorithm using this filter can be found in [Gupta et al., 81].

1.4. An architecture for interactive raster graphics

Graphic architectures in general work with a frame buffer. All user interaction is performed upon the frame buffer. This means that picking operations are done on the basis of individual pixel coordinates. Afterwards the pixel coordinates are used to identify the graphical object the user pointed to. Interactive graphical systems would be much better off if a graphical object could be pointed to directly.
The architecture developed at CWI [Hagen et al., 86][Akman et al., 88][Kuijk et al., 92] concentrates exclusively on real-time interactive 3D graphics. In this architecture the frame buffer has been replaced by a structured list of objects. These objects are the result of a hidden surface removal process. The objects are displayed on the screen by a display controller. This display controller displays the objects row by row. This is done by using X-processors and Y-processors. The Y-processors take care of the intersection of objects with the scan-line. They also take care of the shading functions performed upon the object in a particular scan-line. These Y_processors produce scanline commands which are send to a one dimensional array of X-processors. These processors in turn generate the values for the pixels.
The algorithms discussed later will in fact be executed by an Y-processor. Because of the way the architecture is built, the instructions the Y-processors send to the X_processors should be constructed by using a scan-line algorithm. The instruction set for the X-processors is the following:

SetI(x, I)	Set the intensity at pixel location x to I.
SetdI(x, dI)	Set the first forward difference of the intensity at pixel location x to dI.
SetddI(x, ddI)	Set the second forward difference of the intensity at pixel location x to ddI.
SetPI(x, dx, I)	Set the intensity at pixel locations x, (x + dx), (x + (2 * dx)), (x + (3 * dx)), ...to I.
SetPdI(x, dx, dI)	Set the first forward difference of the intensity at pixel locations x, (x + dx), (x + (2 * dx)), (x + (3 * dx)), ... to dI.
SetPddI(x, dx, ddI)	Set the second forward difference of the intensity at pixel locations x, (x + dx), (x + (2 * dx)), (x + (3 * dx)), ... to ddI.
Eval0(x, dx, I)	Set the intensities between the pixel locations x and (x + dx) to I and disable the accumulation of intensities until the next Refresh commands.
Eval1(x, dx, I)	Accumulate the intensities between the pixel locations x and (x + dx). If I has been set use the set value.
Eval2(x, dx, I, dI)	Interpolate and accumulate the intensities between the pixel locations x and (x + dx) by first order forward differencing. If I or dI has been set use set values.
Eval3(x, dx, I, dI, ddI)	Interpolate and accumulate the intensities between the pixel locations x and (x + dx) by second order forward differencing. If I, dI, or ddI has been set, use the set values.
Dis(x, dx)	Disable the accumulation of the intensities between the pixel locations x and (x + dx) until the next Eval command.
Acc_mode()	Enable/disable accumulation of negative intensities.
Refresh()	Output the accumulated intensity and reset the processor.
Nop()	No operation

For flat shading we need the Eval1 and the Refresh instruction. For Gouraud shading we only need to concern us with the Refresh and the Eval2 instruction. These instructions will be generated with help of the scan line algorithm which will be discussed in the next section.

2. THE SCAN LINE ALGORITHM

The implementation of the rendering is done on the basis of an algorithm used for scan converting polygons. This algorithm, called the scan line algorithm, is described in [Foly et al., 90]. This algorithm makes use of a data structure called the **Active Edge Table** (AET). The AET keeps track of the edges which intersect with the scan line and keeps track of the intersection points.
To use this data structure we first store the edges of the polygon in another data structure called the **edge-list**. The edge-list consists of two types of data structures. Namely the **edge-list-nodes** and the **edge-nodes**.
The edge-list-node is depicted in Figure 1.

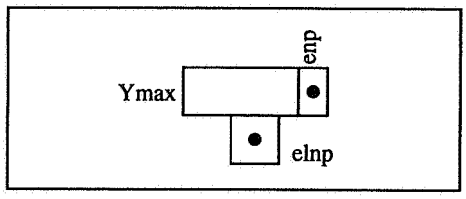


Figure 1: The edge-list-node

An edge-list-node contains the following three fields:
- The field **enp** (edge-node pointer) which contains a pointer to a linked list of nodes representing the edges
- The **Ymax** field which contains the rounded y coordinate of the topmost vertex of the edges it points to
- The field **elnp** (edge-list-node pointer) which contains a pointer to another edge-list-node

The nodes which represent the edges are called edge-nodes. The edge-node is depicted in Figure 2.

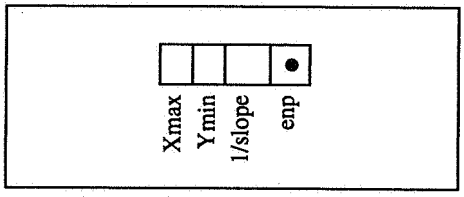


Figure 2: The edge-node

Each edge-node contains the following four fields:
- The field **Xmax** contains the value of the x-coordinate of the topmost point of the edge.
- The **Ymin** field contains the rounded value of the y-coordinate of the bottom point of the edge.
- The field **1/slope** contains the value of 1 divided by the slope of the edge (= dx/dy).
- The **enp** field contains a pointer to the next edge-node.

The complete structure of the edge-list is depicted in Figure 3.

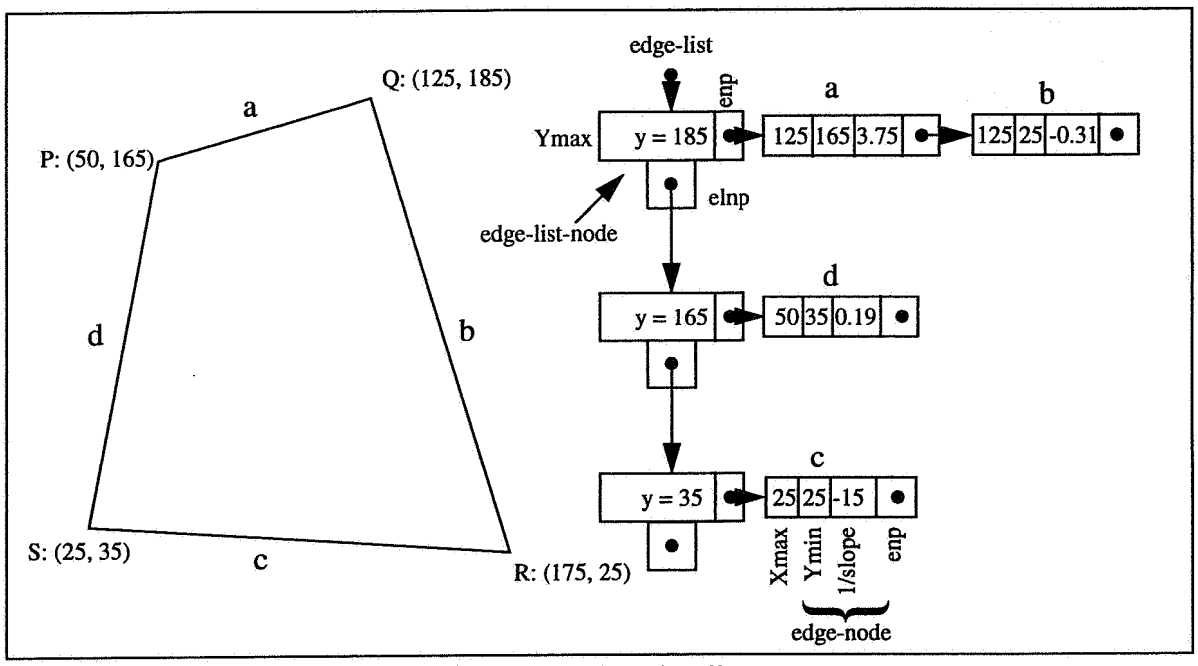


Figure 3: The edge-list

The edge-list in this figure is the edge-list of an example polygon shown in the same figure. This example polygon will be used throughout the rest of the article. The polygon has four vertices P, Q, R and S. Between the vertices P and Q we find edge a, between vertices Q and R we find edge b and so on.
The edge-list-nodes are sorted on the values of the Ymax fields. The edge-nodes attached to the edge-list-nodes are sorted on the values of their Xmax fields. The highest Ymax value is situated in the top edge-list-node and the lowest Xmax is situated in the left most edge-node.
When the edge-list has been built, the actual scan-line algorithm can begin. This algorithm maintains a list consisting of edge-nodes in the data structure AET. The edges represented by the edge-nodes in the AET are the edges which are intersecting with the current scan-line. When we move to the next scan-line the data of the edge-nodes is updated, the value of the 1/slope field is substracted from the value of the Xmax field. If there are edge-nodes present which represent edges which don't intersect with the current scan-line, then these edge-nodes get deleted. If there are new edges which intersect with the current scan-line, then the edge-nodes representing these edges are copied from the edge-list. In this way the edge-list-nodes in the edge-list are parsed from top till bottom and the edge-nodes from left to right.

