

# **Computer Graphics**

**Responsible persons:**

**Regula Stopper**

**(Overall)**

**Hansruedi Bär**

**(Content)**

**Olaf Schnabel**

**(Revision)**



# Table Of Content

|  |    |
|--|----|
| 1. Computer Graphics .....                     | 2  |
| 1.1. Screen .....                              | 3  |
| 1.1.1. Have a Look! .....                      | 3  |
| 1.1.2. Pixels and Resolution .....             | 4  |
| 1.1.3. Coordinate System of a Screen .....     | 7  |
| 1.1.4. Unit-Summary .....                      | 7  |
| 1.2. Colours .....                             | 8  |
| 1.2.1. Colour Models .....                     | 8  |
| 1.2.2. Colour Depth .....                      | 9  |
| 1.2.3. Colour Codes .....                      | 10 |
| 1.2.4. Unit-Summary .....                      | 11 |
| 1.3. Rasterisation .....                       | 12 |
| 1.3.1. Setting the Colour of a Pixel .....     | 12 |
| 1.3.2. Line Rasterisation .....                | 13 |
| 1.3.3. Filling Areas .....                     | 16 |
| 1.3.4. Unit-Summary .....                      | 18 |
| 1.4. Anti-Aliasing .....                       | 19 |
| 1.4.1. Staircase Effects .....                 | 19 |
| 1.4.2. How is Anti-Aliasing realized? .....    | 20 |
| 1.4.3. Unit-Summary .....                      | 22 |
| 1.5. Geometric Transformations .....           | 23 |
| 1.5.1. Geometric Transformations .....         | 23 |
| 1.5.2. Homogeneous Coordinates .....           | 27 |
| 1.5.3. Exemplary Computation .....             | 29 |
| 1.5.4. Unit-Summary .....                      | 30 |
| 1.6. Curves .....                              | 31 |
| 1.6.1. Bézier Curve .....                      | 31 |
| 1.6.2. Quadratic and Cubic Bézier Curves ..... | 35 |
| 1.6.3. De Casteljau Algorithm .....            | 37 |
| 1.6.4. Unit-Summary .....                      | 41 |
| 1.7. Paths .....                               | 43 |
| 1.7.1. Definition .....                        | 43 |
| 1.7.2. Creating any Path .....                 | 43 |
| 1.7.3. Presentation of a Path .....            | 44 |
| 1.7.4. Unit-Summary .....                      | 46 |
| 1.8. Self Assessment .....                     | 47 |
| 1.9. Summary .....                             | 48 |
| 1.10. Recommended Reading .....                | 49 |
| 1.11. Glossary .....                           | 50 |
| 1.12. Bibliography .....                       | 51 |

# 1. Computer Graphics

## Learning Objectives

- You will be able to explain how lines, objects or images are drawn on raster displays.
- You will be able to establish the relationship between geometric transformations and homogeneous coordinates.
- You will be able to list the main characteristics of Bézier curves.

## Introduction

Wikipedia, the free encyclopaedia, defines computer graphics as following: "*Computer Graphics is the field of visual computing, where one utilizes computers both to generate visual images synthetically and to integrate or alter visual and spatial information sampled from the real world.*" (Wikipedia)

Almost everyone of us uses computers every day and takes for granted, that its screen displays for example our writing or our images. But scarcely anybody is asking himself why or how a screen displays all these objects correctly or what is needed for doing so.

In this lesson we will learn the main aspects of computer graphics. After this lesson you will understand some of the computer's operations better.



### 1.1. Screen

#### Learning Objectives

- You will be able to chart the coordinate system of the screen.
- You will be able to distinguish between images with a high resolution and images with a low resolution

#### Introduction

On screens all objects are visualised with tiny picture elements, so called pixels. The screen is composed of variable numbers of pixels depending on the screen type. The number of pixels of an image can vary as well depending on the image resolution. In this unit we will deepen these aspects.

Since the coordinate system of a screen differs from the coordinate system of a printed medium that you are normally used to, we will show and explain you the main aspects of the screen's coordinate system as well as the resolution of a screen.



*A monitor*

#### 1.1.1. Have a Look!

Have a look at the following animation. By clicking on the red rectangle you get an enlarged view of a picture's detail.

At first you see a wonderful landscape with beautiful colours. You notice that the image shows a lake with some mountains in the background. But if you zoom in very close you can't identify an object anymore. You only see thousands of tiny coloured elements. This is exactly the way objects are displayed on a screen.

The next chapters will deepen the theory about some aspects related to screens. Some issues you might have already heard in other lectures, but since they are very important for the understanding of the further units, we mention them anyway.

### **1.1.2. Pixels and Resolution**

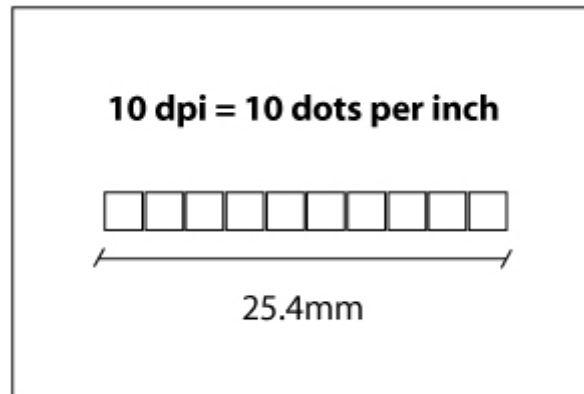
#### **Pixels**

As you could experience in the example above, an image displayed on a screen is composed of thousands of picture elements, so called pixels. The pixel size is about  $0.2 \times 0.2\text{mm}$  on conventional displays or even smaller on high-resolution displays. A typical screen size is about  $1024 \times 768$  pixels.

### Image Resolution

The **resolution** indicates how many pixels (dots) are displayed within an inch (1 inch = 25.4mm).

Example for resolution units:



*Example for Resolution*

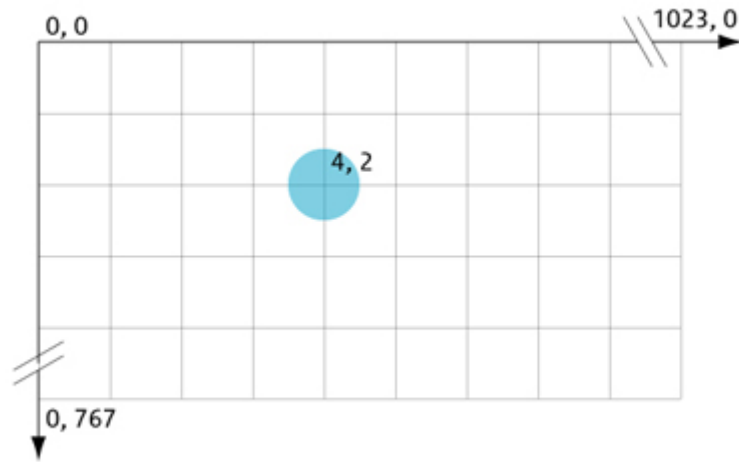
The higher the resolution is, the better is the appearance of fine details.





### 1.1.3. Coordinate System of a Screen

The coordinate system of the screen is a *cartesian coordinate system*<sup>1</sup>. The origin (0,0) is at the top left of the screen. Positive x increases toward the right and positive y increases toward the bottom.



*Coordinate system of a screen*

The pixels are always centred around the grid points.

### 1.1.4. Unit-Summary

In this unit you learned the following facts:

- A typical screen size is about 1024 x 768 pixels.
- An image displayed on a raster screen is composed of pixels. The resolution of an image indicates how many pixels are used to visualise the image.
- The coordinate system of a screen is a cartesian coordinate system. Its origin is at the top left.

---

<sup>1</sup> A cartesian coordinate system is a two dimensional system in which x measures horizontal distance and y measures vertical distance. The pair (x, y) defines every point on the plain.

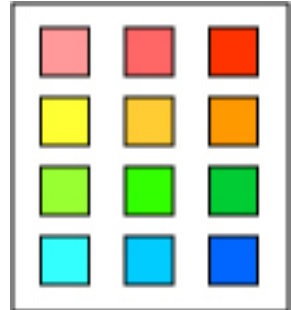
# 1.2. Colours

## Learning Objectives

- You will be able to list three characteristics of the RGB colour model.
- You will be able to differ various colour depths.

The definitions of screen colours differ in those of the printed ones.

In this unit we will show and explain you the main aspects of the definition of screen colours and some other important facts about them. Furthermore we will learn how to encode colours so that the computer knows which colour has to be visualised.



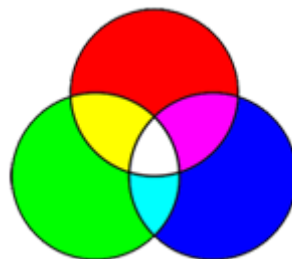
### 1.2.1. Colour Models

A colour model is a mathematical model describing how colours can be represented by sequences of numbers. These numbers are referenced to a certain colour space. There are many different colour spaces in use. In this lesson, we will concentrate only on the colour space which is mainly used to display colours on a computer monitor.

#### RGB Colour Space

In the RGB colour model a colour is represented by three values, giving the amount of red (R), green (G) and blue (B) light. The combination of these 3 colours is utilized to create all other colours.

The RGB colour space is an additive colour schema. The primary colours sum up to white. RGB is a common colour model for computer graphics.



*Additive colour space*

There are other colour-models, such as HSV (Hue, Saturation, Value or Brightness (HSB)) which is also used in computer graphics or CMY(K) (Cyan, Magenta, Yellow, Black) which is a subtractive colour schema and is used for printing colours. We will not go into their details in this lesson. If you're interested in more information about these and other colour spaces have a look at the GITTA lesson "[Colour Design – Colour Models](#)".

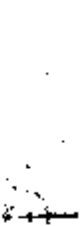




Now you know how to create any arbitrary colour by mixing the primary colours Red Green and Blue (in **chapter 1.3.4.** you will have the possibility to create any arbitrary colour in an interaction part). But if really all possible colours are used to display an image on a screen depends on the colour depth of an image. To understand the correlation between colour depth and the number of used colours in an image, please read the next paragraph.

### 1.2.2. Colour Depth

A bit is the basic information unit used in computing theory and practice. A single bit can represent two states, normally 0 / 1 or true / false. In our case a single bit allows us to distinguish two different colours. Often these will be considered black and white. With  $n$  bits we are able to produce  $2^n$  colours. 4 bits for example allow to represent 16 different colours. Therefore the colour depth of an image tells us how many colours are used to visualise the image.

Very often one *byte* (one byte is composed of 8 bits) is used for each primary colour (e.g. red, green and blue). Thus, the range of every colour is  $2^8 = 256$  and it therefore goes from 0 to 255, whereas 0 stands for the lowest and 255 for the highest intensity. Consequently, by combining all primary colours,  $2^8 * 2^8 * 2^8$  or  $256 * 256 * 256 = 16.777.216$  different colours can be produced.

The following colour depths are most common:

| Colour Depth | Name                           | Code   | Number of Presentable Colours | Example Picture  |
|--------------|--------------------------------|--|-------------------------------|--|
| 1 Bit        | Black and White                | Black: 1 Bit   | 2                             |   |
| 8 Bit        | Greyscale                      | Black: 8 Bit   | 256                           |   |
| 16 Bit       | High Colour                    | Red: 5 Bit<br>Green: 5 Bit<br>Blue: 6 Bit  | $2^{16} = 65'536$             |   |
| 24 Bit       | True Colour                    | 1 Byte (8 Bits) per R, G and B   | $2^{24} = 16'777'216$         |   |
| 32 Bit       | True Colour with 8-Bit channel | 1 Byte (8 Bits) per R, G, B and # (# includes a value for the transparency of the pixel) | $2^{24} = 16'777'216 + \#$    |  |

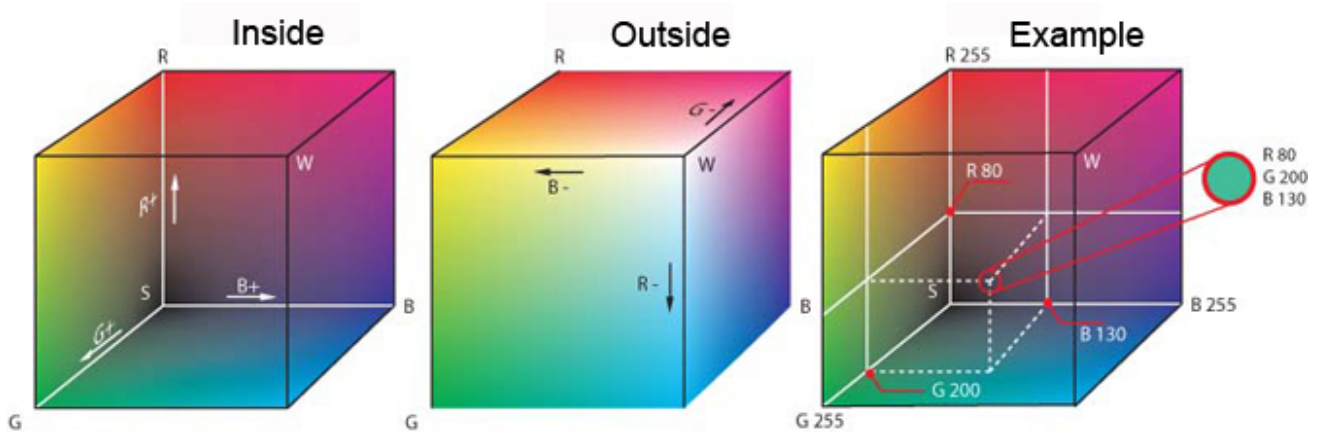
*Table of the most common colour depths*

### 1.2.3. Colour Codes

Since in computer graphics particularly 1 byte per primary colour is used as colour depth, the numbers 0-255 are used to encode all colours of one primary colour, whereas 0 stands for the lowest and 255 for the highest intensity. Below is listed the RGB-Code of some significant colours:

- Black: (0,0,0)
- White: (255,255,255)
- Red: (255,0,0)
- Green: (0,255,0)
- Blue: (0,0,255)

The following picture illustrates how accomplishes the codification of colours in the RGB-Colour space with the numbers 0-255. The grayscale colours are on the diagonal of the cube.



Alternatively, colours can be described using a *hexadecimal*<sup>2</sup> notation. In this case the values range from 00 to FF, whereas 00 stands for the lowest and FF for the highest intensity.

Hexadecimal RGB-code of significant colours:

- Black: #000000
- White: #FFFFFF
- Red: #FF0000
- Green: #00FF00
- Blue: #0000FF

Experience the RGB-Colour-Coding in the following animation tool. You can change the intensity of each primary colour by dragging the sliders to an arbitrary position.



### 1.2.4. Unit-Summary

The following list recapitulates the main aspects of this unit:

- On screens, colours are represented in the RGB (Red, Green, Blue) Colour Model. The RGB colour model is an additive colour schema. Therefore the primary colours sum up to white.
- The colour depth defines how many colours are used in a picture. The basic information unit is a bit, which represents two states.
- To visualise colours on a screen they have to be coded. We have two possibilities to do the encoding: either with numbers or in hexadecimal codes.

<sup>2</sup> A counting system that uses 16 digits, notated as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

### 1.3. Rasterisation

#### Learning Objectives

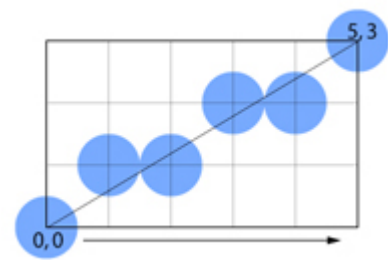
- You will be able to write down the algorithm for colouring a particular pixel of the screen.
- You will be able to choose the right pixels of the screen to draw a continuous line.
- You will be able to explain the different steps of the algorithm for filling polygons.

#### Introduction

Now you know something about colours in the RGB-Colour Space and their codification, but we still do not know how to tell the computer which pixel on the screen has to feature which colour. In this unit we will learn a simple *algorithm*<sup>3</sup> that solves this problem.

Since it is often not efficient to colour each pixel separately - think of colouring a whole screen that features about one million pixels - we will demonstrate you other algorithms for setting the colour of sequences of pixels.

