*FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING UNIVERSITY OF ZAGREB*

*Department of electronics, microelectronics, computer and intelligent systems*

# INTERACTIVE COMPUTER GRAPHICS

Laboratory assignments

*Željka Mihajlović, Marko Đurasević*

*Zagreb, 2018.*

# INTRODUCTION

The laboratory assignments are an integral part of the interactive computer graphics course. By practical work through these assignments the knowledge of computer graphics is adopted, established and extended. Furthermore, the assignments give the notion of the difficulty of certain procedures.

The basic ideas of the procedures need to be understood and practically adopted so that the underlying principle can be used to solve problems in another environment. Almost every process involves special cases that need to be taken into account, and this part is largely left as an additional research challenge.

Selected parts of the material are included in laboratory exercises. Primarily, this is about:

- graphic primitives,
- geometric calculations,
- transformations and projections,
-calculation of illumination,
- shading procedures,
- hidden lines and surfaces,
- spatial curve interpolation methods,
- fractal sets.

For each assignment it is required to prepare a working solution. Along with each assignment, the teaching material relevant for that assignment is briefly repeated. In addition, a possible solution for the assignment is given in the form of a short procedure, and the expected results of the program are indicated.

Creating the program requires modest hardware and software equipment in the lab. A device is required which gives the work program the ability to plot points, lines, change the colour on screen, and draw a triangle with different colour intensity in the vertices.

The first exercise provides basic mathematical operations that are used in computer graphics. These are, for example, dot product of vectors, cross product of vectors, and multiplication of matrices. The second exercise is concerned with drawing a line by using the Bresenham algorithm. This exercise provides the basic conversion process from a continuous form to the discrete form required when displaying on the screen.

In the third exercise, a convex polygon filling algorithm is used and some basic geometric calculations are performed when testing the relation of point and polygon. Filling the polygon is based on calculating the intersection of lines. In the fourth exercise it is necessary to build a topological body structure and examine the relationship between a point and the body.

The fifth exercise describes the process of view transformations and perspective projection, which are used to project the constructed body. Creating a spatial curve by the Bezier procedure is described in the sixth exercise, which is used to construct the animation when displaying the body.

In order for the body created in the previous exercises to be more faithfully displayed, it is necessary to shade it. The seventh exercise gives a comparison of constant and Gouraud's shading procedures, which are based on calculating the intensities, and the procedure for removing the rear polygons. The eighth exercise provides a simple procedure for creating and displaying fractal clusters.

# 1. Basic math operations in computer graphics

Linear algebra is a part of math that deals with vectors and matrices. In a computer graphics we want to show objects whose surface is approximated by dots and polygons. A convenient way to record such data is through vectors, while matrices are convenient for recoding transformations of objects such as translation, rotation, and similar. Therefore, we will need basic matrix operations that need to be performed in this exercise. It is desirable to have a library of basic mathematical operations that will be used in further exercises.

The choice of the programming language and program implementation which needs to be done in the exercises, i.e. the way in which the data structures and the implementation itself are defined, are entirely left to the student. The use of available mathematical libraries such as Glm OpenGL Mathematics (http://glm.g-truc.net/) or any other similar library is permitted, in which case it is necessary to demonstrate their use.

For understanding more complex procedures it is necessary to understand the elementary concepts as well as the geometric interpretation of basic operations.

## 1.1. Vector operations

For two given vectors, for example vectors $\mathrm{v}_1 = (x_1\, y_1\, z_1)$ i $\mathrm{v}_2 = (x_2\, y_2\, z_2)$ the addition and subtraction operators are defined.

$$\mathrm{v}_1 + \mathrm{v}_2 = (x_1 + x_2 \quad y_1 + y_2 \quad z_1 + z_2), \tag{1.1}$$

$$\mathrm{v}_1 - \mathrm{v}_2 = (x_1 - x_2 \quad y_1 - y_2 \quad z_1 - z_2).$$

The dot (scalar) product of vectors $V_1 = (x_1\, y_1\, z_1)$ and $V_2 = (x_2\, y_2\, z_2)$ is calculated as the sum of products of corresponding vector components, and the result is a scalar value :

$$\mathrm{v}_1 \cdot \mathrm{v}_2 = x_1 x_2 + y_1 y_2 + z_1 z_2. \tag{1.2}$$

The dot product is used for calculating the angle $\theta$ between two vectors by the following expression:

$$\cos(\theta) = \frac{\mathrm{v}_1 \cdot \mathrm{v}_2}{|\mathrm{v}_1|\,|\mathrm{v}_2|} = \frac{x_1 x_2 + y_1 y_2 + z_1 z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\,\sqrt{x_2^2 + y_2^2 + z_2^2}}. \tag{1.3}$$

where in the denominator the lengths of the vectors are calculated.

Additionally, it is useful to determine the projection of one vector on another. The projection of vector $\mathrm{v}_1 = (x_1\, y_1\, z_1)$ on vector $\mathrm{v}_2 = (x_2\, y_2\, z_2)$ is $\mathrm{v}_{1\ \mathrm{na}\ 2.}$

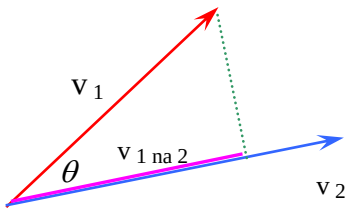



Figure 1.1. Projection fo vector $\mathrm{v}_1$ on vector $\mathrm{v}_2$.