3. THE IMPLEMENTATION

3.1. Flat shading

If we have calculated an intensity for a certain polygon, for example with the formula described on page 2, then we can apply the scan-line algorithm directly to the polygon. An algorithm for the building of an edge-list and the scan-line algorithm itself are described in appendix A (Algorithm 1 and 2).
For colouring with help of the hardware architecture the following rule should be added to Algorithm 2 after rule 3.:

```
From (the top of the window we are drawing in) Until (current_y has been
```

```
reached) do:
1. Refresh()
```

The Refresh() statement is an instruction for the X-processors. Here they the Refresh() statements are used to ensure that the polygon will be drawn at the appropriate height.
rule 4.2.1. should be replaced by the following rule:

```
Let X_left be the value of the Xmax field of the first edge-node in the cur-
   rent pair of edge-nodes
Let X_right be the value of the Xmax field of the second edge-node in the
   current pair of edge-nodes
Let I be the colour of the polygon.

4.2.1.  Eval1(X_left, (X_right - X_left), I)
```

The Eval1() statement is an instruction for the X-processors. Figure 4 shows the result of the algorithm applied on the example in Figure 1.

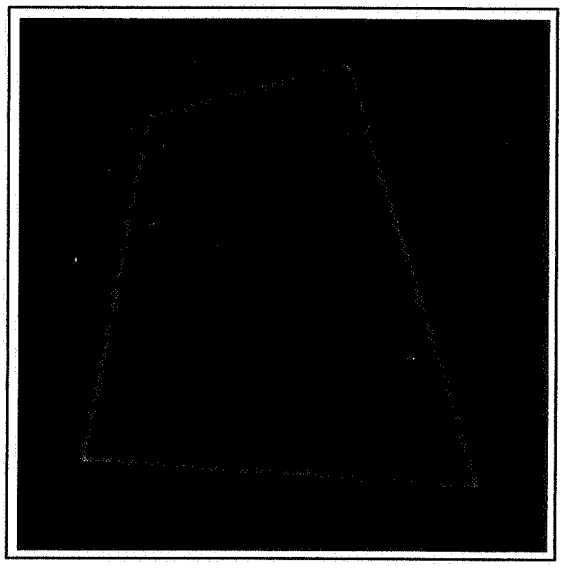


Figure 4: The polygon flat shaded.

3.2. Adding Gouraud shading to the scan-line algorithm

The scan-line algorithm presented in the previous chapter is used for filling in a polygon in one colour. With just a few changes in the data structure on the node edge-node and some minor changes to the scan-line algorithm, the scan-line algorithm can be used to Gouraud shade a polygon. To show this let us give for example the vertices P, Q, R and S in figure 1 the following intensities: Ip = 0.5, Iq = 1, Ir = 0.75, Is = 0.0 (0 = black, 1 = white). For a full colour picture we should use three valued vectors instead of single numbers. The three values in this vector would then represent the amount of Red, Green and Blue in the colour.

The added fields to the data structure edge-node are the following:

- The field **Imax** contains the value of the intensity of the topmost vertex of the edge.
- The field **di/dy** contains the difference in intensity going one step in the y direction.

The edge-node for edge c is depicted in Figure 5.

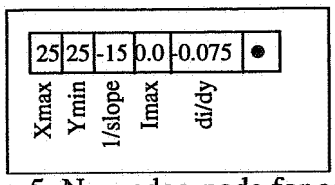


Figure 5: New edge-node for edge c

The algorithm for the building of the edge-list must now include steps which initialize the new fields. This means that after step 2 in algorithm 2 the algorithm has to be expanded. The new algorithm (algorithm 3) can be found in Appendix A Page A3.
The scan-line algorithm must do something more than colouring the pixels on a scanline between two edges in one colour. Because of this the drawing step 4.2.1. of Algorithm 2 has to change. Each step in the x direction should give an increase or decrease in intensity. The same goes for steps in the y direction or the update of the edge-nodes in the AET. The resulting algorithm (algorithm 4) is described in appendix A page A4.

For doing Gouraud shading with the hardware architecture, except for the adding of the refresh rule mentioned in the flat shading paragraph, the steps 4.2.1.1. till 4.2.1.3. of algorithm 4 can be replaced by the following step:

```
4.2.1.1. Eval2(X_left, (X_right - X_left), I_left, di/dx)
```

The Eval2() statement is an instruction for the X-processors.

The next picture shows the result of the modified algorithms applied on the example in figure 1, with the intensities mentioned earlier.

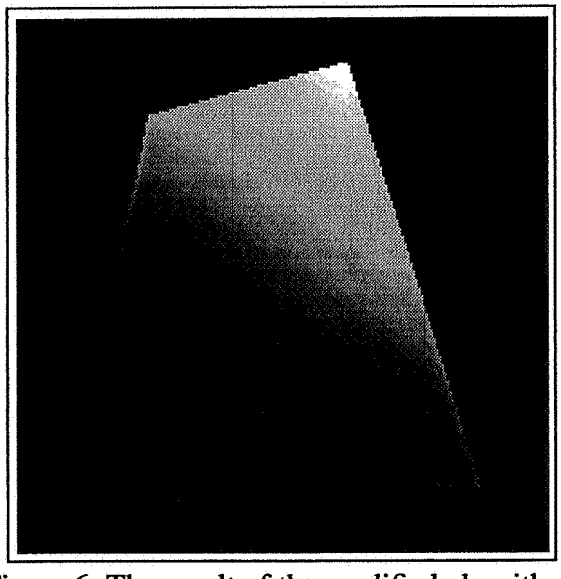


Figure 6: The result of the modified algorithms

3.3. Adding antialiasing with the box-filter

As mentioned in the introduction, to perform antialiasing with the box-filter we need to know

how much the polygon is contributing to the pixels it covers. One way of doing this is to add a value to the intensity of a pixel proportional to the amount of area covered by the polygon. This has to be done in such a way that, if we have polygons which lie next to each other, then the sum of the contributions to a pixel (if the pixel is completely covered by both polygons) is exactly 100%. To adapt the algorithm in order to calculate this we need to know the precise values concerning the end points of the edges. This means that the integer value of the Ymin field of an edge-node is no longer sufficient, it should therefore contain the floating point value of the bottom y-coordinate of the edge the edge-node is depicting. We also need to know the exact value of the current y-coordinate. This is why a new field Ymax is added to the edge-node. This field is updated each time we start with a new scan-line

Two new fields are added to the edge-node:
- The field **Ymax** initially contains the value of the y-coordinate of the topmost vertex of the edge.
- The field **Ytop** contains the same value as Ymax, but keeps this value during the whole scanning process. This field is necessary, to know if we are at the top of the polygon during scanning. This field actually contains the same value as the field Ymax of the edge-list-node. Since edge-list-nodes are not present while working with the AET, the value is copied to the Ytop field of the edge-node.

Not every topmost y-coordinate of each edge denotes the top of the polygon. To solve this we divide the polygon in trapezia. Now every topmost y-coordinate of each edge denotes the top of a trapezium (disregarding the horizontal edges). When we divide Figure 1 into trapezia keeping the intensities mentioned at Paragraph 3.2. Page 6 we get Figure 7. The polygon now consists out of two triangles and one trapezium.
The new edge-list is depicted in Figure 8.