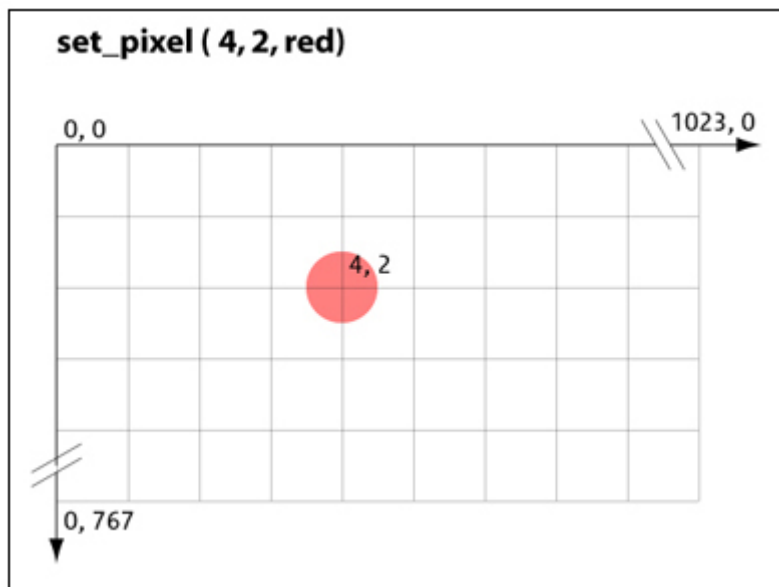
Drawing lines on a raster display is not that simple as it seems to. Imagine drawing a diagonal straight line by only colouring tiny quadratic elements of a cartesian coordinate system. Colouring the right pixels is not always that easy. We will provide you some solutions.



#### 1.3.1. Setting the Colour of a Pixel

There is a simple function that sets the colour of a pixel. You only have to specify the coordinates of the pixel and the colour you want to set:

**set\_pixel (x, y, colour);**



The function for setting the colour of a pixel is the simplest function in computer graphics.

---

<sup>3</sup> An algorithm is a mathematical rule or procedure for solving a problem.

In computer graphics not only setting the colour of a single pixel matters but also setting the colour of a particular group of pixels. Drawing shapes such as rectangles etc. and filling them with a particular colour is such an example.

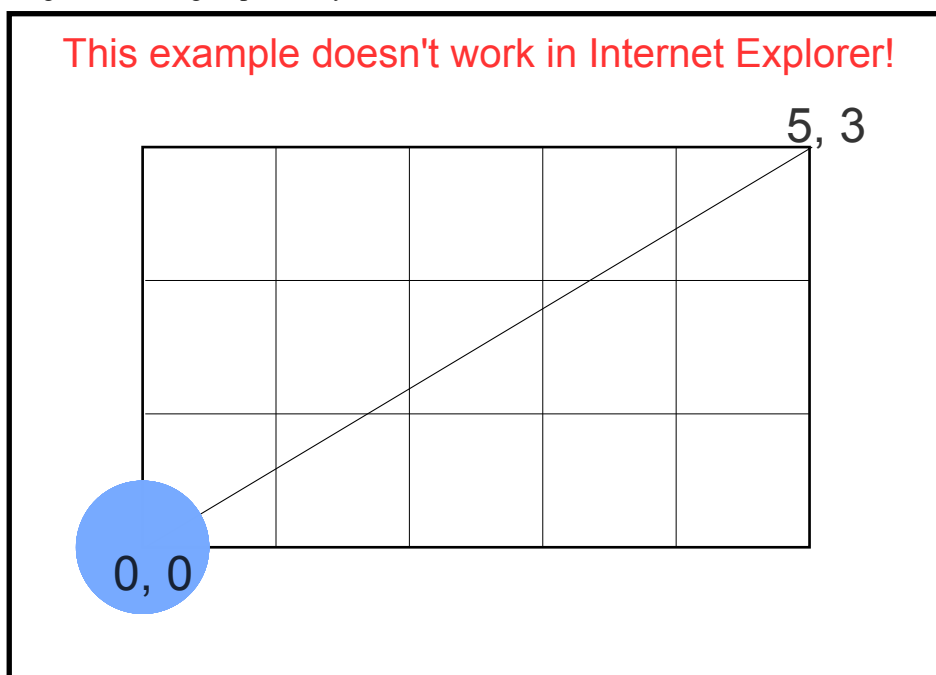
For these and also more complex applications the presented algorithm is too inefficient because it would require considerable processor power and time. Thus higher concepts are necessary, which will be introduced in the following chapters.

### 1.3.2. Line Rasterisation

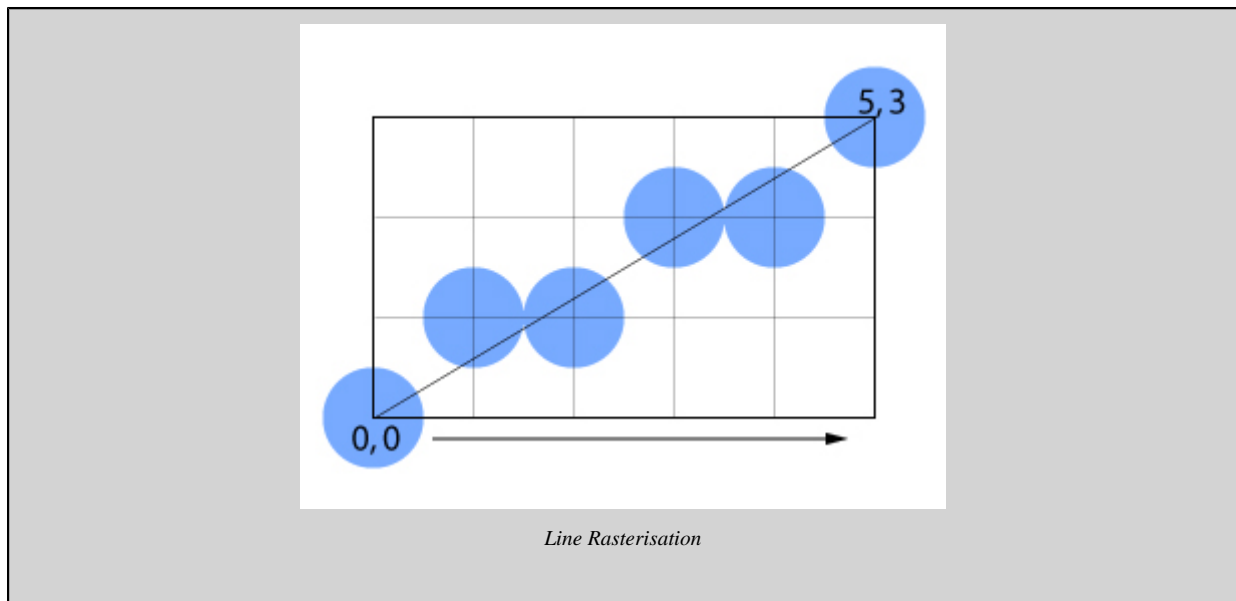
As you know, on a paper sheet, we are able to draw easily a straight line between two points located anywhere on a paper. On raster displays such as a screen, however, we can draw lines only between grid points. The line must be approximated by intensifying the grid-point pixels lying on it or nearest to it. (FOLEY et al. 1996)

But which pixels approximate best the line?

Try to approximate best the line by placing the blue circles on the right position of the grid. Click on the blue circle and drag it to the the grid position you want to. You can set several circles!



Compare your result with the given solution!



### Midpoint Line Algorithm

A simple algorithm for the line approximation may look like this (programmed in C):

```
/*  x0, y0: left endpoint
    x1, y1: right endpoint
*/
void line_to (int x0, int y0, int x1, int y1)
{
    double dx = x1-x0;
    double dy = y1-y0;
    double m = dy/dx;
    double y = y0;
    int x;

    for (x=x0, x < x1; ++x) {
        set_pixel (x, round (y));
        y+= m;
    }
}
```

### Comments on the algorithm

Variables:

- dx defines the difference between the x-coordinates of the two points;
- dy defines the difference between the y-coordinates of the two points;



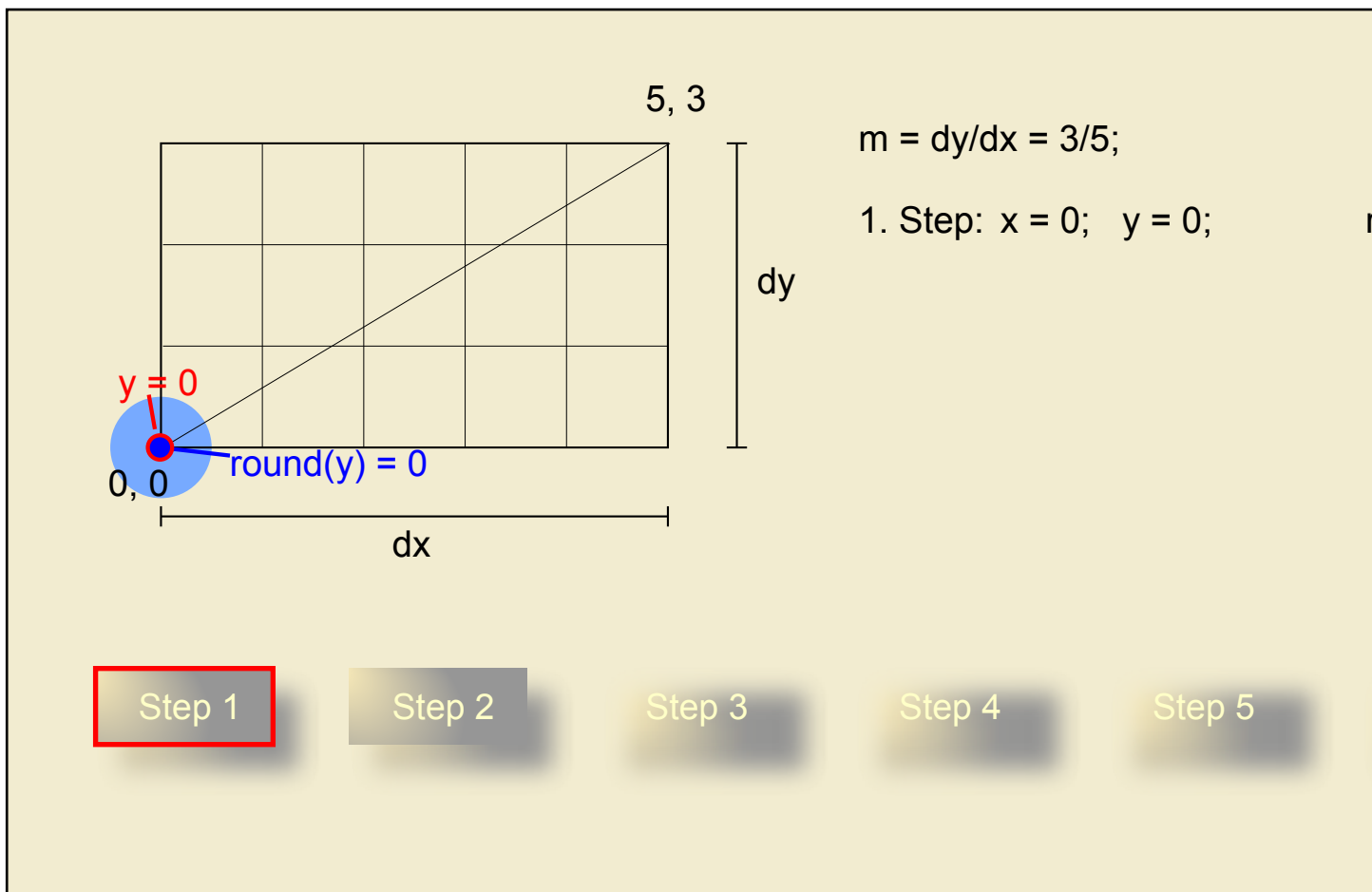
- $m$  defines the slope of the line;
- "double" is a *floating-point number*<sup>4</sup> which is a datatype. Double stands for double precision.
- "int" stands for integer which is also a datatype. Integers consist of positive natural numbers (1, 2, 3, ...).

### For-Loop::

A for loop is a control structure which allows code to be executed iteratively. For loops are typically used when the number of iterations is known before entering the loop. (Wikipedia - The Free Encyclopedia)

- First Step: Set pixel  $x=x_0$ ,  $y=y_0$ ;
- Second Step: Set pixel  $x=x_0+1$ ,  $y=y_0+m$  and round  $y$ ;
- Third Step: Set pixel  $x=x_0+1$ ,  $y=y_0+m$  and round  $y$ ;
- ...

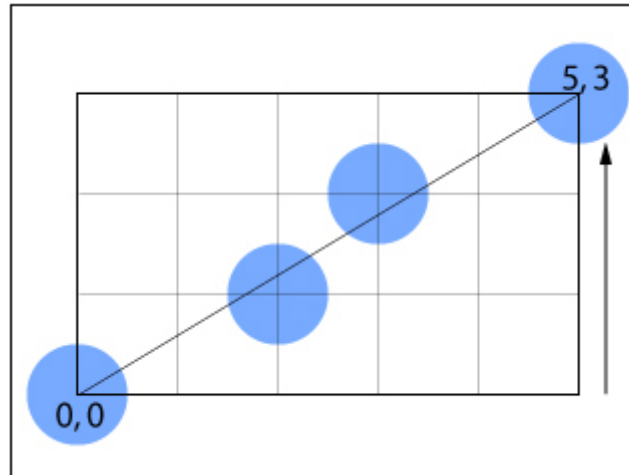
The following animation shows you the different steps of the algorithm in pictures. Have a look!



In the presented algorithm the x-coordinates of the missed pixels are taken as fixed integers (1, 2, 3, etc.). The y-coordinates have to be adapted by rounding them, as you saw it in the above example.

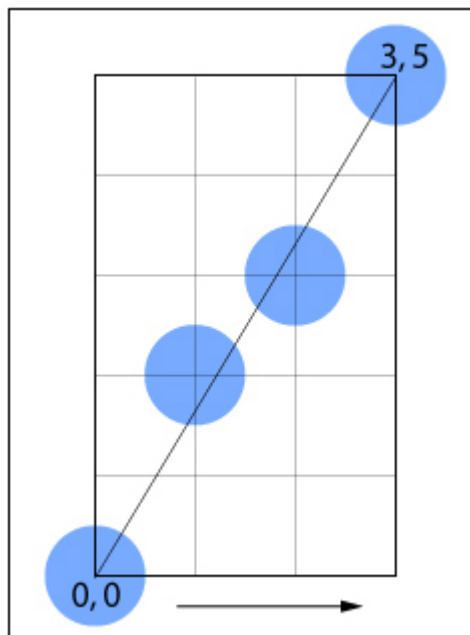
Naturally we can change the case so that the y-coordinates are fixed and the x-coordinates are rounded. The result looks like this:

<sup>4</sup> A floating-point number is a digital representation for a number in a certain subset of the rational numbers. The number of decimal places can be defined individually. Examples 1.235, 1.23, 1.2, etc.

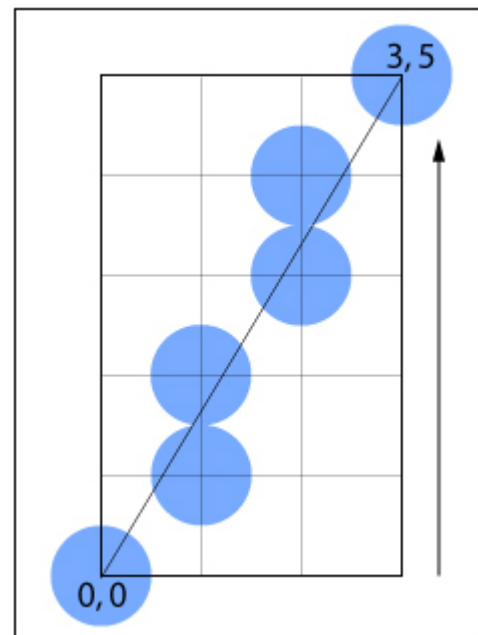


As you notice, there are less pixels defining the line and the gaps between the pixels are larger. Therefore this solution is suboptimal.