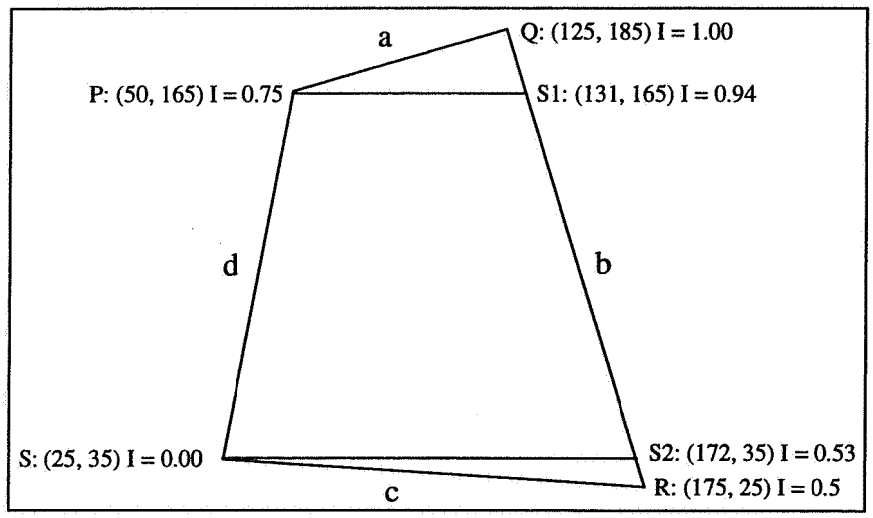


Figure 7: the polygon divided into trapezia

The algorithm for the building of the edge-list stays largely the same. With the difference that the Ymin field of the edge-nodes should be initialized with the exact value of the bottom y-coordinate instead of the rounded value. The fields Ymax and Ytop of the edge-nodes must be initialized with the exact values of the topmost y-coordinate of their edge.
In the scan-line algorithm again only step 4.2. changes. Step 4.2. now will contain the procedure DO_FILL which will be described later. The updating of the edge-nodes in the AET now includes one step more. This is the step in which the value of the field Ymax is decreased by 1.

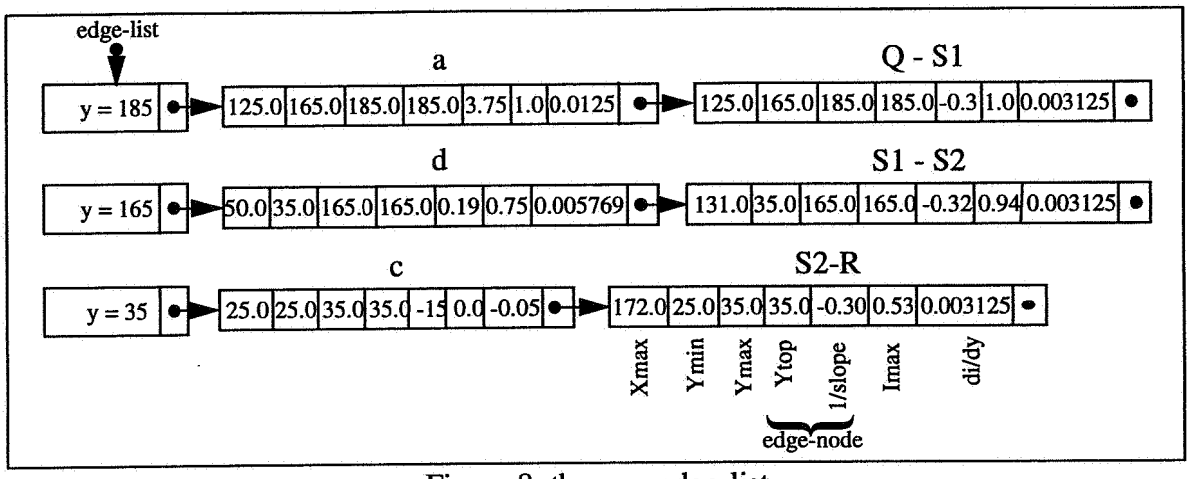


Figure 8: the new edge-list

3.3.1. The DO_FILL procedure

We consider the centre of a pixel to lie on whole coordinate values of the x and y coordinate of the pixel. This is depicted in Figure 9.

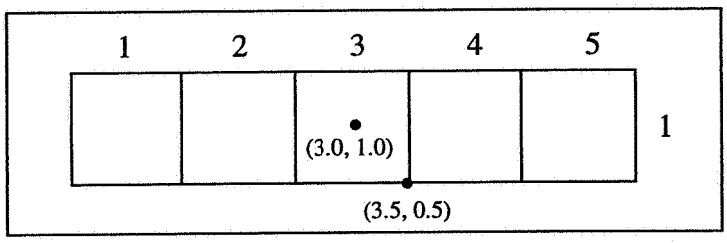


Figure 9: The centre of a pixel

We have to calculate the area of the trapezium that covers the pixels in one pixel row. To do this we first have to find out if we are at the top, in the middle or at the bottom of the trapezium. We are starting at the top of the trapezium if the Ymax value of one of the edge-nodes is equal to the Ytop value. We are working at the bottom of a trapezium if the rounded Ymin value of one of the active edges is equal to the current y-coordinate. If neither of the two conditions is true we are working somewhere in the middle of the trapezium. If both conditions are true, then the whole trapezium is falling in one pixel row.
An example is shown in Figure 10.

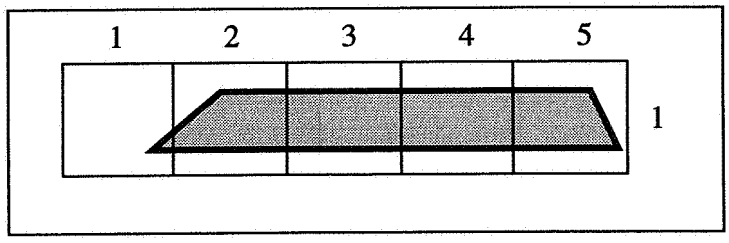


Figure 10: A trapezium lying in one pixel row

For each of the pixels in the considered pixel row we have to find out how much the trapezium is contributing to that pixel. We can do that by simply calculating the surface of the piece of

trapezium covering a pixel. To avoid calculating the covered area for every single pixel we can use the fact that in some parts of the pixel row the covered area increases or decreases with a fixed amount. Still some of the pixels must be calculated separately. To make use of coherency, we have to find out in which pixels the covered area changes continuously.
If we make a difference between an edge cutting through one pixel in a pixel row and an edge cutting through more pixels in a pixel row, then there are four possibilities for two opposite edges to lie in a pixel row. Namely: They both cut through one pixel, they both cut through more pixels, the first cuts through more than one pixel and the second cuts through one pixel, or the first cuts through one pixel and the second cuts through more than one pixel.
The two edges can either meet each other at a common pixel or lie further from each other. When incorporating this, there are two possibilities for each of the four previous possibilities, which gives us eight possibilities for two opposite edges to lie in a pixel row.
If both edges cut through more pixels there is a possibility that they share more than one pixel together. They can either share pixels in the beginning, share pixels at the end, or share pixels in the middle or even share all the pixels from the beginning of the edges till the end. This gives another four extra possibilities which leaves us with twelve ways in which two edges can lie in one pixel row. All these possibilities are treated differently. We will now discuss them one by one.

1. Both edges of the trapezium cut through one pixel and they share a common pixel.

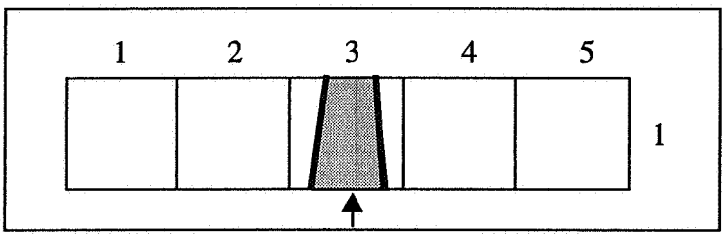


Figure 11: both edges cut through one pixel.

In this case we only need to calculate the covered area of this one pixel.

2. Each of the edges cuts through one pixel, but the edges don't share a common pixel.

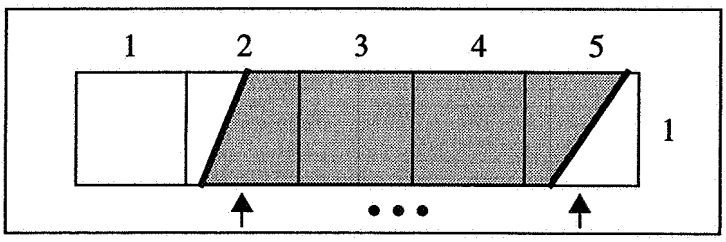


Figure 12: Each of the edges cuts through one pixel.

Here we have to calculate the covered area of the begin and end pixel. Of the middle pixels (if there are any) we only have to calculate one covered area, because the other pixels have the same covered area. The covered area of the middle pixels here is not necessarily 100%. If we are scanning the top or the bottom of the trapezium the covered area is less than 100%. If we have calculated the covered area of the left most 'middle pixel' then every other 'middle pixel' gets the amount of intensity of its left neighbour. Lets call this procedure the **middle-part-procedure**. This procedure will be discussed in more detail later.