But what happens if the slope of the line is greater than one? As you can see in the following examples, the results change:



*x-axis*



*y-axis*

We conclude that the decision which algorithm (e.g.  $x$  is set as fix or  $y$  is set as fix) has to be applied for a line rasterisation depends on the slope value of the line.

Since the introduced algorithm needs a lot of computing for complex applications, it is better to use a more efficient algorithm such as the "Algorithm by Bresenham". If you are interested in more information about this algorithm, have a look at (FOLEY et al. 1996, p. 74-78)

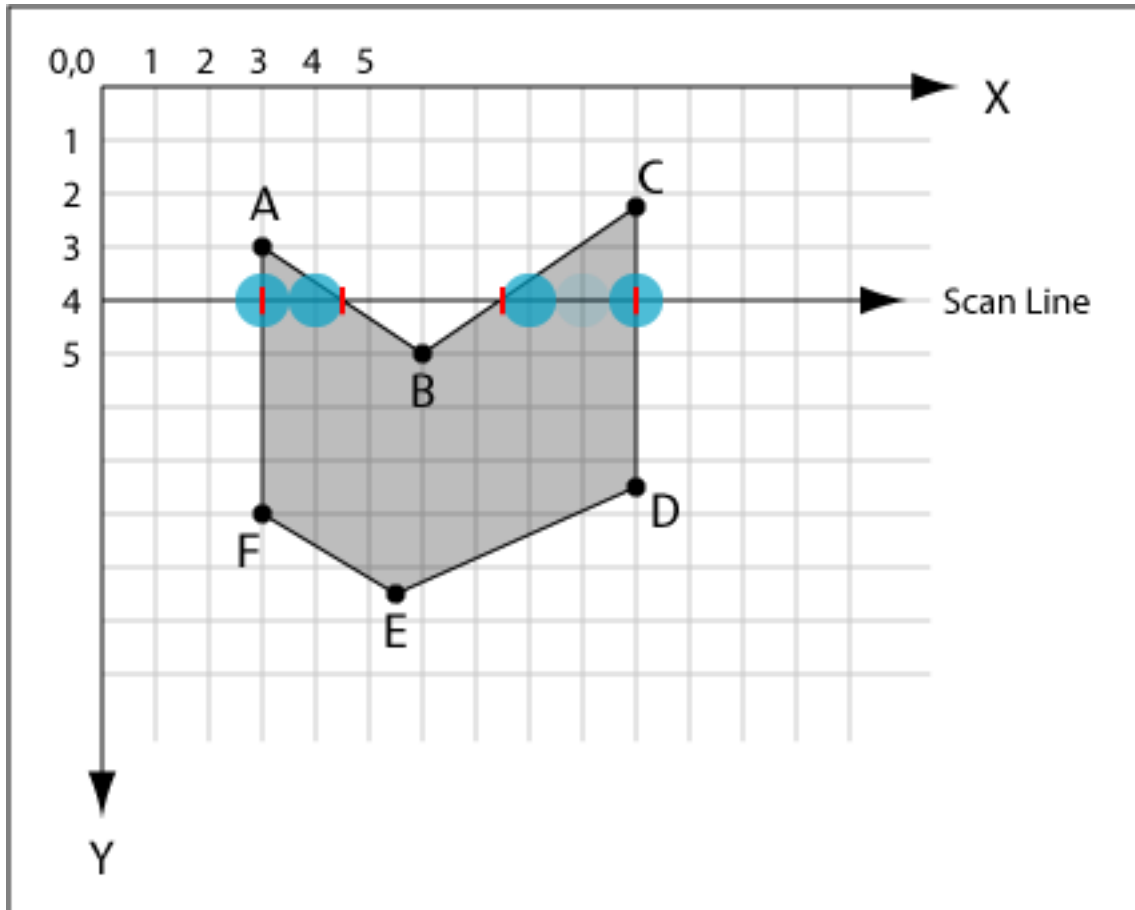
### 1.3.3. Filling Areas

Similarly to the line rasterisation, solid figures such as filled *polygons*<sup>6</sup> or circles are created by intensifying the pixels in their interiors and on their boundaries.

---

<sup>6</sup> A polygon is a closed planar path composed of a finite number of sequential line segments.

We present you an algorithm that operates by computing spans that lie between left and right edges of the polygon. The span extrema are calculated by computing a scan line intersection from the intersection with the previous scan line. Have a look at the next image, which shows a polygon and one scan line passing through it. The intersections of scan 4 with edges AF and CD lie on integer coordinates, whereas those for AB and BC do not; the intersections are marked in the figure by vertical red marks. (FOLEY et al. 1996, p. 92)



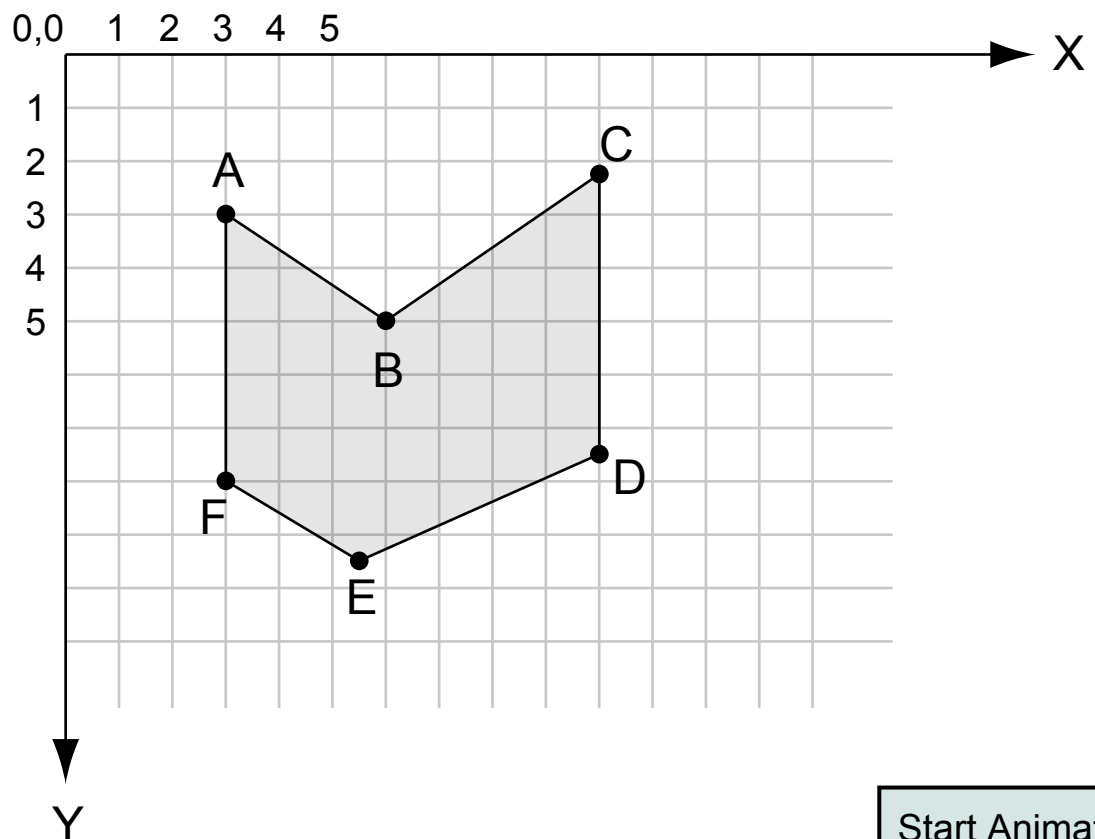
We must determine which pixels on each scan line are within the polygon, and we must set the corresponding pixels to their appropriate values. By repeating this process for each scan line that intersects the polygon, we scan convert the entire polygon. (FOLEY et al. 1996, p. 92-93)

The described algorithm handles both convex and concave polygons, even those that are self-intersecting or have interior holes. (FOLEY et al. 1996, p. 92)

### Algorithm "Filling Polygon" in few words

When filling a polygon, the intersections that lie between left and right edges of the polygon have to be computed. Afterwards, the resulting spans can be filled. (FOLEY et al. 1996, p. 92) According to this, the algorithm for filling polygons features two steps:

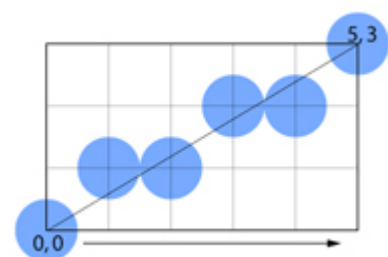
- Computing the intersections for each scan line.
- Filling the polygon alternately between the intersections.



### 1.3.4. Unit-Summary

On a raster display a line is drawn by colouring pixels. Since the pixels are centered around the grid points, we have to approximate the line by intensifying the grid-point pixels lying on the line or nearest to it.

There are various algorithms to find the pixels that approximate best the line. Which algorithm have to be applied depends on the slope of the line.



The algorithm for filling polygons contains two steps:

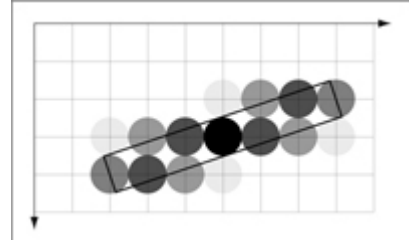
- Computing the intersections for each scan line
- Filling the polygon alternately between the intersections.

# 1.4. Anti-Aliasing

## Learning Objectives

- You will be able to explain the staircase effects of a picture.
- You will be able to give a summary of the principle of Anti-Aliasing.
- You will be able to distinguish between anti-aliased and aliased pictures.

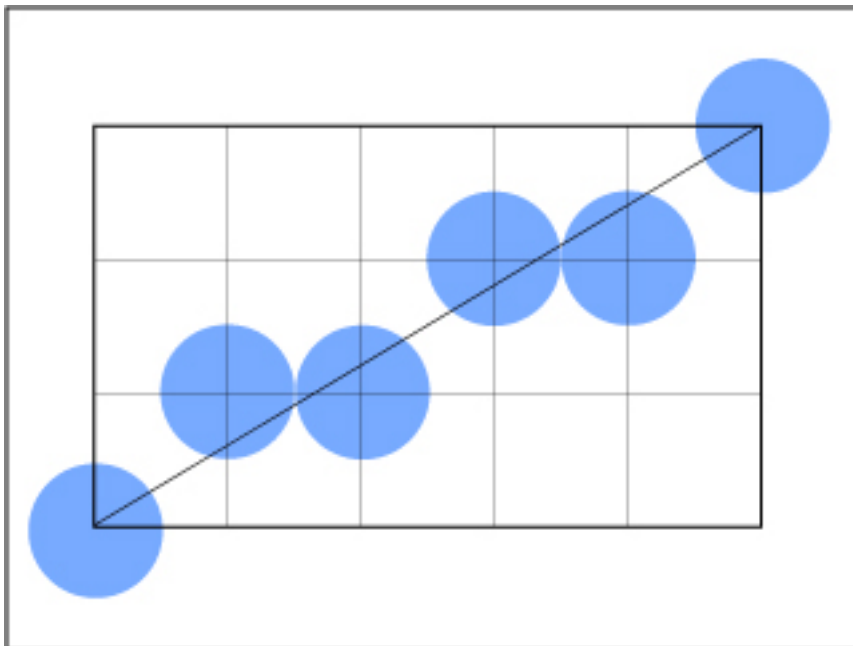
You were certainly already confronted with the expression "Anti-Aliasing" or staircase effects. But do you know exactly what staircase effects are or what happens to a picture when anti-aliasing is applied to it? If so, you can skip this chapter. Otherwise the next two units give you the answers for these questions.



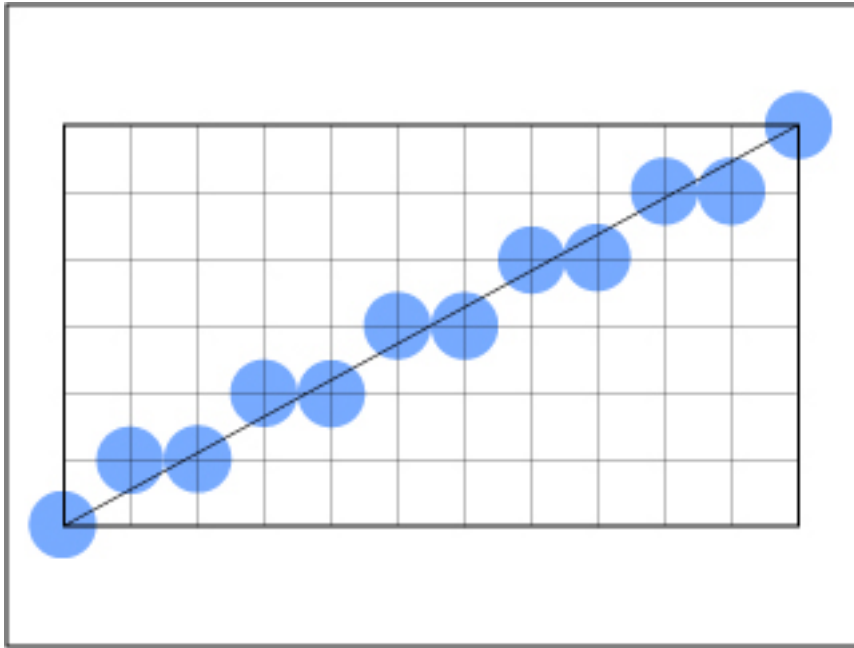
### 1.4.1. Staircase Effects

As described in the chapter **line rasterisation**, a line represented in raster mode has to be approximated by a sequence of pixels.

*"Consider using the midpoint algorithm to draw a 1-pixel-thick black line, with slope between 0 and 1, on a white background. In each column through which the line passes, the algorithm sets the colour of the pixel that is closest to the line. Each time the line moves between columns in which the pixels closest to the line are not in the same row, there is a sharp jag in the line drawn into the canvas. The same is true for other scan-converted primitives that can assign only one of two intensity values to pixels."* (FOLEY et al. 1996, p. 132)



If we now use a display device with a higher horizontal and vertical resolution, the jags are getting smaller but more numerous:



Although the resulting picture looks better, jags are still existing and in addition the memory requirement of the picture increases. Increasing resolution is an expensive solution that only diminishes the problem of jaggies, but it does not eliminate the problem itself.

We therefore will explain you in the next chapter a technique called anti-aliasing that results in better images without shooting up its memory requirements.

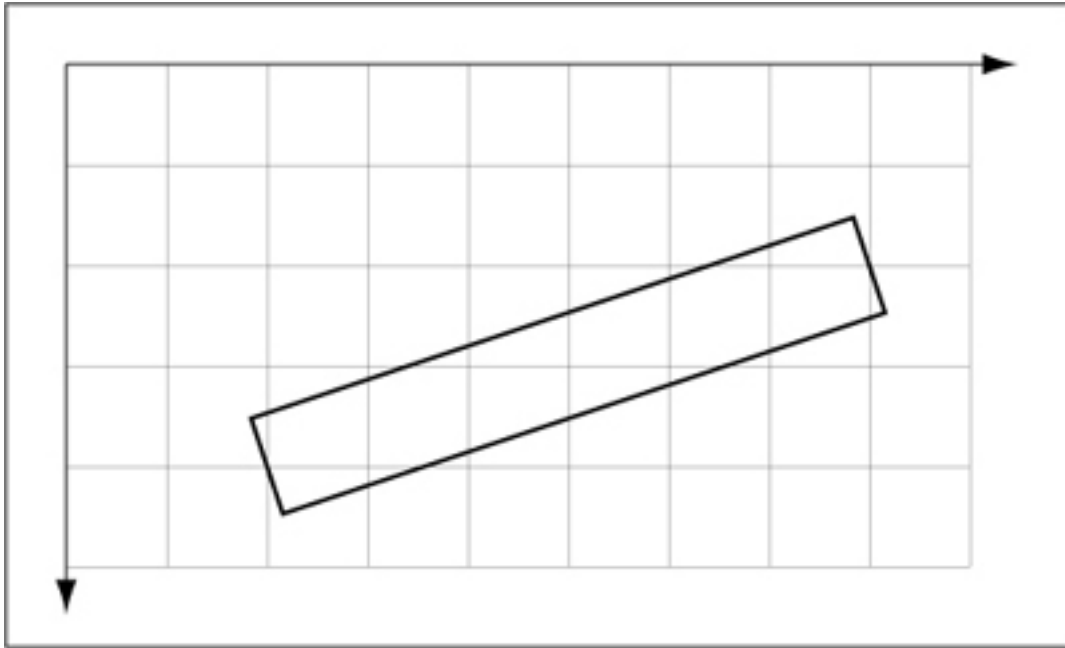
### 1.4.2. How is Anti-Aliasing realized?

When you look at the next picture you see the staircase effect as the result of the contrast between black and white pixels.

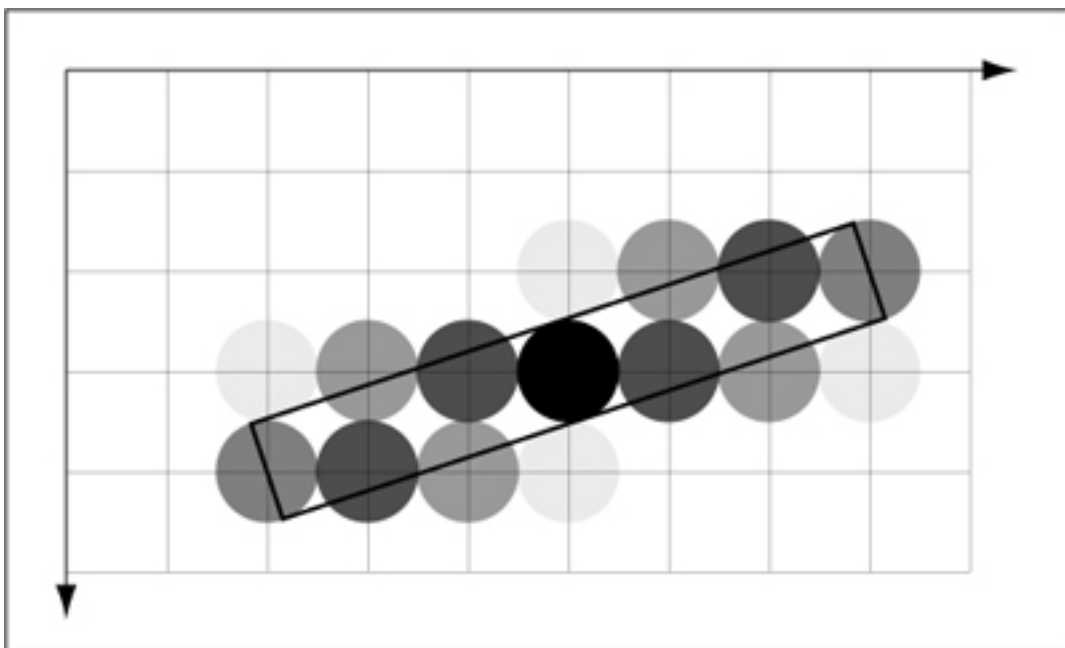


To soften this effect, we can use intermediate grey values for some pixels. But which pixels do we have to choose and what grey value do we have to assign the pixel?

For this purpose we think of any line as a rectangle of a desired thickness covering a portion of grid, as shown in the next image:



We now colour each pixel that intersects with the rectangle in a shade of grey. Its brightness is proportional to the area of the intersection. A fully covered pixel is still set to black. Pixels not intersected by the rectangle are completely white.



*"The number of pixels that are no longer white is greater than before, but if we take the grey pixels and multiply the area of each one by the value used to colour it, the total amount of greyness, as it were, is the same as that produced by only using a single black level to colour the original collection of pixels."* (CHAPMAN et al. 2000, p. 91)

This technique softens the harsh of a line. The blurring makes a line look better at a distance. See pictures:



*without Anti-Aliasing*



*with Anti-Aliasing*



*without Anti-Aliasing (zoomed)*



*with Anti-Aliasing (zoomed)*

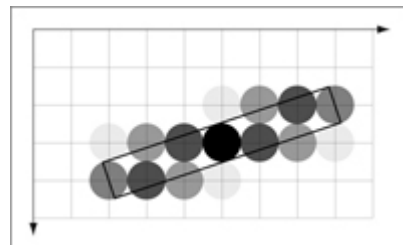
Anti-Aliasing can not only be applied to black and white pictures but also to coloured pictures. In this case we have to vary the transparency of the intersecting pixels instead of the grey values.

If you want to know more about anti-aliasing have a look at the pages 132-140 of (FOLEY et al. 1996).

### 1.4.3. Unit-Summary

On raster displays lines have to be approximated by intensifying the grid-point pixels lying on it or nearest to it. From this method emerge jags in the line which are called staircase effect.

To soften this effect we apply a procedure called anti-aliasing. Its principle is to use grey values for some pixels in black and white pictures or to use various transparency values for some pixels in coloured pictures.





# 1.5. Geometric Transformations

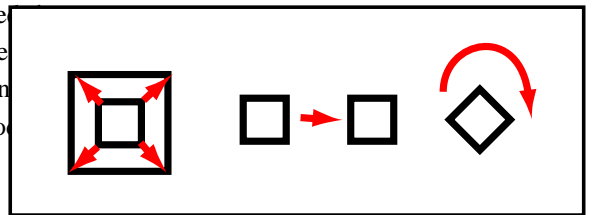
## Learning Objectives

- You will be able to list the three transformation types.
- You will be able to distinguish the transformation types by their formulas.
- You will be able to list the main characteristics of homogeneous coordinates.

## Introduction

You are certainly already familiar with the three geometric transformation types: translation, scaling and rotation. In this chapter we repeat their characteristics and their formula.

You will see that translation, addition and multiplication expressed in the usual matrix form cannot be combined that easily. Whereas the use of homogeneous coordinates allows to treat all three transformation types in a unified way, multiplication and alleviates their combination. That's why we introduce homogeneous coordinates and their principles in this chapter.



We will concentrate on transformations in the 2D-Space.

### 1.5.1. Geometric Transformations

We are able to change the geometry of each object by changing its size, position or orientation. These operations are called geometric transformations.

We translate, scale or rotate objects by first translating, scaling or rotating their *vertices*<sup>7</sup> and then drawing new lines between these translated vertices. Like this we get the new object.

#### Translation

We translate objects in the (x, y) plane to new positions by adding translation vectors to the coordinates of their vertices.

$$x' = x + d_x$$

$$y' = y + d_y$$

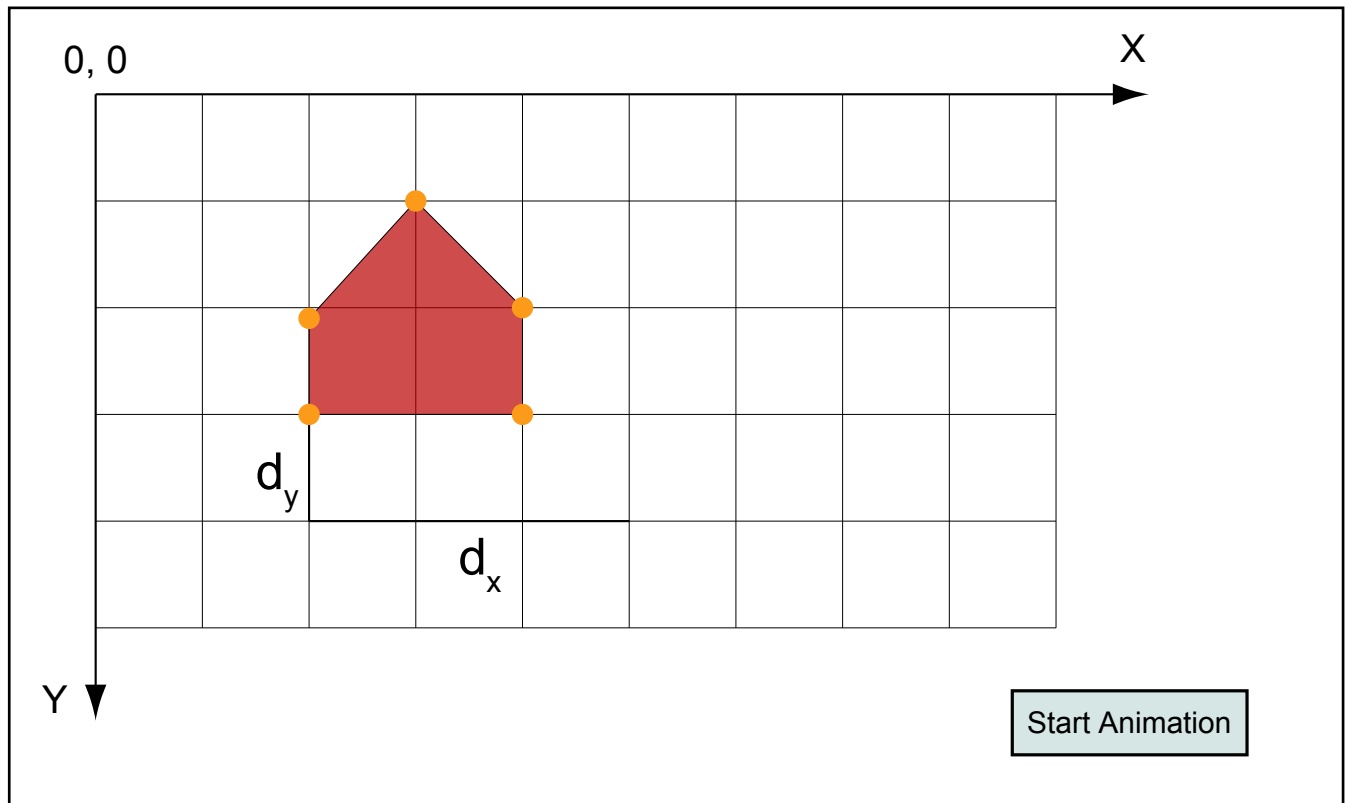
In matrix form this is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \quad \text{or} \quad P' = T + P$$

As explained above, after having translated the vertices, we draw new lines between these translated vertices for getting the translated object.

---

<sup>7</sup> A vertex is a corner of a polygon.



### Scaling

Objects can be stretched or shrunk by scaling their vertices. Their coordinates have to be multiplied by the factor  $s_x$  along the x axis and by the factor  $s_y$  along the y axis:

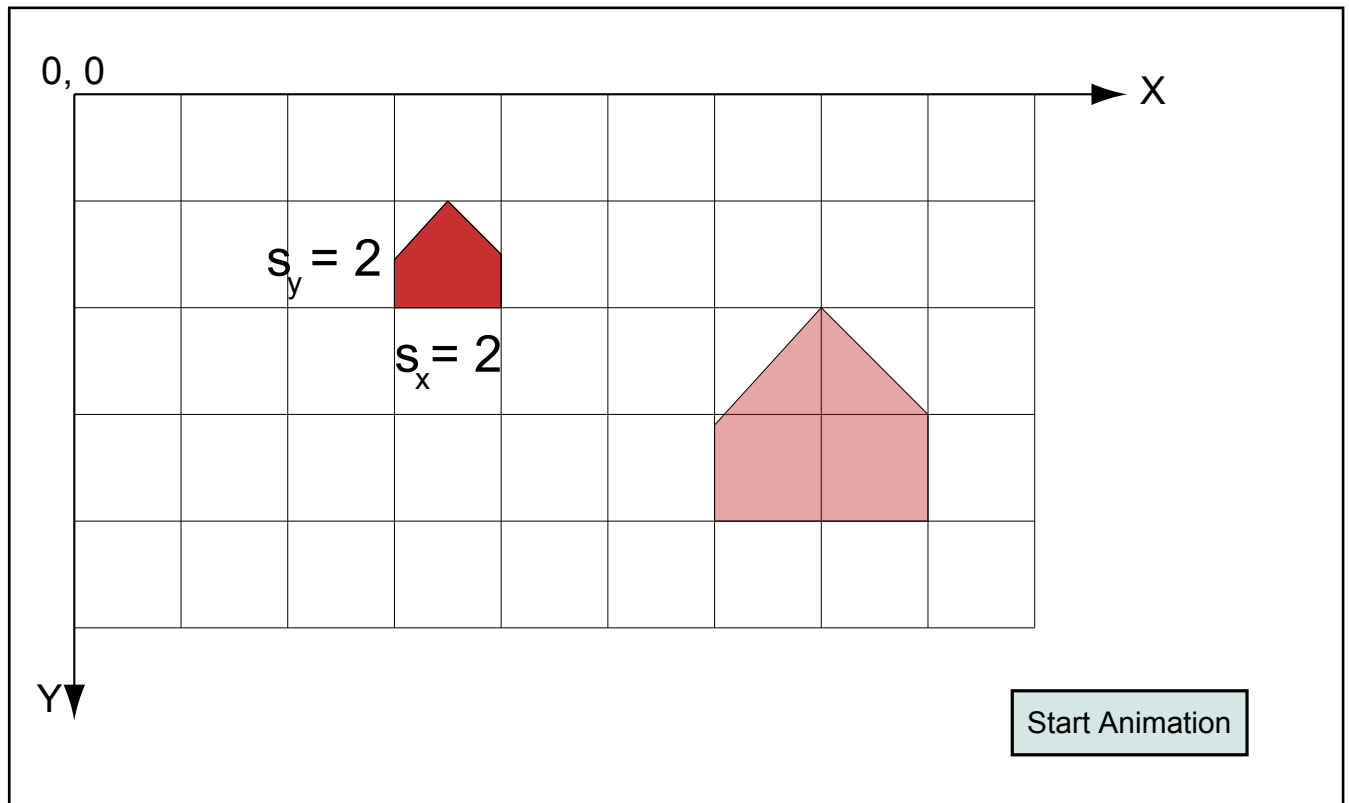
$$x' = x * s_x$$

$$y' = y * s_y$$

In matrix form this is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \text{ or } P' = S * P$$

The next animation part shows the scaling of an object. We don't show you again the two steps (first scaling vertices and then drawing the new object between these new vertices) because you know now that scaling happens in two steps.



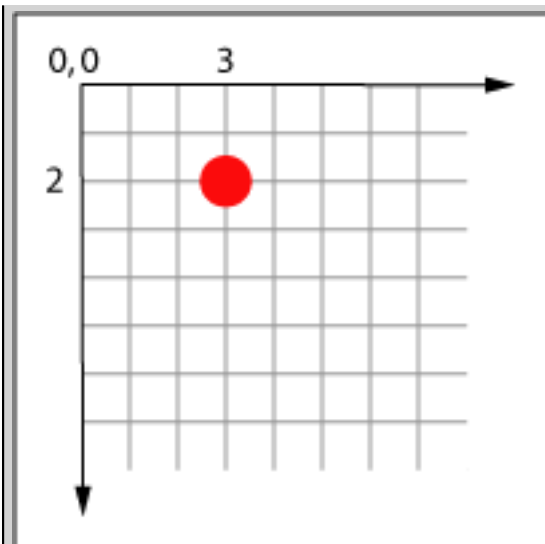
Notice that the reference point for scaling is the origin. Therefore, scaling is about the origin: The objects are getting smaller and closer to the origin if the scale factors are less than 1, if they are greater than 1, the objects are both larger and further from the origin. (FOLEY et al. 1996, p. 202)

**If you want to get more clarity why the object changes position by applying only scaling operations, have a look at the solutions which provide a calculation example of scaling.**

We want to scale an object with the coordinates  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$  and  $P_3(x_3, y_3)$  with factor 2 ( $s_x, s_y = 2$ ). We do the calculation with the coordinates of  $P_1$ .