3. The first edge cuts through more than one pixel, the second edge doesn't. The two edges share a common pixel

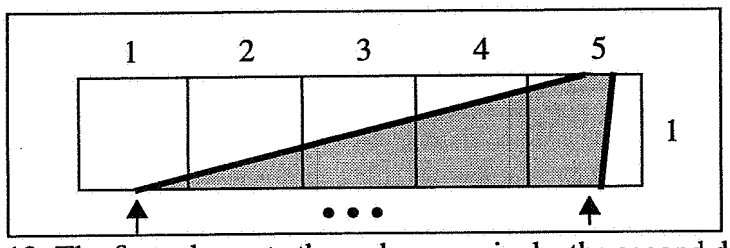


Figure 13: The first edge cuts through more pixels, the second doesn't.

Again we have to calculate the covered area of the begin and end pixel. From the pixels between these two pixels we only have to calculate the first one. The covered area of the pixel to the right of the known middle pixel is the covered area of that pixel plus the absolute value of 1 divided by (2 * the 1/slope field of the first edge). We call this procedure the **sequence-procedure**. Also this procedure will be discussed later in more detail.

4. The first edge cuts through more than one pixel, the second edge doesn't. The two edges don't share a common pixel.

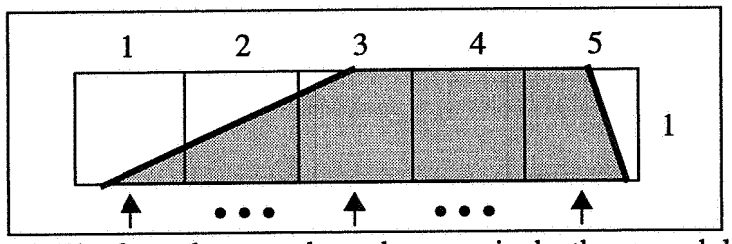


Figure 14: The first edge cuts through more pixels, the second doesn't.

From the first edge we have to calculate the area covered of its first pixel and the area covered of its last pixel. In between these two pixels we can use the sequence-procedure. The area covered of the last pixel must also be calculated separately. Between the last pixel of the first edge and the last pixel covered by the trapezium we can apply the middle-part-procedure.

5. The first edge cuts through one pixel, the second edge cuts through more than one pixel. The two edges share a common pixel.

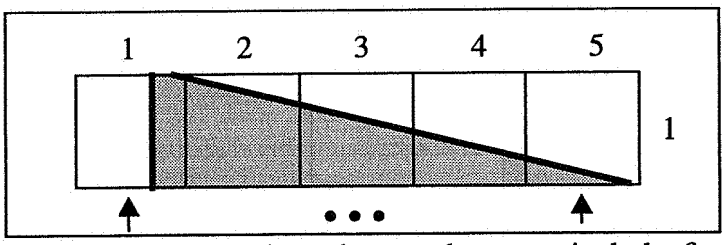


Figure 15: The second edge cuts through more than one pixel, the first one doesn't.

The first and last pixel should be calculated separately. In between these two we can use the sequence-procedure, except that we use minus the absolute value of 1 divided by (2 * the 1/ slope field of the second edge) to add to the first calculated pixel of the sequence.

6. The first edge cuts through one pixel, the second edge cuts through more than one pixel.

The two edges don't share a common pixel.

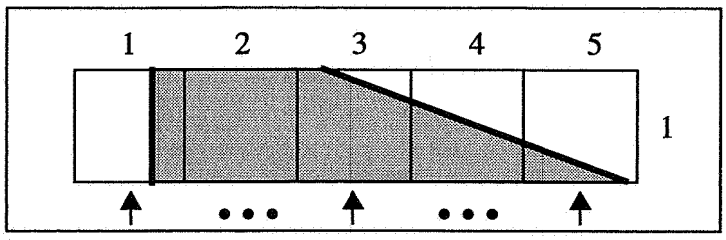


Figure 16: The second edge cuts through more than one pixel, the first doesn't.

The first pixel and the first and last pixel of the second edge should be calculated separately. Between the first pixel and the first pixel of the second edge the middle-part-procedure can be used. Between the first and last pixel of the second edge the sequence-procedure can be used.

7. Both edges cut through more than one pixel. They share a common pixel.

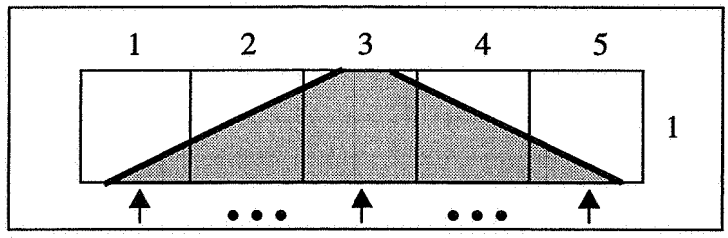


Figure 17: Both edges cut through more pixels and have one pixel in common.

Here we have to calculate the first and last pixel separately and the pixel in which the two edges intersect. Between those pixels we can use the appropriate sequence-procedure.

8. Both edges cut through more than one pixel. They don't share a common pixel.

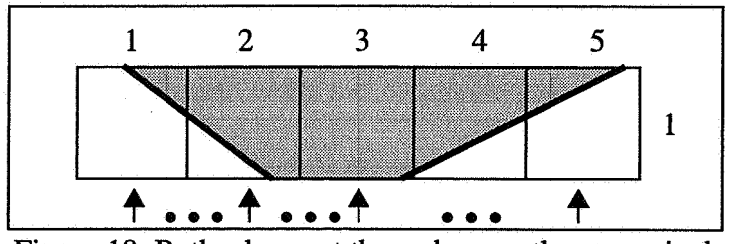


Figure 18: Both edges cut through more than one pixel.

Here we have to calculate the first and last pixel of the first edge and the first and the last pixel of the second edge. Between the end pixels of the first edge we can use the appropriate sequence-procedure. The same holds for the end pixels of the second edge.
Between the last pixel of the first edge and the first pixel of the second edge the middle-part-procedure can be used.

9. Both edges cut through more than one pixel. They share pixels in the beginning.

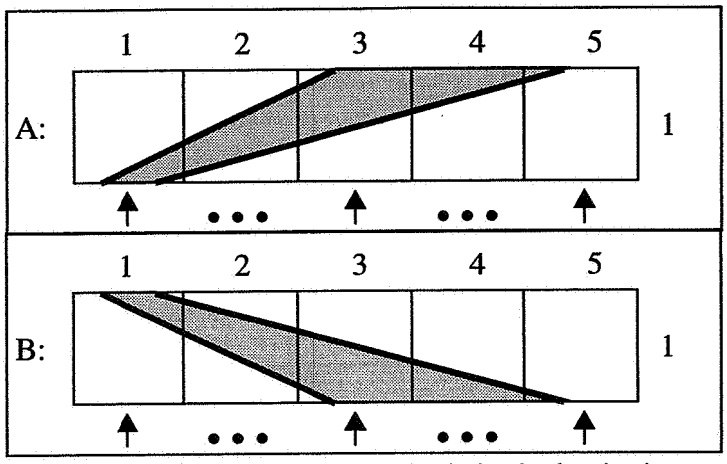


Figure 19: The edges share pixels in the beginning.

The first pixel, the last pixel of the first edge and the last pixel of the last edge should be calculated separately. Between the first pixel and the last pixel of the first edge we only have to calculate one pixel. The pixel right to the calculated pixel is the covered area of that pixel + ((1 divided by (2 * the 1/slope field of the first edge)) - (1 divided by (2 * the 1/slope field of the second edge))). In case B, instead of dividing 1 by (2 * the 1/slope field) we have to divide -1. This procedure is called the **share_sequence_procedure**. Between the last pixel of the first edge and the last pixel of the second pixel we can use the appropriate sequence-procedure.

10. Both edges cut through more than one pixel. They share pixels in the middle.

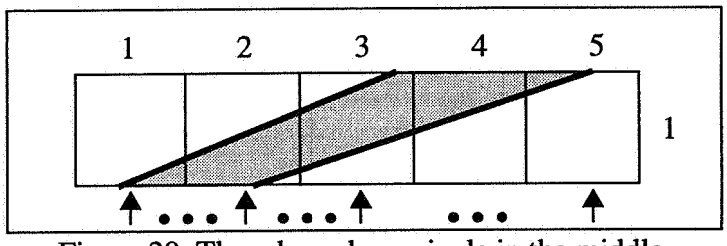


Figure 20: The edges share pixels in the middle.

Here we have to calculate the begin and end pixels of both edges separately. We can use the sequence-procedure between the first pixel of the first edge and the first pixel of the second edge, and between the last pixel of the first edge and the last pixel of the second edge. Between the first pixel of the second edge and the last pixel of the first edge the share_sequence_procedure can be used.

11. Both edges cut through more than one pixel. They share pixels at the end.

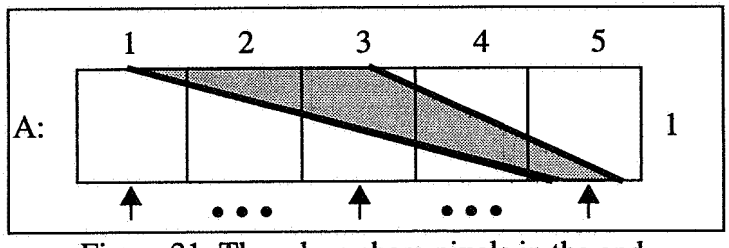


Figure 21: The edges share pixels in the end.

The first pixel and the first and last pixel of the second edge should be calculated separately. Between the first pixel and the first pixel of the second edge the sequence-procedure can be applied. Between the first and the last pixel of the second edge the share_sequence_procedure can be used.

12. Both edges cut through more than one pixel. They share pixels from the beginning till the end of the edges.

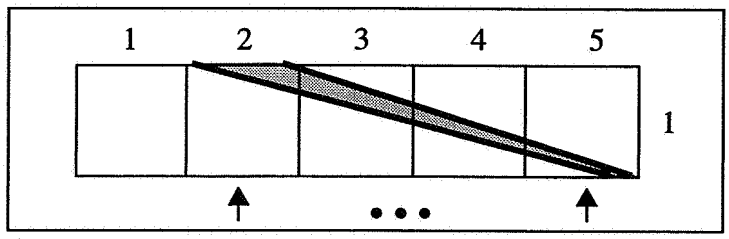


Figure 20: The edges share pixels from the beginning till the end of the edges.

Here only the first and the last pixel have to be calculated separately. In between those two the share_sequence_procedure can be used.

All twelve above mentioned cases make use of three procedures. Namely the middle-part-procedure, the sequence-procedure and the share-sequence-procedure. Each of these procedures can be represented by one rule from the hardware architecture. Suppose we have an AET with two edge-nodes. The rules are then the following:

```
Let X_left be the value of the Xmax field of the first edge-node in the AET.
Let X_right be the value of the Xmax field of the second edge-node in the
   AET.
Let dx be (X_right - X_left).
Let I_left be the value of the Imax field of the first edge-node in the AET.
Let I_right be the value of the Imax field of the second edge-node in the
   AET.
Let di be (I_right - I_left)/dx
```

middle-part-procedure:

```
Let FPx be the value of the x-coordinate of the first pixel in the middle-
   part procedure
Let FPi be the value of the intensity of the first pixel in the middle-part
   procedure (as if the pixel had been completely covered)
Let AC be the area covered of the first pixel in the middle-part procedure.
Let N be the number of pixels in the middle_part procedure.

Eval2(FPx, N, (FPi * AC), (di * AC))
```

sequence-procedure:

```
Let FPx be the value of the x-coordinate of the first pixel in the sequence-
   procedure
Let FPi be the value of the intensity of the first pixel in the sequence-
   procedure (as if the pixel had been completely covered)
Let AC be the area covered of the first pixel in the sequence-procedure
Let N be the number of pixels in the sequence-procedure
Let S be 1 divided by the value of (2 * the 1/slope field of the appropriate
   edge-node)
```

```
Eval3(FPx, N, (FPi * AC), ((di * S) + (di * AC) + (S * FPi)), (2 * di * S))
```

share-sequence-procedure:

```
Let FPx be the value of the x-coordinate of the first pixel in the share-
   sequence-procedure
Let FPi be the value of the intensity of the first pixel in the share-
   sequence-procedure (as if the pixel had been completely covered)
Let AC be the area covered of the first pixel in the share-sequence-proce-
   dure
Let N be the number of pixels in the share-sequence-procedure
Let S1 be 1 divided by (2 * the value of the 1/slope field of the first
   edge-node in the AET)
Let S2 be 1 divided by (2 * the value of the 1/slope filed of the second
   edge-node in the AET)
Let S be (S1 - S2)

Eval3(FPx, N, (FPi * AC), ((di * S) + (di * AC) + (S * FPi)), (2 * di * S))
```

The parameters of the Eval2 call for the middle-part-procedure are easy to infer. The parameters for the Eval3 call are inferred as follows. Suppose S is the amount of covered area which is added to the covered area of a previous pixel.
Then the function for calculating the intensity for a pixel is as follows:
$(FPi + (di * X)) * (AC + (S * X)) =$
$(di * S * X^2) + (((FPi * S) + (di * AC)) * X) + (FPi * AC)$
The calculation of the first order difference is then:
$(di * S * (t + 1)^2) + (((FPi * S) + (di * AC)) * (t + 1)) + (FPi * AC) -$
$(di * S * t^2) - (((FPi * S) + (di * AC)) * t) - (FPi * AC) =$
$(2 * di * S * t) + (di * S) + (FPi * S) + (di * AC)$
The calculation of the second order difference is then:
$(2 * di * S * (t + 1)) + (di * S) + (FPi * S) + (di * AC) -$
$(2 * di * S * t) - (di * S) - (FPi * S) - (di * AC) =$
$2 * di * S$

Picture 23 shows the result of the algorithm described above.

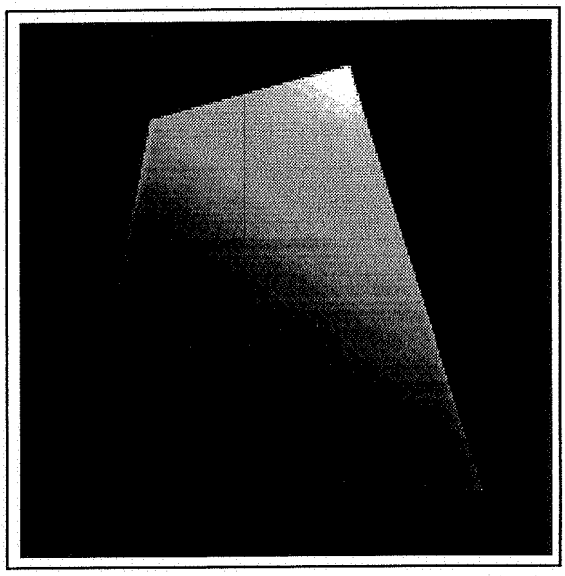


Figure 23: gouraud shading with antialiasing

If we are at the top or/and the bottom of a trapezium the calculations are in great extent the same. The only thing we have to consider is that the bottom part and/or the top part of the pixels in a pixel row is not covered at all.

3.4. Shading a three dimensional object

Until now we have talked about the shading of one trapezium. If we want to shade a three dimensional object consisting out of several trapezia, we have to adjust the algorithm a bit. We also have to change the way the edge_list and the AET are built up.
In figure 24 a wire frame of a three dimensional pot is shown. This pot consists out of triangles and trapezia situated in a three dimensional space. The intensities at the corners of the trapezia are calculated using the formula mentioned on Page 2.

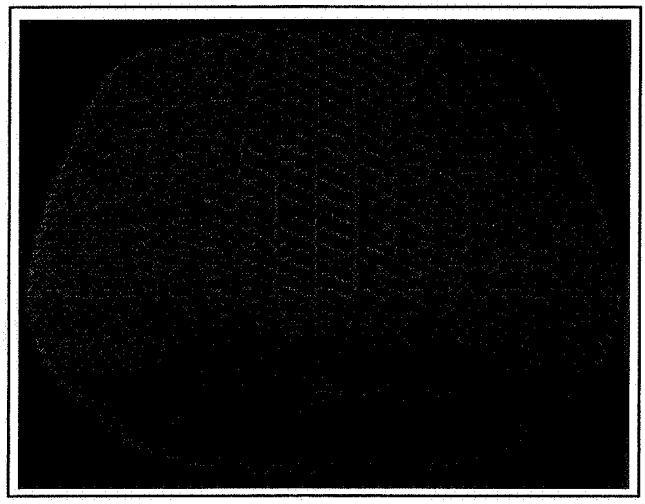


Figure 24 wire frame of a pot

3.4.1. The algorithm for more than one trapezium

Because a scan-line can now intersect with more than one trapezium, the AET will include edge-nodes of more than one trapezium. To indicate which edge-node belongs to what trapezium, we add a new field called **Identifier.**
Originally edge-nodes in the AET were sorted on their Xmax values. This is no longer appropriate, which will be illustrated with an example.
If we look at figure 25 we see three triangles: P, Q and R. The end points of the triangles are the following: A = (2.0, 12.0), B = (10.0, 12.0), C = (10.0, 4.67), D = (2.0, 6.0), E = (8.55, 6.0) and F = (14.0, 1.0). For P the Identifier is 1, for Q it is 2 and for R it is 3. The point where it goes wrong is when current_y has the value 6. To illustrate this we will show the AET sorted on the Xmax fields in Figure 26.
If we would shade between each pair of edge-nodes in this AET, we would shade between an edge of triangle P and one of R, then between an edge of P and one of Q and last between an edge of R and one of Q. This is clearly not correct. To take care of this problem the AET will be sorted in such a way that edges of the same trapezium are kept together. The field Identifier can be used to accomplish this. The new AET for the example will then look as in Figure 27.