Initial Position:

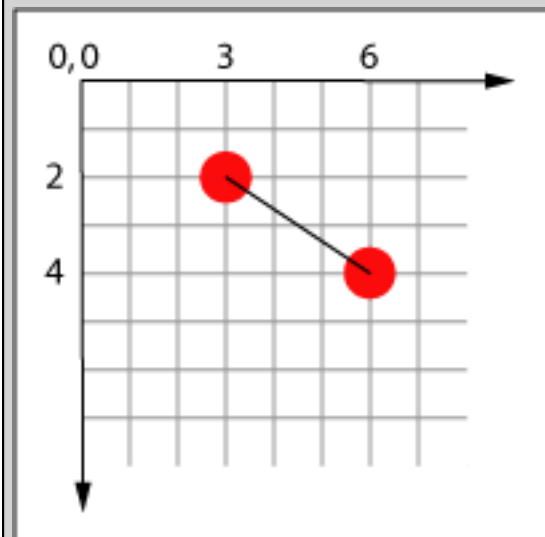
$x_1 = 3$ ;  
 $y_1 = 2$ ;



Calculation:

$$x' = 3 * 2 = 6$$

$$y' = 2 * 2 = 4$$



You now see, that the point is not only scaled but also translated.

If you do this calculation with all other points of the object, you get the scaled object.

### Rotation

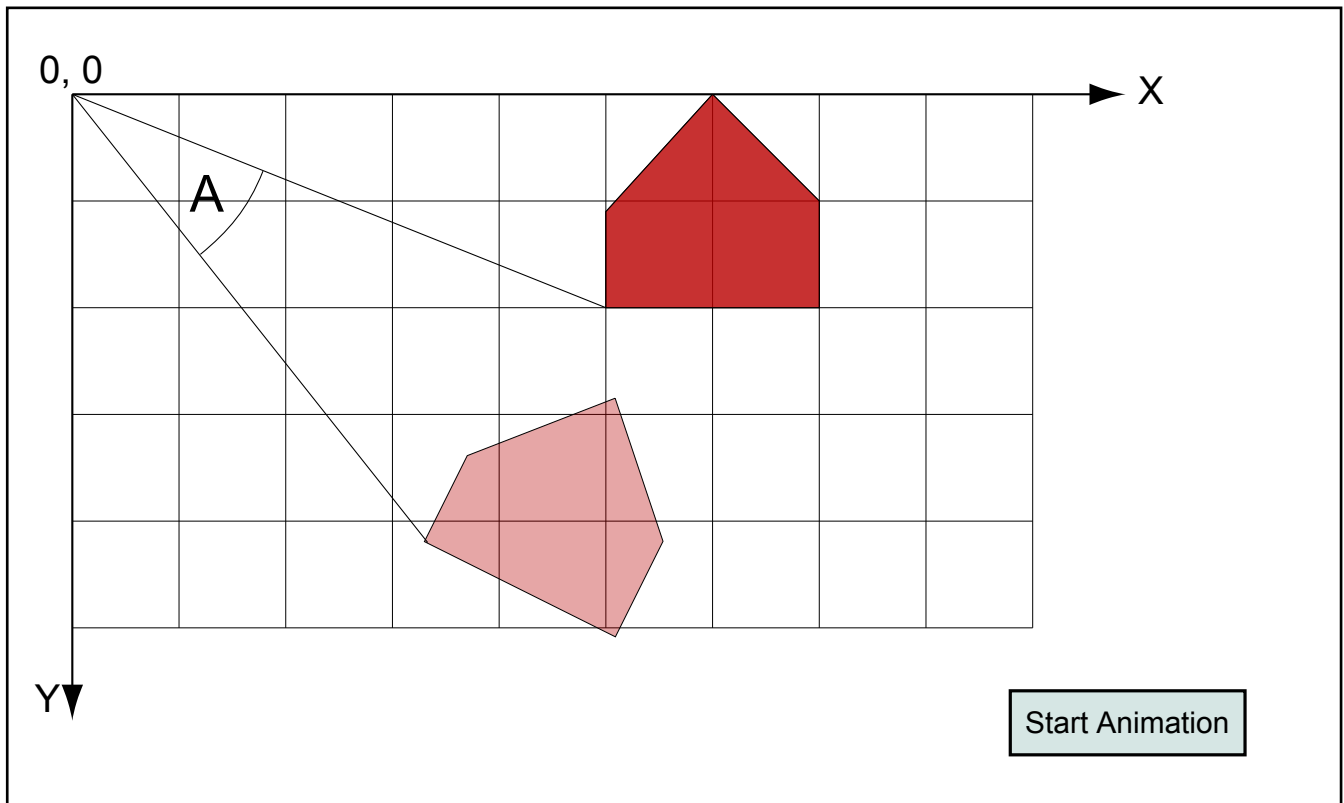
Objects can be rotated by an angle  $A$  around the origin. Positive angles are measured clockwise from  $x$  toward  $y$ . A rotation is defined mathematically by:

$$x' = x * \cos(A) - y * \sin(A)$$

$$y' = x * \sin(A) + y * \cos(A)$$

In matrix form this is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos A & -\sin A \\ \sin A & \cos A \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{or} \quad P' = R * P$$



As with scaling, rotation is about the origin, that's why objects change also position. (FOLEY et al. 1996, p. 203). To proof this statement make the calculation of an example as it is shown in the popup window above.

### 1.5.2. Homogeneous Coordinates

As explained above the matrix representation for translation, scaling and rotation are:

$$P' = T + P$$

$$P' = S * P$$

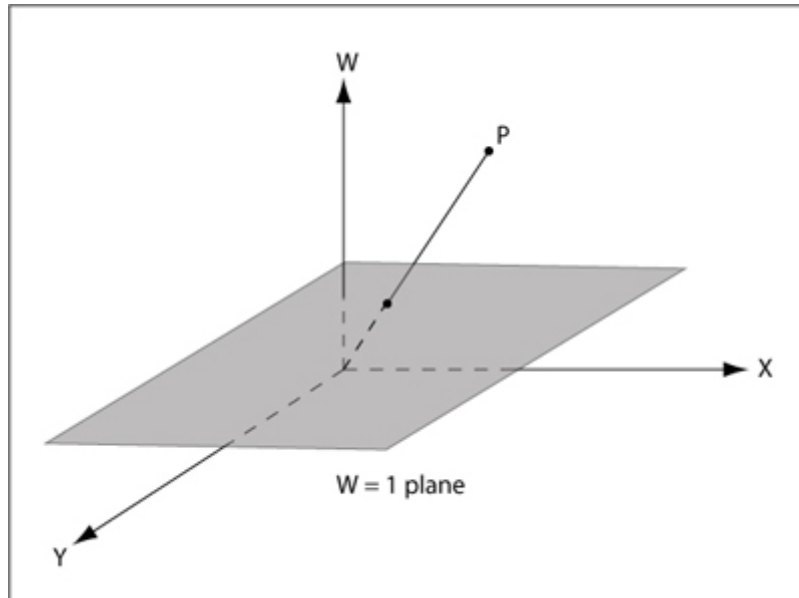
$$P' = R * P$$

Unfortunately, translation is treated as an addition whereas scaling and rotation as a multiplication. But we would like to be able to treat all three transformations in a consistent way, so that they can be combined easily.

The use of homogeneous coordinates allows to treat all three transformations as multiplications. But what are homogeneous coordinates?

In homogeneous coordinates, we add a third coordinate to a point. Instead of being represented by a pair of numbers  $(x, y)$ , each point is represented by a triple  $(x, y, W)$ . Two sets of homogeneous coordinates  $(x, y, W)$  and  $(x', y', W')$  represent the same point if one is a multiple of the other. Thus,  $(4, 5, 6)$  and  $(8, 10, 12)$  are the same points represented by different coordinate triples. *"That is, each point has many different homogeneous coordinate representations. Also one of the homogeneous coordinates must be nonzero:  $(0, 0, 0)$  is not allowed. If the  $W$  coordinate is nonzero, we can divide by it:  $(x, y, W)$  represents the same point as  $(x/W, y/W, 1)$ . When  $W$  is nonzero, we normally do this division, and the numbers  $x/W$  and  $y/W$  are called the cartesian coordinates of the homogeneous point. The points with  $W=0$  are called points of infinity."* (FOLEY et al. 1996, p. 204)

We are now using triples of coordinates, which normally represent points in 3D-space, to represent points in 2-D space. The intention is this: If we take all triples representing the same point (all triples are of the form  $(tx, ty, tW)$  with  $t$  unequal 0) we get a line in 3D-space. Therefore each homogeneous point represents a line in 3D-space. If we homogenise the point by dividing by  $W$ , we get a point of the form  $(x, y, 1)$ . Thus, the homogenized points form the plane defined by the equation  $W=1$  in  $(x, y, W)$ -space. (FOLEY et al. 1996, p. 204-205)



The XYW homogeneous coordinate space, with the  $W=1$  plane and point  $P(X,Y,W)$  projected onto the  $W=1$  plane. According to (FOLEY et al. 1996)

The translation can now be expressed as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation matrix by using homogeneous coordinates

The scale and rotation don't differ significantly in the notation showed above:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale matrix by using homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos A & -\sin A & 0 \\ \sin A & \cos A & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation matrix by using homogeneous coordinates

Now we are able to combine easily each operation by only calculating matrix products. Thus any sequence of transformations can be reduced to one single transformation matrix. The next chapter will show you an exemplary calculation.

### 1.5.3. Exemplary Computation

In this chapter we first combine a scaling operation with a translation operation. The new matrix is then combined with a rotation. Surely that these steps could be merged to one operation. But by looking at these steps separately it is easier to understand the whole operation.

Combination of Scaling and Translation:

| Scale Matrix  | Translation Matrix  | New Matrix  |
|---|---|---|
| $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$ | $= \begin{bmatrix} s_x & 0 & d_x \cdot s_x \\ 0 & s_y & d_y \cdot s_y \\ 0 & 0 & 1 \end{bmatrix}$ |

Combination of the new matrix and rotation:

| New Matrix  | Rotation Matrix  | Result   |
|---|--|--|
| $\begin{bmatrix} s_x & 0 & d_x \cdot s_x \\ 0 & s_y & d_y \cdot s_y \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} \cos A & -\sin A & 0 \\ \sin A & \cos A & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $= \begin{bmatrix} s_x \cdot \cos A & -s_x \cdot \sin A & d_x \cdot s_x \\ s_y \cdot \sin A & s_y \cdot \cos A & d_y \cdot s_y \\ 0 & 0 & 1 \end{bmatrix}$ |

Now we make the same steps with exemplary numbers. First we combine the scaling operation (parameters:  $s_x=3$  and  $s_y=3$ ) with a translation (parameters:  $d_x=2$  and  $d_y=4$ ):

| Scale Matrix  | Translation Matrix  | New Matrix   |
|---|---|--|
| $\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix}$ | $= \begin{bmatrix} 3 & 0 & 6 \\ 0 & 3 & 12 \\ 0 & 0 & 1 \end{bmatrix}$ |

And now we integrate the rotation considering an angle  $A = 90^\circ$ :

| New Matrix   | Rotation Matrix  | Result  |
|--|--|---|
| $\begin{bmatrix} 3 & 0 & 6 \\ 0 & 3 & 12 \\ 0 & 0 & 1 \end{bmatrix}$ | $\cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $= \begin{bmatrix} 0 & -3 & 6 \\ 3 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix}$ |

The transformation matrix including scaling, translation and rotation can now be applied to any arbitrary point:

| New Point                                     | Translation, Scaling and Rotation  | Point                                       |
|---|--|---|
| $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ | $= \begin{bmatrix} s_x \cdot \cos A & -s_x \cdot \sin A & d_x \cdot s_x \\ s_y \cdot \sin A & s_y \cdot \cos A & d_y \cdot s_y \\ 0 & 0 & 1 \end{bmatrix} \cdot$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ |

Using a point with the coordinates x=3 and y=2 the computation looks like this:

| New Point                                     | Translation, Rotation and Scaling  | Point                                       | Result  |
|---|--|---|---|
| $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ | $= \begin{bmatrix} 3 & 0 & 6 \\ 0 & 3 & 12 \\ 0 & 0 & 1 \end{bmatrix} \cdot$ | $\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$ | $= \begin{bmatrix} 15 \\ 18 \\ 1 \end{bmatrix}$ |

#### 1.5.4. Unit-Summary

There are three different types of transformations:

- Translation
- Scaling
- Rotation

To combine these transformations easily, we use homogeneous coordinates. In homogeneous coordinates we add a third coordinate (W) to a point (x, y). By dividing the x- and y-coordinate by W, we receive coordinates of the form (x/W, y/W, 1). By doing so, the translation can be expressed as a multiplication. Therefore we are able to combine the three transformations by only calculating matrix products.



# 1.6. Curves

## Learning Objectives

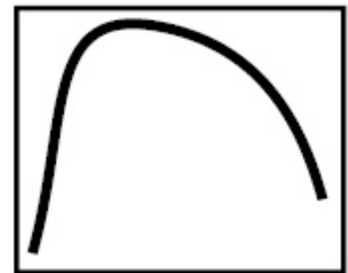
- You will be able to distinguish between cubic and quadratic Bézier curves.
- You will be able to draw a Bézier curve using the "De Casteljau Algorithm".
- You will be able to list at least three main characteristics of Bézier curves.

## Introduction

The majority of the objects of the real-world can not be modelled by approximated straight lines, because the objects have smooth contours. To model objects such as e.g. rivers in a map, we have to introduce smooth curves. A mathematical description of these objects is normally not available. Of course, one can use the coordinates of the infinitely many points of the object as a model, but this is not feasible for a computer with finite storage.

Normally the form of the objects is approximated with pieces of planes, spheres, or other shapes that are easy to describe mathematically. In computer graphics, particularly Bézier curves are used to model smooth objects.

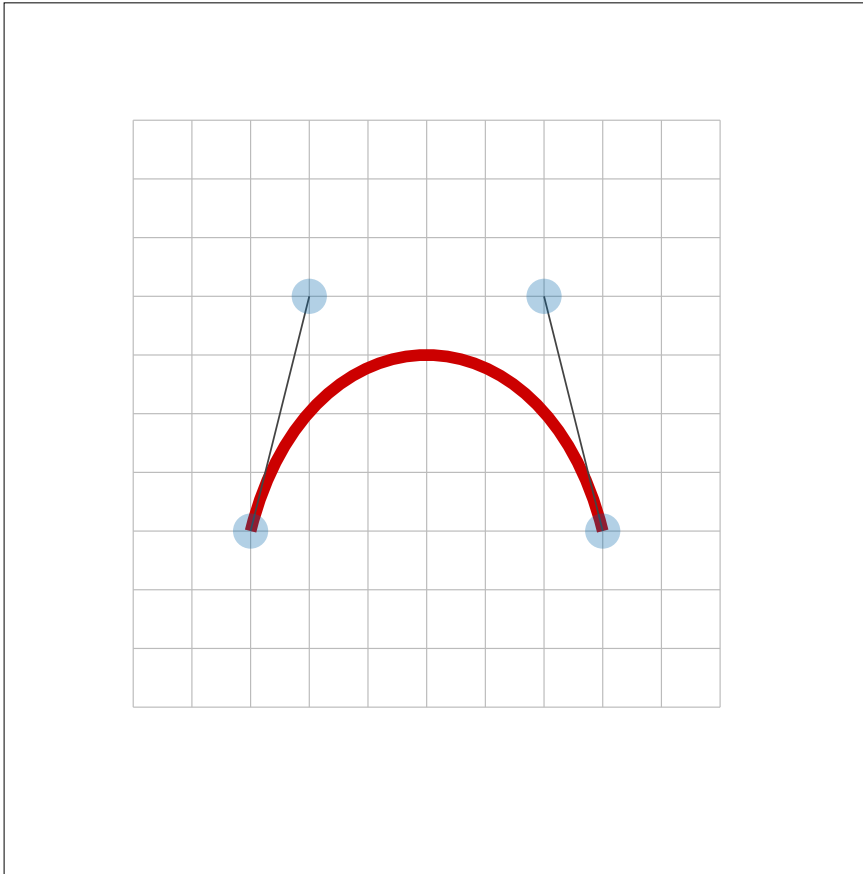
That's why the definition and characteristics of Bézier curves are explained in the next few paragraphs.



### 1.6.1. Bézier Curve

#### Experience the Bézier Curve

Have a look at the following interaction part. You can experience the characteristics of a cubic Bézier curve without having any knowledge of their theory. Later on we will explain their characteristics. You can change the form of the curve by moving the blue circles.