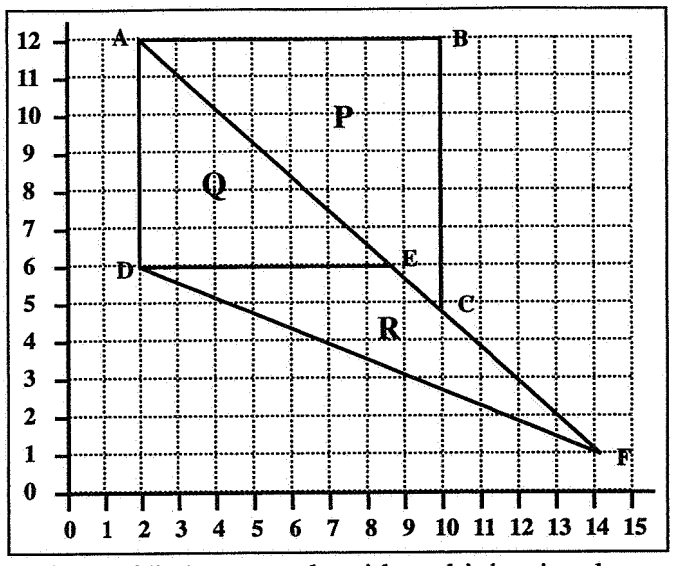


figure 25: An example with multiple triangles

Identifier	1
Xmax	2.0
Ymin	6.0
Ymax	6.0
Ymax_stay	12.0
1/slope	0.0
Imax	1.0
di/dy	0.0

Identifier	3
Xmax	2.0
Ymin	1.0
Ymax	6.0
Ymax_stay	6.0
1/slope	-2.4
Imax	1.0
di/dy	0.0

Identifier	1
Xmax	8.55
Ymin	6.0
Ymax	6.0
Ymax_stay	12.0
1/slope	-1.09
Imax	1.0
di/dy	0.0

Identifier	2
Xmax	8.55
Ymin	4.67
Ymax	6.0
Ymax_stay	12.0
1/slope	-1.09
Imax	1.0
di/dy	0.0

Identifier	3
Xmax	8.55
Ymin	1.0
Ymax	6.0
Ymax_stay	6.0
1/slope	-1.09
Imax	1.0
di/dy	0.0

Identifier	2
Xmax	10
Ymin	4.67
Ymax	6.0
Ymax_stay	12.0
1/slope	0.0
Imax	1.0
di/dy	0.0

figure 26: The AET sorted on the Xmax value of its edge nodes

Identifier	1
Xmax	2.0
Ymin	6.0
Ymax	6.0
Ymax_stay	12.0
1/slope	0.0
Imax	1.0
di/dy	0.0

Identifier	1
Xmax	8.55
Ymin	6.0
Ymax	6.0
Ymax_stay	12.0
1/slope	-1.09
Imax	1.0
di/dy	0.0

Identifier	3
Xmax	2.0
Ymin	1.0
Ymax	6.0
Ymax_stay	6.0
1/slope	-2.4
Imax	1.0
di/dy	0.0

Identifier	3
Xmax	8.55
Ymin	1.0
Ymax	6.0
Ymax_stay	6.0
1/slope	-1.09
Imax	1.0
di/dy	0.0

Identifier	2
Xmax	8.55
Ymin	4.67
Ymax	6.0
Ymax_stay	12.0
1/slope	-1.09
Imax	1.0
di/dy	0.0

Identifier	2
Xmax	10
Ymin	4.67
Ymax	6.0
Ymax_stay	12.0
1/slope	0.0
Imax	1.0
di/dy	0.0

figure 27: The AET with edges of the same trapezium kept together

The algorithm to construct the edge_list should now initialize the Identifier field and sort on this field. This algorithm can be found in appendix A page A7-8. The algorithm for rendering with Gouraud shading and antialiasing stays the same. The only difference is that now the AET will be sorted first on the Identifier field and then on the Xmax field of the edge-nodes. This algorithm can be found in appendix A page A9. The object of figure 24 with known intensities at the corners of the trapezia, shaded with this last algorithm is shown in Figure 28.

4. CONCLUDING REMARKS

In this paper we discussed implementations of scan-line algorithms for antialiasing and shading algorithms. These implementation included linear interpolation and antialiasing with the box filter. We also showed how the scan-line algorithm could be adjusted for dealing with more than one trapezium.
We have shown that procedures used for shading and antialiasing could be represented by one

command for a X_processor of the hardware architecture for each individual trapezium. Future work will focus on incorporating this implementation in a larger scale system where also higher order interpolation will be included. Example of this are Phong shading or vector rotated based shading [Kuijk et al., 89].

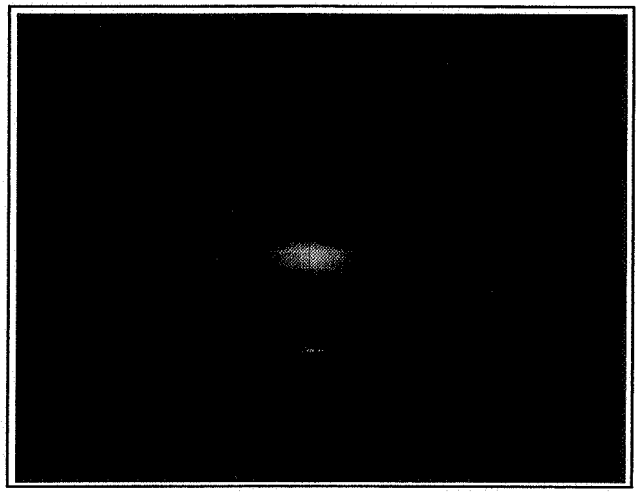


Figure 28: the shaded pot

REFERENCES

[Akman et al., 88] Akman, V., P.J.W. ten Hagen and A.A.M. Kuijk, "*A vector-like architecture for raster graphics*", Report CS-R8802, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[Blinn, 89a] Blinn, J.F., "*What we need around here is more aliasing*", IEEE CG & A, Vol. 9, No. 1, January 1989, Pages 75-79.

[Blinn, 89b] Blinn, J.F., "*Return of the Jaggy*", IEEE CG & A, Vol. 9, No. 2, March 1989, Pages 82-89.

[Crow, 77] Crow, F.C., "*The Aliasing Problem in Computer-Generated Shaded Images*", Communications of the ACM, Vol. 20, No 11, November 1977, Pages 799-805.

[Crow, 81] Crow, F.C., "*A Comparison of Antialiasing Techniques*", IEEE CG & A, Vol. 1, No. 1, January 1981, Pages 40-48.

[Foley et al., 90] Foley, J.D., A. van Dam, S.K. Feiner and J. F. Hughes. "*Computer Graphics: Principles and Practice*", 2nd edition. Addison-Wesley, 1990, Pages 96-99.

[Gouraud, 71] Gouraud, H., "*Continuous Shading of Curved Surfaces*", IEEE, Transactions on Computers, Vol. c-20, No. 6, June 1971, Pages 623-629.

[Gupta et al., 81] Gupta, S. and R.F. Sproull, "*Filtering Edges for Gray-Scale Displays*", Computer Graphics, Vol. 15, No. 3, August 1981, Pages 1-5.