Note that the demonstrated cubic Bézier curve is a sub type of Bézier curves in general.

### Formula of Bézier Curves

Bézier curves are widely used in computer graphics to model smooth curves.

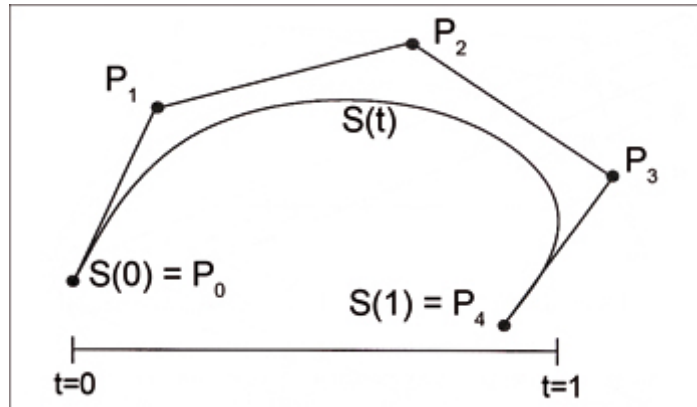
A n-dimensional Bézier curve is a curve of degree n. It is composed of Bernstein basis polynomials of degree n:

$$S(t) = \sum_{i=0}^n P_i b_{i,n}(t), \quad t \in [0, 1]$$

with the Bernstein basis polynomials of degree n defined as:

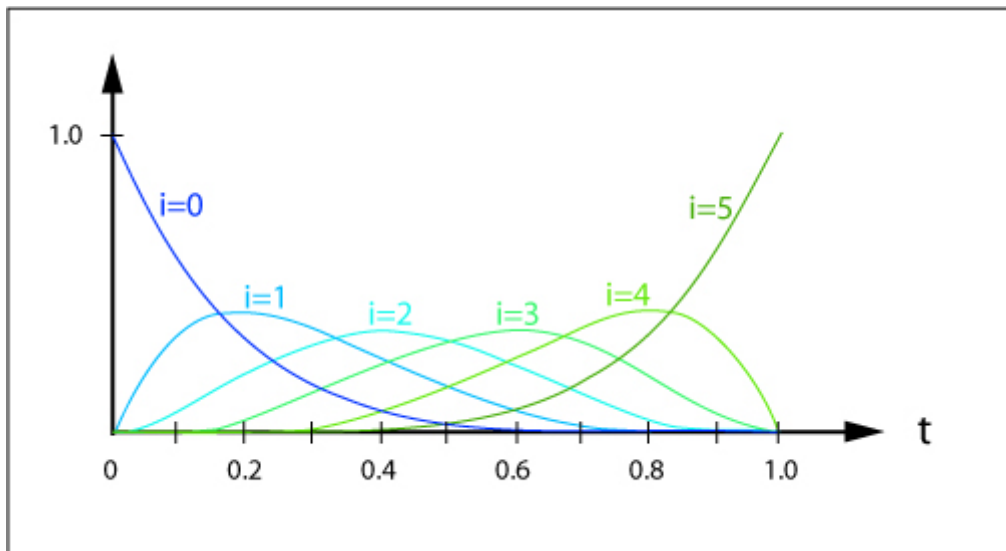
$$b_{i,n}(t) := \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \dots, n$$

$P_i$  is called *control point* for the Bézier curve. By connecting all control points with lines, we receive a polygon - starting in  $P_0$  and finishing in  $P_n$  - that is called Bézier polygon. The Bézier curve is completely contained in the hull built from the Bézier polygon.



*Bézier Curve of degree 4. (GIGER-HOFMANN 1992)*

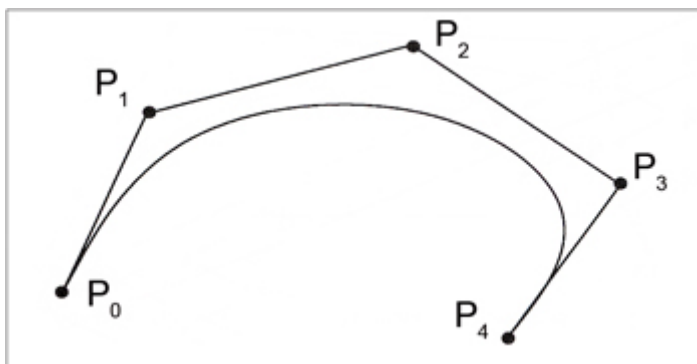
We go back to the issue of the Bernstein polynomials: The Bernstein polynomials are the weighting functions for Bézier curves. The following image shows the Bernstein polynomial functions of a Bézier curve of degree 5. You can see how much influence has each polynomial on the curve progression. For example at  $t=0$  only  $b_0$  is nonzero. Therefore all other polynomials don't have a bearing on the curve progression in the point  $S(t=0)$ .



### Characteristics of Bézier Curves

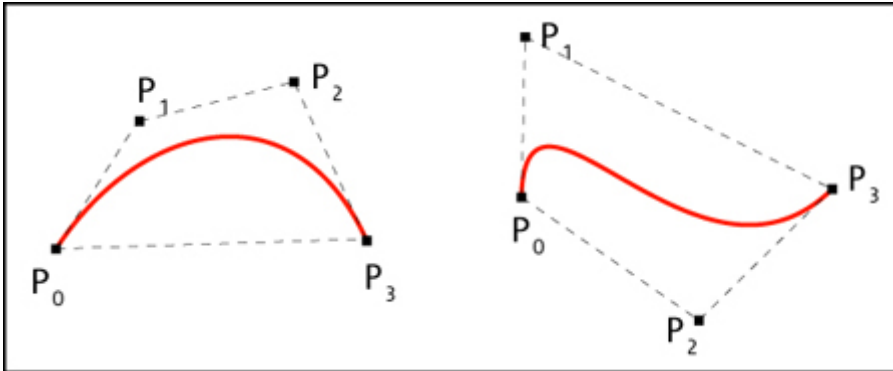
The following list contains the main characteristics of Bézier curves

- The starting point of the curve is  $P_0$  and the ending point is  $P_n$ .
- Normally, the other control points are not positioned on the curve.

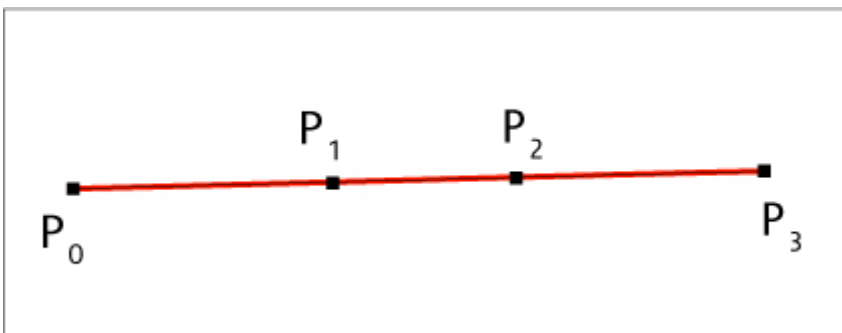


*Bézier Curve of degree 4. (GIGER-HOFMANN 1992)*

- The Bézier curve is completely contained in the convex hull built from the control points.

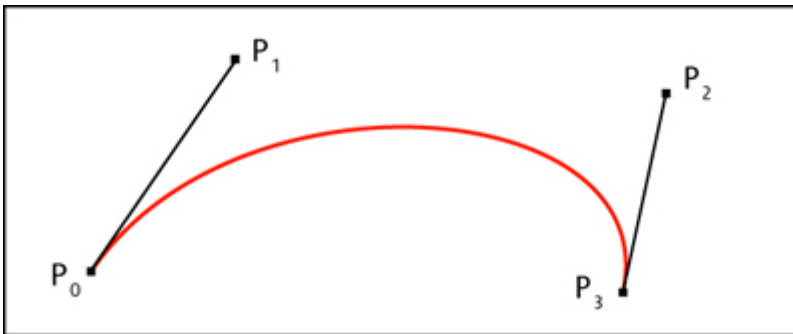


- If and only if all control points lie on the curve it is a straight line.



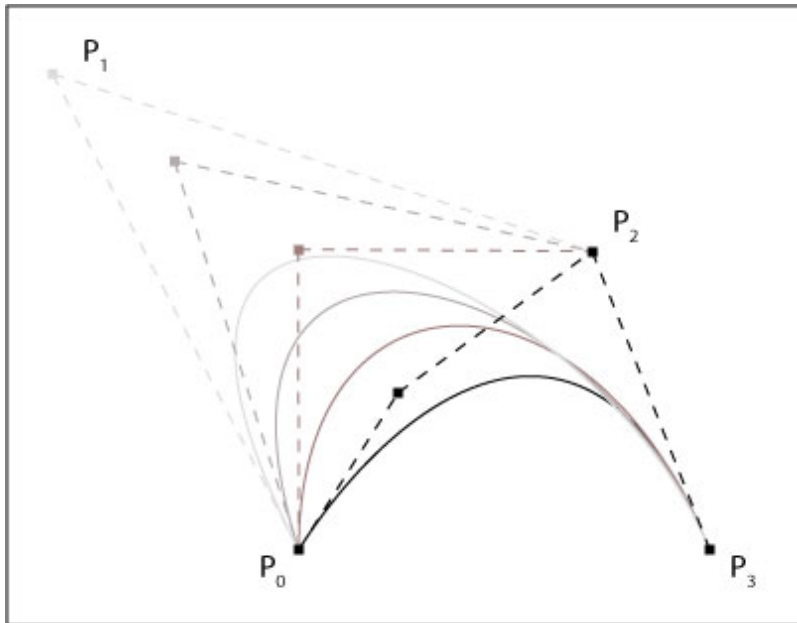
*All control points on the curve*

- The start (end) of the curve is tangent to the first (last) section of the Bézier polygon.



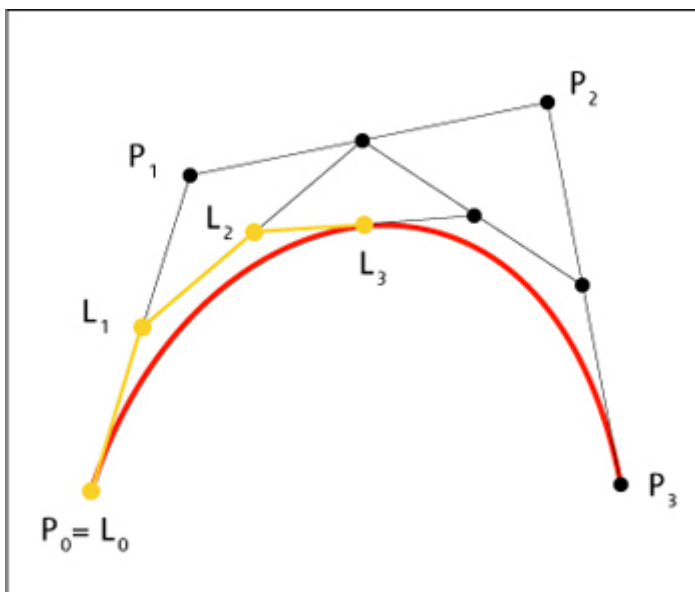
*Tangents of the curve*

- $P_i$  provide the direction of the curve (they pull the curve in their direction). The weighting of the points depends on the Bernstein polynomials and therefore on  $t$  as you could see above.



*Weighting of control points according to (WATT et al. 1998)*

- A curve can be split at any point into 2 subcurves, or into arbitrarily many subcurves, each of them is also a Bézier curve.

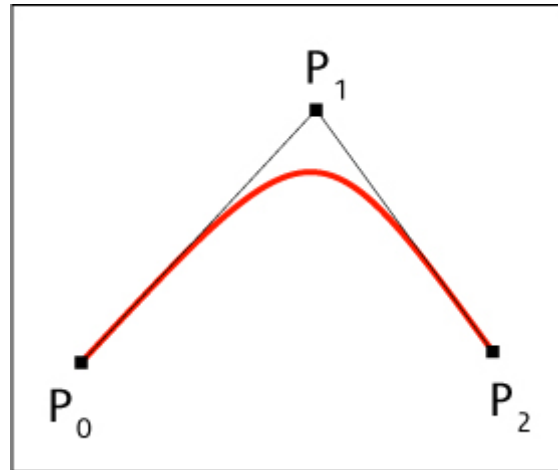


*Subdivision of Bézier curves*

### 1.6.2. Quadratic and Cubic Bézier Curves

#### Quadratic Bézier Curve

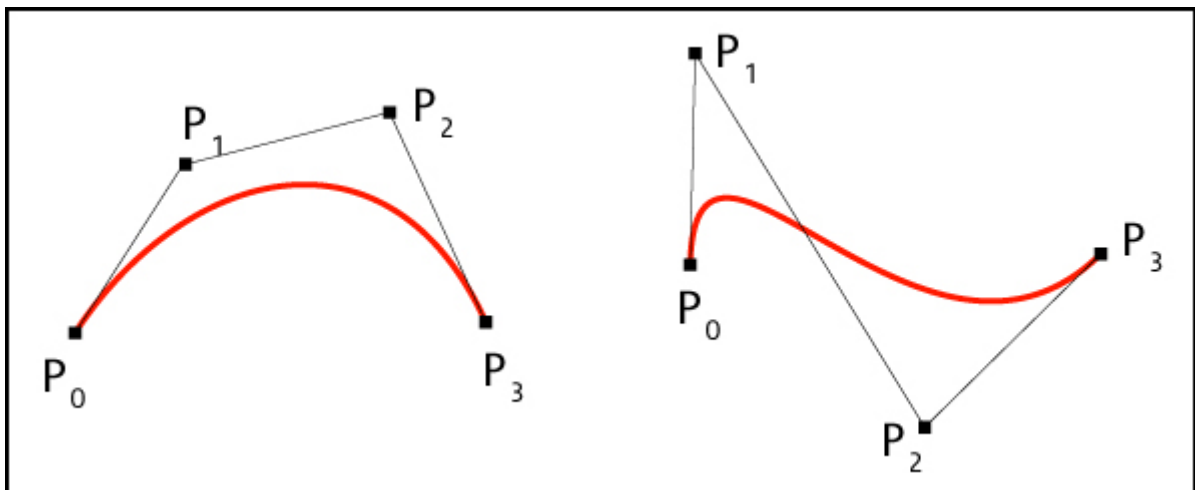
A quadratic Bézier curve is a Bézier curve of degree 2 and is defined through 3 points ( $P_0$ ,  $P_1$  and  $P_2$ )



*Quadratic Bézier curve*

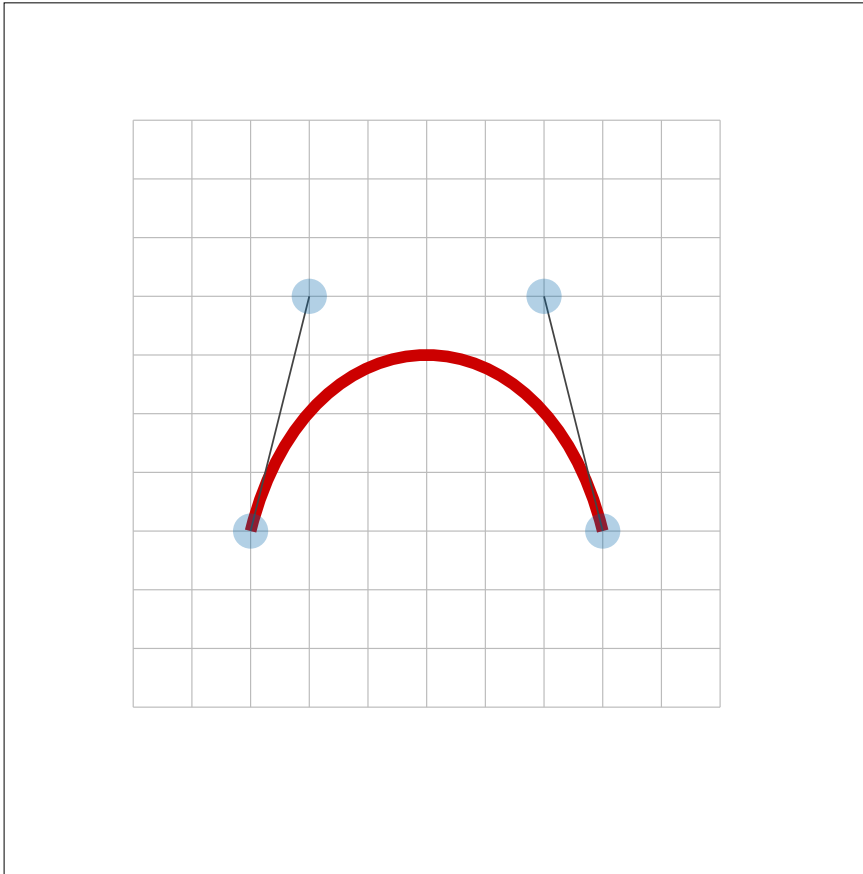
### Cubic Bézier Curve

A cubic Bézier curve is a Bézier curve of degree 3 and is defined by 4 points ( $P_0$ ,  $P_1$ ,  $P_2$  and  $P_3$ ). The curve starts at  $P_0$  and stops at  $P_3$ . The line  $P_0P_1$  is the tangent of the curve in point  $P_0$ . And so it is the line  $P_2P_3$  in point  $P_3$ . In general, the curve will not pass through  $P_1$  or  $P_2$ ; the only function of these points is providing directional information. The distance between  $P_0$  and  $P_1$  determines “how long” the curve moves into direction  $P_1$  before turning towards  $P_3$ .



*Cubic Bézier curve*

Now by considering the applied knowledge about Bézier curves, experience the Cubic Bézier Curve in the following interaction part a second time by moving the blue circles.



### 1.6.3. De Casteljau Algorithm

With the de Casteljau algorithm it is possible to construct a Bézier curve or to find a particular point on the Bézier curve. In this chapter we won't go into detail of the numeric calculation of the de Casteljau algorithm. We only want to illustrate figuratively the steps that are to be fulfilled to construct a Bézier curve or to find a point on the Bézier curve. If you are interested in the mathematical calculation have a look at the pages 121-125 of (HOSCHEK et al. 1992).

#### Construction of a Bézier Curve

To construct a Bézier curve we have to find several points through which the curve will go. These points depend on a parameter  $t$  "element"  $0,1$ . After we have found all points we are able to approximate the Bézier curve by connecting these points. It is obvious that the more points we have, the better is the approximation of the Bézier curve.

We can for example first look for the center of the curve and afterwards look for the quarter points of the curve and then connect these four points.

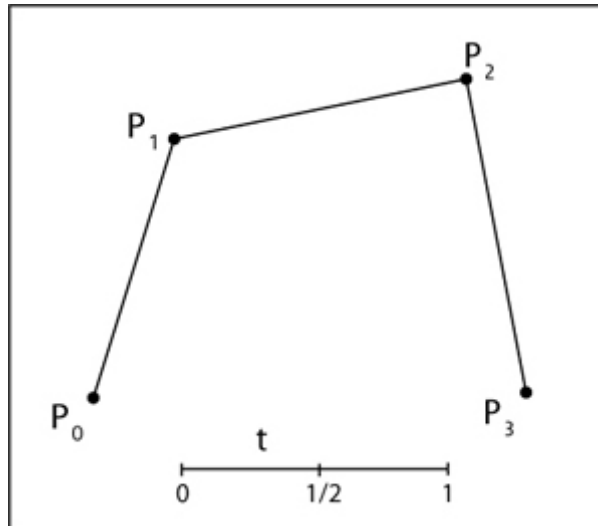
#### Exemplary Computation of One Point of the Curve

We are looking for the center ( $t=1/2$ ) of the curve.

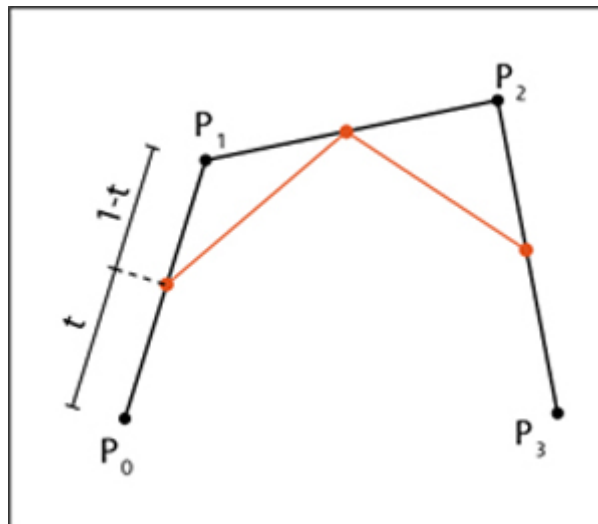
For each segment of the control polygon the points  $t=1/2$  are calculated. Afterwards the points of two consecutive segments are connected to each other. In the next step the points  $t=1/2$  of these new segments have to be determined. These steps have to be repeated as many times as the degree of the Bézier curve is; e.g. a cubic Bézier curve is of degree 3 therefore you have to execute the steps three times. By doing so, you get the center ( $t=1/2$ ) of the Bézier curve.

#### De Casteljau Algorithm in pictures

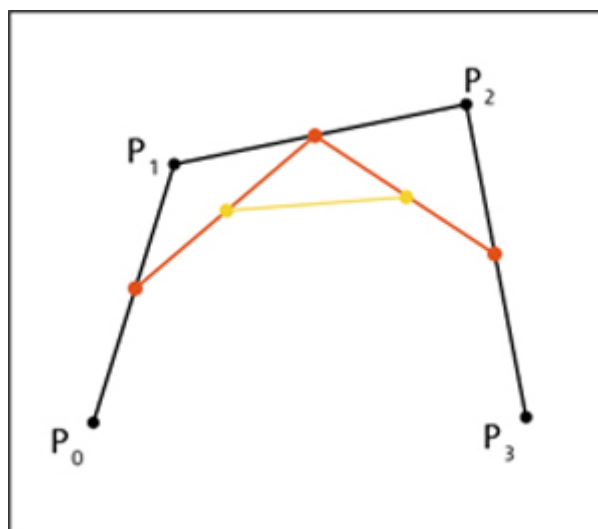
The following control polygon is given. The point with the parameter  $t=1/2$  ought to be calculated.



Now occurs the fragmentation of the polygon segments. The proportion of the fragmentation is defined through the parameter  $t$ . Each polygon segment is now divided in the ratio of  $t$  (as it is shown in the previous and the next image) . By doing so we reach the next polygon level:

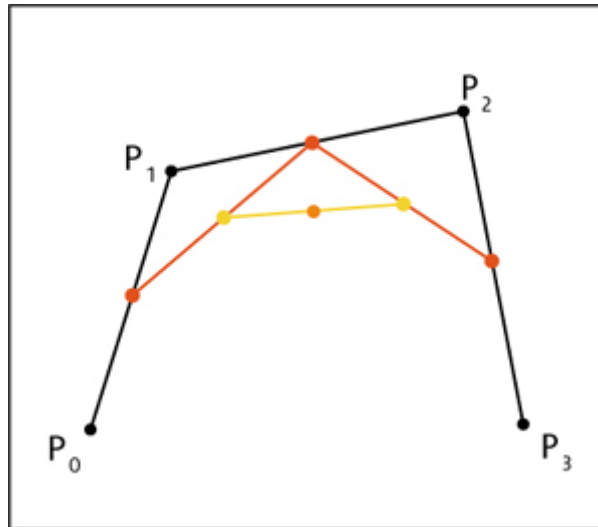


With the red polygon is dealt in the same manner as above. Each segment between the new points is divided in the ratio of  $t$ .

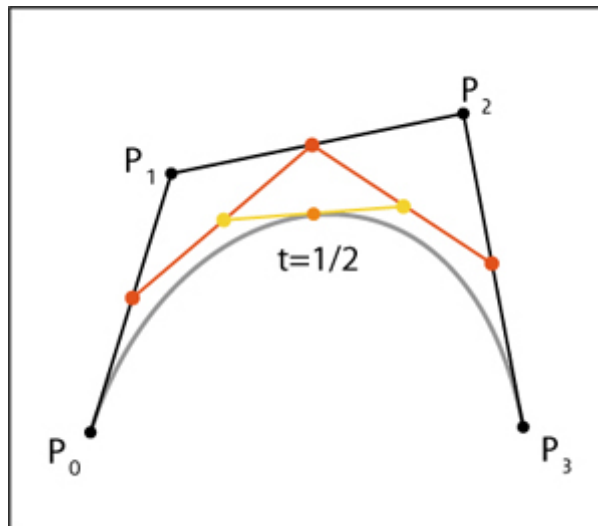




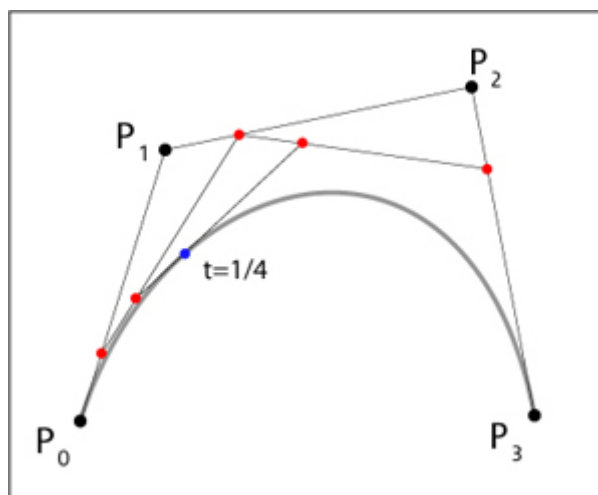
Also the last resulted segment is divided in the ratio of  $t$  and we get the final point marked in orange.



If this algorithm is proceeded for many values of  $t$ , we finally get the grey marked curve.



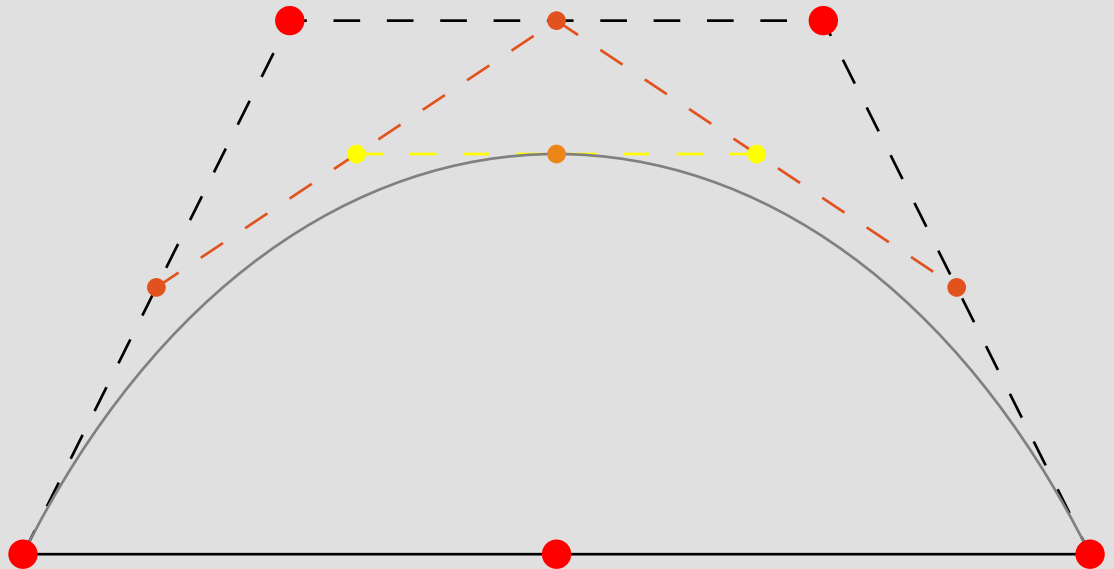
The solution for the point  $t=1/4$  looks as following:



Experience the deCasteljau algorithm in the following interaction part by moving the red dots.

Cubic Bezier : construction with De Casteljau Algorithm  
Coefficients for barycentres : 0.5 and 0.5

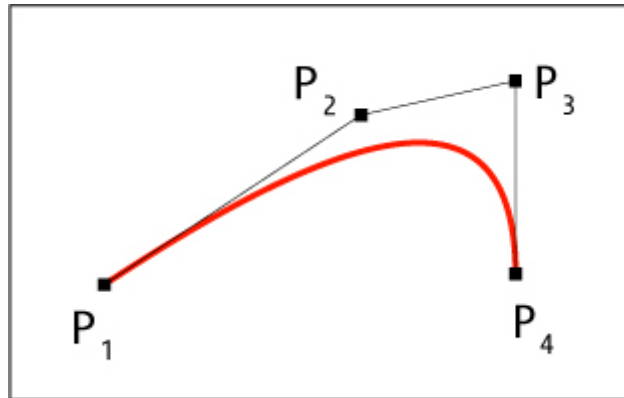
Bar



### Finding a Point on a Bézier Curve

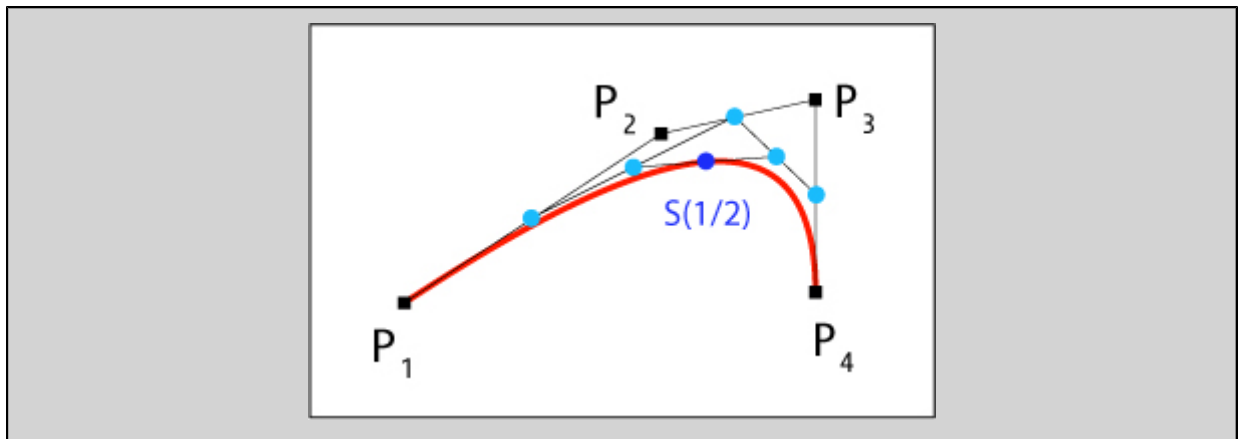
The same algorithm is used for determining points on a Bézier curve. In this case the curve already exists. A possible task may look like this:

Find the center ( $t=1/2$ ) of the mapped Bézier curve!



By applying the "De Casteljau algorithm", you will find the center of the curve.

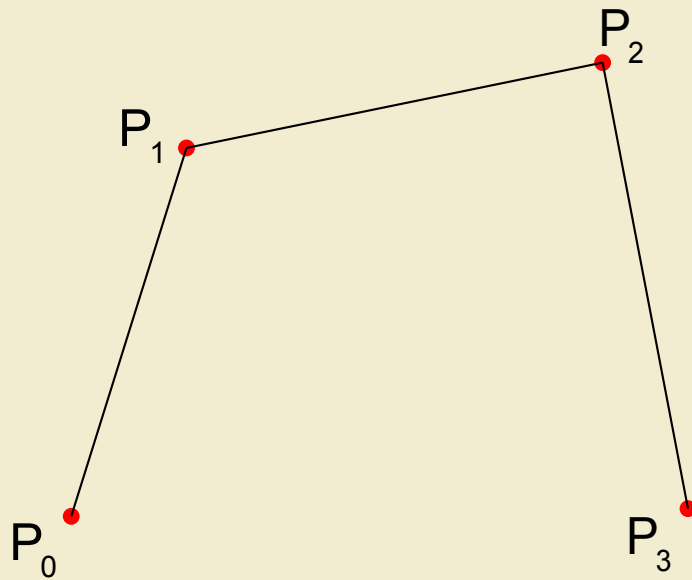
**Have a look to see the solution!**



### 1.6.4. Unit-Summary

Bézier curves are used to model objects with smooth contours such as a ball, a teacup etc. Bézier curves are composed of Bernstein basis polynomials of degree  $n$  and points  $P_i$  called control points. By connecting all control points with lines, we receive the control polygon. The Bézier curve is completely contained in the hull built from the control polygon. A Bézier curve of degree 2 is called quadratic Bézier curve and a curve of degree 3 is called cubic Bézier curve. The "De Casteljau Algorithm" is used to construct a Bézier curve or to find a particular point on the curve. In the following interaction part you find again the different steps of the construction of a Bézier curve using the De Casteljau algorithm.