[Hagen et al., 86] ten Hagen, P.J.W., A.A.M. Kuijk and C.G. Trienekens, "*Display Architecture for VLSI-based graphics workstations*", Report CS-R8637, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[Kuijk et al., 89] Kuijk, A.A.M. and E.H. Blake, "*Faster Phong Shading via Angular Interpolation*", Computer Graphics Forum, Vol. 8, No. 4, December 1989, Pages 315-324.

[Kuijk et al., 92] Kuijk, A.A.M., E.H. Blake and P.J.W. ten Hagen, "*An architecture for interactive raster graphics*", Report CS-R9229, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[Phong, 75] Phong, B.T., "*Illumination for Computer Generated Pictures*", Communications of the ACM, Vol. 18, No. 6, June 1975, Pages 311-317.

Algorithm 1. The building of the edge-list

el-pointer: pointer to edge-list-node
edge-list: pointer to edge-list-node

1. Initialize **edge-list** to be empty
2. for (each edge of the polygon) do:
   Let **topmost_x** be the value of the topmost x-coordinate of the current edge.
   Let **topmost_y** be the value of the topmost y-coordinate of the current edge.
   Let **bottom_x** be the value of the bottom x-coordinate of the current edge.
   Let **bottom_y** be the value of the bottom y-coordinate of the current edge.

   2.1. Make a new edge-node.
   2.2. Set the Xmax field of the new edge-node to **topmost_x**.
   2.3. Set the Ymin field of the new edge-node to **bottom_y** rounded to an integer.
   2.4. Set the 1/slope field of the new edge-node to the value of (**bottom_x** - **topmost_x**) / (**bottom_y** - **topmost_y**).
   2.5a. If (edge-list is empty) then:
      2.5a.1. Make a new edge-list-node
      2.5a.2. Set the Ymax field of the new edge-list-node to **topmost_y** rounded to an integer.
      2.5a.3. Insert the new edge-node in the list pointed to by the enp field of the new edge-list-node.
      2.5a.4. Insert the new edge-list-node in **edge-list** sorted on its Ymax field.
   2.5b. Else:
      2.5b.1. Set **el-pointer** to **edge-list**.
      2.5b.2. While (there is a next edge-list-node) AND (the Ymax field of the current edge-list-node is bigger than **topmost_y**) do:
         2.5b.2.1. Set **el-pointer** to the next edge-list-node in **edge-list**.
      2.5b.3a. If (the Ymax field of the current edge-list-node is smaller than **topmost_y**) then:
         2.5b.3a.1. do 2.5a.1. till 2.5a.4.
      2.5b.3b. Else:
         2.5b.3b.1. Insert the new edge-node in the list pointed to by the enp field of the current edge-list-node. This insertion should be done ordered on the Xmax field of the edge-nodes in that list.

Algorithm 2. The scan-line algorithm

el-pointer: pointer to edge-list-node
AET: pointer to edge-node
current_y: integer depicting an y coordinate

1. Set **el-pointer** to the top of the edge-list.
2. Set **current_y** to the value of the Ymax field of the edge-list-node pointed at by **el-pointer.**
3. Initialize **AET** to be empty.
4. Repeat until (the **AET** is empty) OR (the bottom of the edge-list has been reached):
   4.1. If (the value of the Ymax field of the current edge-list-node is equal to **current_y**) then:
      4.1.1. Insert the edge-nodes, in the list pointed at by the enp field of the current edge-node-list, in the **AET**, sorted on the Xmax values of the edge-nodes.
      4.1.2. Set **el-pointer** to the next edge-list-node.
   4.2. For (each pair of edge-nodes in the **AET**) do:
      4.2.1. Fill in the pixels, lying on scan line with its y coordinate equal to **current_y**, between the rounded Xmax values of the current pair of edge-nodes.
   4.3. Remove from the **AET** those edge-nodes which have their Ymin field equal to **current_y**.
   4.4. Decrement **current_y** by 1
   4.5. For each edge-node in the **AET** do:
      4.5.1. Update the Xmax field by substracting the value of the 1/slope field.

Algorithm 3. The building of the edge-list when applying Gouraud shading*

el-pointer: pointer to edge-list-node
edge-list: pointer to edge-list-node

1. Initialize **edge-list** to be empty
2. for (each edge of the polygon) do:
   Let **topmost_x** be the value of the topmost x-coordinate of the current edge.
   Let **topmost_y** be the value of the topmost y-coordinate of the current edge.
   Let **bottom_x** be the value of the bottom x-coordinate of the current edge.
   Let **bottom_y** be the value of the bottom y-coordinate of the current edge.
   Let **topmost_I** be the value of the intensity of the topmost vertex of the current edge
   Let **bottom_I** be the value of the intensity of the bottom vertex of the current edge.

   2.1. Make a new edge-node.
   2.2. Set the Xmax field of the new edge-node to **topmost_x**.
   2.3. Set the Ymin field of the new edge-node to **bottom_y** rounded to an integer.
   2.4. Set the 1/slope field of the new edge-node to the value of (**bottom_x** - **topmost_x**) / (**bottom_y** - **topmost_y**).
   2.5. Set the Imax field of the new edge-node to **topmost_I**.
   2.6. Set the di/dy field of the new edge-node to the value of (**bottom_I** - **topmost_I**) / (**bottom_y** - **topmost_y**).
   2.7a. If (edge-list is empty) then:
      2.7a.1. Make a new edge-list-node
      2.7a.2. Set the Ymax field of the new edge-list-node to **topmost_y** rounded to an integer.
      2.7a.3. Insert the new edge-node in the list pointed to by the enp field of the new edge-list-node.
      2.7a.4. Insert the new edge-list-node in **edge-list** sorted on its Ymax field.
   2.7b. Else:
      2.7b.1. Set **el-pointer** to **edge-list**.
      2.7b.2. While (there is a next edge-list-node) AND (the Ymax field of the current edge-list-node is bigger than **topmost_y**) do:
         2.7b.2.1. Set **el-pointer** to the next edge-list-node in **edge-list**.

      2.7b.3a. If (the Ymax field of the current edge-list-node is smaller than **topmost_y**) then:
         2.7b.3a.1. do 2.5a.1. till 2.5a.4.
      2.7b.3b. Else:
         2.7b.3b.1. Insert the new edge-node in the list pointed to by the enp field of the current edge-list-node. This insertion should be done ordered on the Xmax field of the edge-nodes in that list.

*. The bar shows where the algorithm has changed from algorithm 1

Algorithm 4. The scan-line algorithm when applying gouraud shading*

el-pointer: pointer to edge-list-node
AET: pointer to edge-node
current_y: integer depicting an y coordinate

1. Set **el-pointer** to the top of the edge-list.
2. Set **current_y** to the value of the Ymax field of the edge-list-node pointed at by **el-pointer.**
3. Initialize **AET** to be empty.
4. Repeat until (the **AET** is empty) OR (the bottom of the edge-list has been reached):
   4.1. If (the value of the Ymax field of the current edge-list-node is equal to **current_y**) then:
      4.1.1. Insert the edge-nodes, in the list pointed at by the enp field of the current edge-node-list, in the **AET**, sorted on the Xmax values of the edge-nodes.
      4.1.2. Set **el-pointer** to the next edge-list-node.
   4.2. For (each pair of edge-nodes in the **AET**) do:

      Let **X_left** be the rounded value of the Xmax field of the first edge-node in the current pair of edge-nodes
      Let **X_right** be the rounded value of the Xmax field of the second edg-e_node in the current pair of edge-nodes
      Let **I_left** be the value of the Imax field of the first edge-node in the current pair of edge-nodes
      Let **I_right** be the value of the Imax field of the second edge-node in the current pair of edge-nodes.
      Let **di/dx** be the value of (**I_right** - **I_left**) / (**X_right** - **X_left**)

      4.2.1. While **X_left** is not bigger than **X_right** do:
         4.2.1.1. plot the point with coordinates (**X_left**, **current_y**) in intensity **I_left**.
         4.2.1.2. Increment **X_left** by 1.
         4.2.1.3. Increment **I_left** by **di/dx**.
   4.3. Remove from the **AET** those edge-nodes which have their Ymin field equal to **current_y**.
   4.4. Decrement **current_y** by 1
   4.5. For each edge-node in the **AET** do:
      4.5.1.Update the Xmax field by substracting the value of the 1/slope field.
      4.5.2.Update the Imax field by substracting the value of the di/dy field.

*. The bar shows where the algorithm has changed from algorithm 2

Algorithm 5. The building of the edge-list when applying Gouraud shading and antialiasing with the box filter*

el-pointer: pointer to edge-list-node
edge-list: pointer to edge-list-node

1. Initialize **edge-list** to be empty
2. for (each edge of the polygon) do:
   Let **topmost_x** be the value of the topmost x-coordinate of the current edge.
   Let **topmost_y** be the value of the topmost y-coordinate of the current edge.
   Let **bottom_x** be the value of the bottom x-coordinate of the current edge.
   Let **bottom_y** be the value of the bottom y-coordinate of the current edge.
   Let **topmost_I** be the value of the intensity of the topmost vertex of the current edge
   Let **bottom_I** be the value of the intensity of the bottom vertex of the current edge.

   2.1. Make a new edge-node.
   2.2. Set the Xmax field of the new edge-node to **topmost_x**.
   2.3. Set the Ymin field of the new edge-node to **bottom_y**.
   2.4. Set the 1/slope field of the new edge-node to the value of (**bottom_x** - **topmost_x**) / (**bottom_y** - **topmost_y**).
   2.5. Set the Imax field of the new edge-node to **topmost_I**.
   2.6. Set the di/dy field of the new edge-node to the value of (**bottom_I** - **topmost_I**) / (**bottom_y** - **topmost_y**).
   2.7. Set the Ytop field and the Ymax field of the new edge-node to **topmost_y**.
   2.8a. If (edge-list is empty) then:
      2.8a.1. Make a new edge-list-node
      2.8a.2. Set the Ymax field of the new edge-list-node to **topmost_y** rounded to an integer.
      2.8a.3. Insert the new edge-node in the list pointed to by the enp field of the new edge-list-node.
      2.8a.4. Insert the new edge-list-node in **edge-list** sorted on its Ymax field.
   2.8b. Else:
      2.8b.1. Set **el-pointer** to **edge-list**.
      2.8b.2. While (there is a next edge-list-node) AND (the Ymax field of the current edge-list-node is bigger than **topmost_y**) do:
         2.8b.2.1. Set **el-pointer** to the next edge-list-node in **edge-list**.

      2.8b.3a. If (the Ymax field of the current edge-list-node is smaller than **topmost_y**) then:
         2.8b.3a.1. do 2.5a.1. till 2.5a.4.
      2.8b.3b. Else:
         2.8b.3b.1. Insert the new edge-node in the list pointed to by the enp field of the current edge-list-node. This insertion should be done ordered on the Xmax field of the edge-nodes in that list.

*. The bar shows where the algorithm has changed from algorithm 3

Algorithm 6. The scan-line algorithm when applying gouraud shading and antialiasing with the box filter*

el-pointer: pointer to edge-list-node
AET: pointer to edge-node
current_y: integer depicting an y coordinate

1. Set **el-pointer** to the top of the edge-list.
2. Set **current_y** to the value of the Ymax field of the edge-list-node pointed at by **el-pointer.**
3. Initialize **AET** to be empty.
4. Repeat until (the **AET** is empty) OR (the bottom of the edge-list has been reached):
   4.1. If (the value of the Ymax field of the current edge-list-node is equal to **current_y**) then:
      4.1.1. Insert the edge-nodes, in the list pointed at by the enp field of the current edge-node-list, in the **AET**, sorted on the Xmax values of the edge-nodes.
      4.1.2. Set **el-pointer** to the next edge-list-node.
   4.2. For (each pair of edge-nodes in the **AET**) do:

      4.2.1 Fill in the pixels, lying on scan-line with its y coordinate equal to **current_y**, between the Xmax values of the edge-nodes in the **current pair of edge-nodes**, using the procedure DO_FILL.
   4.3. Remove from the **AET** those edge-nodes which have their Ymin field equal to **current_y**.
   4.4. Decrement **current_y** by 1
   4.5. For each edge-node in the **AET** do:
      4.5.1. Update the Xmax field by substracting the value of the 1/slope field.
      4.5.2. Update the Imax field by substracting the value of the di/dy field.
      4.5.3. Update the Ymax field by substracting 1.

*. The bar shows where the algorithm has changed from algorithm 4

Algorithm 7. The building of the edge-list when applying Gouraud shading and antialiasing with the box filter on more than one polygon*

el-pointer: pointer to edge-list-node
edge-list: pointer to edge-list-node
id: integer

1. Initialize **edge-list** to be empty.
2. Initialize id to 0.
3. For (each polygon of the graphical object) do:
   3.1. **id** = **id** + 1.
   3.2. for (each edge of the polygon) do:
      Let **topmost_x** be the value of the topmost x-coordinate of the current edge.
      Let **topmost_y** be the value of the topmost y-coordinate of the current edge.
      Let **bottom_x** be the value of the bottom x-coordinate of the current edge.
      Let **bottom_y** be the value of the bottom y-coordinate of the current edge.
      Let **topmost_I** be the value of the intensity of the topmost vertex of the current edge
      Let **bottom_I** be the value of the intensity of the bottom vertex of the current edge.

      3.2.1. Make a new edge-node.
      3.2.2. Set the Xmax field of the new edge-node to **topmost_x**.
      3.2.3. Set the Ymin field of the new edge-node to **bottom_y**.
      3.2.4. Set the 1/slope field of the new edge-node to the value of (**bottom_x** - **topmost_x**) / (**bottom_y** - **topmost_y**).
      3.2.5 Set the Imax field of the new edge-node to **topmost_I**.
      3.2.6 Set the di/dy field of the new edge-node to the value of (**bottom_I** - **topmost_I**) / (**bottom_y** - **topmost_y**).
      3.2.7 Set the Ytop field and the Ymax field of the new edge-node to **topmost_y**.
      3.2.8 Set the Identifier field of the new edge-node to the value of **id.**
      3.2.9a. If (edge-list is empty) then:
         3.2.9a.1. Make a new edge-list-node
         3.2.9a.2. Set the Ymax field of the new edge-list-node to **topmost_y** rounded to an integer.
         3.2.9a.3. Insert the new edge-node in the list pointed to by the enp field of the new edge-list-node.
         3.2.9a.4. Insert the new edge-list-node in **edge-list** sorted on its Ymax field.
      3.2.9b. Else:
         3.2.9b.1. Set **el-pointer** to **edge-list**.
         3.2.9b.2. While (there is a next edge-list-node) AND (the Ymax field of the current edge-list-node is bigger than **topmost_y**) do:
            3.2.9b.2.1. Set **el-pointer** to the next edge-list-node in **edge-list**.
         3.2.9b.3a. If (the Ymax field of the current edge-list-node is smaller than **topmost_y**) then:
            3.2.9b.3a.1.do 3.2.9a.1. till 3.2.9a.4.

*. The bar shows where the algorithm has changed from algorithm 3

```
3.2.9b.3b. Else:
   3.2.9b.3b.1.Insert the new edge-node in the list pointed to by
               the enp field of the current edge-list-node. This
               insertion should be done ordered on the Identifier
               field and the Xmax field of the edge-nodes in that
               list.
```

Algorithm 8. The scan-line algorithm when applying gouraud shading and antialiasing with the box filter on more than one polygon*

el-pointer: pointer to edge-list-node
AET: pointer to edge-node
current_y: integer depicting an y coordinate

1. Set **el-pointer** to the top of the edge-list.
2. Set **current_y** to the value of the Ymax field of the edge-list-node pointed at by **el-pointer.**
3. Initialize **AET** to be empty.
4. Repeat until (the **AET** is empty) OR (the bottom of the edge-list has been reached):
   - 4.1. If (the value of the Ymax field of the current edge-list-node is equal to **current_y**) then:
     - 4.1.1. Insert the edge-nodes, in the list pointed at by the enp field of the current edge-node-list, in the **AET**, sorted on the Identifier and Xmax values of the edge-nodes.
     - 4.1.2. Set **el-pointer** to the next edge-list-node.
   - 4.2. For (each pair of edge-nodes in the **AET**) do:
     - 4.2.1 Fill in the pixels, lying on scan-line with its y coordinate equal to **current_y**, between the Xmax values of the edge-nodes in the **current pair of edge-nodes**, using the procedure DO_FILL.
   - 4.3. Remove from the **AET** those edge-nodes which have their Ymin field equal to **current_y**.
   - 4.4. Decrement **current_y** by 1
   - 4.5. For each edge-node in the **AET** do:
     - 4.5.1. Update the Xmax field by substracting the value of the 1/slope field.
     - 4.5.2. Update the Imax field by substracting the value of the di/dy field.
     - 4.5.3. Update the Ymax field by substracting 1.

*. The bar shows where the algorithm has changed from algorithm 6