# Construction of Bézier Curve



Step 1

Step 2

Step 3

Step 4

Be aware that not all applications are able to visualise Bézier curves e.g. some GIS applications. In this case the Bézier curves have to be converted into paths. The higher the number of vertices of the new path the better it equals a Bézier curves. Take into account that such a path needs much more storage space than an original Bézier curves, because of the many vertices.

### 1.7. Paths

#### Learning Objectives

- You will be able to model simple objects such as for example rectangles.
- You will be able to list at least five possible presentation attributes of a path.

So far you know how to draw a straight line on a raster screen and the characteristics of Bézier curves. But since you can not model an object with only one line or curve, we have to introduce an element that combines several lines or curves or both. This element is called path.

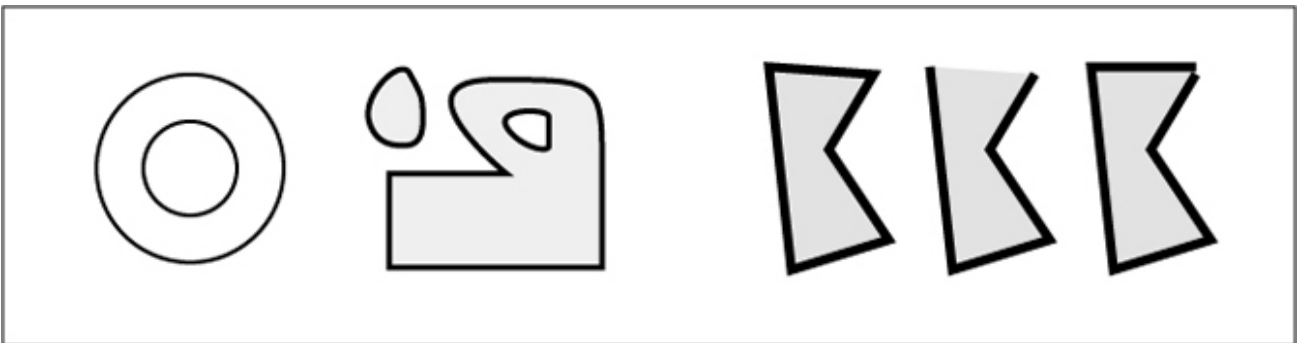


In this chapter we give you a definition of the element "path" and introduce you to its characteristics.

#### 1.7.1. Definition

A path is a sequence of  $n$  points  $P_i$ . Thereby the connection type (line, curve) for every two points has to be specified. Paths are used to describe every arbitrary shape. Paths are particularly used to describe complex geometries such as open, closed, or multi-part shapes.

The following picture shows you a few shapes defined by a path:



*Different shapes defined by paths*

#### 1.7.2. Creating any Path

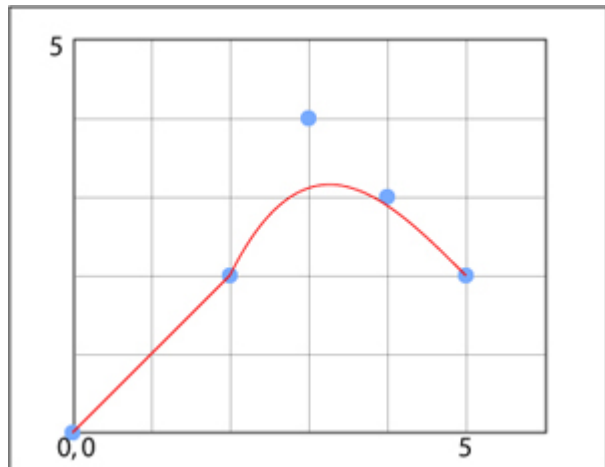
The following statements are needed to create any path element (they vary depending on the used program language):

- |  |   |
|--|---|
| • <code>move_to (x, y)</code>                    | sets the starting point of a shape or a part of a multi-part shape.                 |
| • <code>line_to (x, y)</code>                    | draws a line from the current point to the new defined point.                       |
| • <code>curve_to (x0, y0, x1, y1, x2, y2)</code> | draws a cubic Bézier curve from the current point, using the listed control points. |
| • <code>close_path ()</code>                     | closes the path by drawing a line to the first point of the path.                   |

#### Example of C-Code

```
path.move_to(0,0);  
path.line_to(2,2);  
path.curve_to(3,4,4,3,5,2);
```

*Example of code*





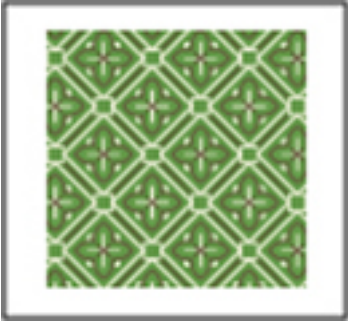


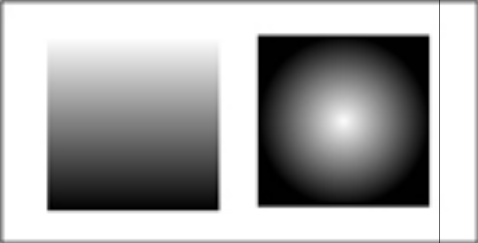

*How it looks on the screen*

The direction of a path often plays an important role e.g. when transforming a path into an arrow some software (e.g. Adobe Illustrator) ask you if the arrowhead has the be added at the end or at the beginning of the path. Therefore be aware of this aspect when working with paths.

### 1.7.3. Presentation of a Path

The presentation of a path is defined through graphical attributes such as:

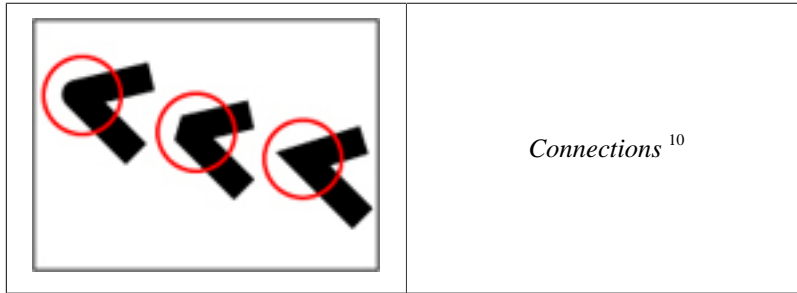
| Attribute   | Example      |
|---|--------------|
|  | Colour       |
|  | Transparency |

|   |   |
|---|---|
|    | Pattern   |
|    | Stroke-Width  |
|   | Dashed Line   |
|  | <i>Colour Gradient</i> <sup>8</sup>                         |
|  | <i>Line Cap Types</i> <sup>9</sup><br>(butt, round, square) |

---

<sup>8</sup> Colour gradient is a smooth blending of shades from light to dark or from one colour to another.

<sup>9</sup> The Line Cap type defines the presentation of the line cap. It may be butt , round or square .



### 1.7.4. Unit-Summary

A path consists of an arbitrary number of vertices. The connection type - line or curve - between these vertices has to be specified. Paths are used to model complex objects such as e.g. rivers in a map.

The presentation of a path is defined by various attributes such as colour, stroke-width, transparency, etc.



---

<sup>10</sup> Connection defines the presentation of the joining of two path segments. It may be square or sharp , round or flattened .



## 1.8. Self Assessment

### 1.9. Summary

Today computers become more and more important since we operate with them almost every day. But scarcely anybody knows how an object is visualised on a computer screen. We use drawing or visualisation programs without asking ourselves how it is possible to draw e.g. a line on the screen. That's why we taught you the main principles of computer graphics in this lesson. Hopefully, you are now more familiar with the workflow of a computer when you operate with drawings and pictures.

We introduced also the main characteristics of raster displays. Among them are the coordinate system of the screen, how pictures and objects are visualised and drawn on the screen etc.

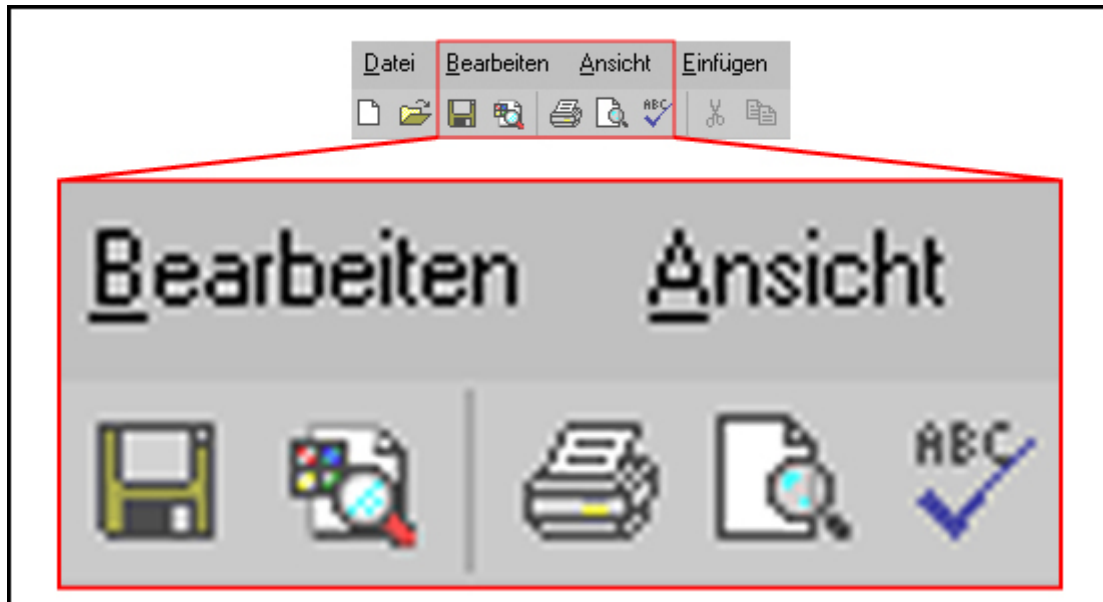
You learned which colour model is used to represent colours on the screen and how colours are coded.

In all drawing programs an object can be translated, scaled or rotated with only one click. Now you know which steps a computer has to calculate to do this transformations.

Some objects on the screen cannot be represented by simple lines. Therefore, Bézier curves were introduced. In drawing programs you will be confronted with cubic Bézier curves and their characteristics when drawing an arbitrary curve.

To soften the staircase effect of objects visualised on screen, anti-aliasing is applied. This technique smoothes the contours of objects. In almost all drawing programs you will find the option to apply anti-aliasing to drawings.

The next image demonstrates you that everything on computer screens consists of pixels. Even the letters and icons in the popular Microsoft Office software.



### 1.10. Recommended Reading

- **FOLEY, J.D., VANDAM, A., FEINER, S.K., Hughes, J.F.**, 1996. *Computer Graphics - Principles and Practice*. Second Edition in C. Addison-Wesley Publishing Company.  
Chapters 3.1 - 3.7 and 5.1 - 5.3

### 1.11. Glossary

**Algorithm:**

An algorithm is a mathematical rule or procedure for solving a problem.




**Cartesian Coordinate System:**

A cartesian coordinate system is a two dimensional system in which x measures horizontal distance and y measures vertical distance. The pair (x, y) defines every point on the plain.

**Colour Gradient:**

Colour gradient is a smooth blending of shades from light to dark or from one colour to another.

**Connection:**

Connection defines the presentation of the joining of two path segments. It may be square or sharp , round  or flattened .

**Floating-Point Number:**

A floating-point number is a digital representation for a number in a certain subset of the rational numbers. The number of decimal places can be defined individually. Examples 1.235, 1.23, 1.2, etc.




**For Loop:**

A for loop is a control structure which allows code to be executed iteratively. For loops are typically used when the number of iterations is known before entering the loop. (Wikipedia - The Free Encyclopedia)

**hexadecimal:**

A counting system that uses 16 digits, notated as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

**Line Cap:**

The Line Cap type defines the presentation of the line cap. It may be butt , round  or square .

**Polygon:**

A polygon is a closed planar path composed of a finite number of sequential line segments.

**Vertex:**

A vertex is a corner of a polygon.

### 1.12. Bibliography

- **Adobe Illustrator CS.** *Illustrator-Hilfe*. [CD-ROM].
- **AGFA**, 1997. *Die Geheimnisse des Farbmanagements; Die digitale Farbe - Teil 5*. Agfa Gevaert AG.
- **BAER, H.R.**, 2005. *Geometrie und Computergraphik - Vorlesungsunterlagen*. ETH Zürich: Institut für Kartographie.
- **CHAPMAN, N.,CHAPMAN, J.**, 2000. *Digital Cartography*. New York: John Wiley & Sons.
- **Computer vom Discounter**. Available from: <http://computervomdiscounter.de/wp-content/images/acer1715.jpg> [Accessed 02.12.2005].
- **EISENBERG, J.D.**, 2002. *SVG Essentials*. 1. O'Reilly.
- **FOLEY, J.D., VAN DAM, A., FEINER, S.K., Hughes, J.F.**, 1996. *Computer Graphics - Principles and Practice*. Second Edition in C. Addison-Wesley Publishing Company.
- **GIGER, C.**, 2005. *Raumbezogene Informationssysteme III - Vorlesungsunterlagen*. ETH Zürich: Institut für Geodäsie und Photogrammetrie - Gruppe Geoinformatik .
- **GIGER-HOFMANN, C.**, 1992. *Ein Ray-Tracing-Verfahren zur Visualisierung polynomialer Tensorproduktflächen*. published. Fachbereich Informatik der Technischen Hochschule Darmstadt.
- **HOMANN, J.P.**, 1998. *Digitales Colormanagement*. Berlin, Heidelberg: Springer-Verlag.
- **HOSCHEK, J., LASSER, D.**, 1992. *Grundlagen der geometrischen Datenverarbeitung*. 2. Auflage. Stuttgart: B.G. Teubner Stuttgart.
- **Pila Information Educative**. Available from: <http://pilat.free.fr/> [Accessed 27.9.2005].  
Download: [http://pilat.free.fr/english/svg/bezier\\_cub\\_bary.htm](http://pilat.free.fr/english/svg/bezier_cub_bary.htm)
- **Universität Rostock.** *Geoinformatik-Service* [online]. Available from: <http://www.geoinformatik.uni-rostock.de/lexikon.asp> [Accessed 02.08.2005-27.09.2005].
- **WATT, A., POLICARPO, F.**, 1998. *The Computer Image*. Addison Wesley Longman Limited.
- **WATT, A.H.**, 2002. *Designing SVG Web Graphics*. New Riders Publishing.
- **Wikipedia.** *Computer Graphics* [online]. Available from: [http://en.wikipedia.org/wiki/Computer\\_graphics](http://en.wikipedia.org/wiki/Computer_graphics) [Accessed 11.07.2008].
- **Wikipedia.** *RGB Colour Cube* [online]. Available from: [http://de.wikipedia.org/w/index.php?title=Bild:RGB\\_farbwuerfel.jpg](http://de.wikipedia.org/w/index.php?title=Bild:RGB_farbwuerfel.jpg) [Accessed 11.07.2008].
- **Wikipedia - The Free Encyclopedia.** *For Loop* [online]. Available from: [http://en.wikipedia.org/wiki/For\\_loop](http://en.wikipedia.org/wiki/For_loop) [Accessed 01.11.2005].