When projecting one vector on another we can observe the scalar value, or the length of the

projected vector $v_1$ on vector $v_2$

$$v_{1na2} = \cos(\theta)|v_1| = \frac{v_1 \cdot v_2}{|v_2|} \tag{1.4}$$

which we obtain from expression (1.3) and Figure 1.1, with the interpretation of the cosines of the angle as the ratio between the adjacent side ($v_{1\ na\ 2}$) and the hypothenuse ($|v_1|$). Therefore, the projection of one vector on the other can be determined by knowing both vectors, or by knowing the vector which is projected and the angle between the vectors.

If we require the vector which represents the projection of vector $v_1$ on vector $v_2$ the previously obtained result will be multiplied with the unit vector in the direction of vector $v_2$

$$v_{1na2} = \cos(\theta)|v_1| \frac{v_2}{|v_2|} = \frac{v_1 \cdot v_2}{|v_2|} \frac{v_2}{|v_2|} \tag{1.5}$$

Now the result is a vector.

The cross product $v_1 = (x_1\, y_1\, z_1)$ i $v_2 = (x_2\, y_2\, z_2)$ is a vector defined as

$$v_1 \times v_2 = \begin{bmatrix} \rho & \rho & \rho \\ i & j & k \\ x_1 & y_1 & h_1 \\ x_2 & y_2 & h_2 \end{bmatrix} = \begin{bmatrix} y_1 h_2 - y_2 h_1 \\ -x_1 h_2 + x_2 h_1 \\ x_1 y_2 - x_2 y_1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{1.6}$$

The geometric interpretation of the cross product of two vectors is a vector which is perpendicular to the plain in which the vectors $v_1$ i $v_2$ lie. The order of these two vectors in the cross product is important and is defined with the right hand rule. If the turn from vector $v_1$ towards vector $v_2$ are determined by the fingers of the left hand, than the thumb is in the direction of the result vector $v_3$.

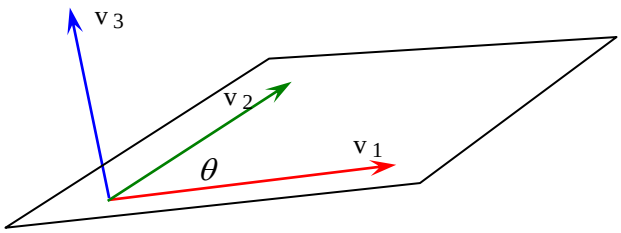



Figure 1.2. Cross product of vectors $v_1$ i $v_2$ results in vector $v_3$.

If we change the order of vectors $v_1$ i $v_2$ in the cross product, the result vector $v_3$ will be oriented towards the other side of the plane.

### *1.2. Matrix operations*

In computer graphics we will transform the vertices of objects which are given as single row matrices of dimensions 1×3 or 1×4. Matrix transformations are given in the form of matrices with dimensions of 3×3 or 4×4. Therefore, matrix multiplication is needed. In addition, transformations can be given as a series of matrices which need to be multiplied.

In inverse transformations it is also required to calculate the inverse of matrices, there this

functionality is also required for matrices of dimensions 3×3 or 4×4.

### *1.3. Barycentric coordinates*

Barycentric coordinates are useful for determining if a point is located inside of a triangle in 3D space. If the vertices of the triangle are A, B, C, then point T can be defined as:

$T= t_1 A+t_2 B+t_3 C$

where $t_1$, $t_2$, $t_3$ are barycentric coordinates. For barycentric coordinates the following expression is always true $t_1 + t_2 + t_3 = 1$. This can be expanded for all coordinates:

$A_x t_1+ B_x t_2 +C_x t_3=T_x$

$A_y t_1+ B_y t_2 +C_y t_3=T_y$

$A_z t_1+ B_z t_2 +C_z t_3=T_z$

Or in matrix form:

$$\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix}\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}=\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}, \text{ with the solution } \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}=\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix}^{-1}\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

Therefore, to determine barycentric coordinates, it will be required to calculate the inverse of a matrix. However, it can happen that the inverse of the matrix does not exist. This would happen for example if one of the vertices is in the origin of the coordinate system, or when all the vertices of the triangle lie in one of the following planes xy, xz or yz, which means that one coordinate would be zero. Such a matrix is singular and can not be inverted. In that case we use the additional constraint which needs to hold for barycentric coordinates ($t_1 + t_2 + t_3 = 1$), and replace the problematic row in the matrix with it.

### *1.4. Assignment*

1. Write a program which calculates the following:

$\vec{\mathbf{v}}_1 = \left(2\vec{i}+3\vec{j}-4\vec{k}\right)+\left(-\vec{i}+4\vec{j}-\vec{k}\right)$

$s = \vec{\mathbf{v}}_1 \cdot \left(-\vec{i}+4\vec{j}-\vec{k}\right)$

$\vec{\mathbf{v}}_2 = \vec{\mathbf{v}}_1 \times \left(2\vec{i}+2\vec{j}+4\vec{k}\right)$, where × is the cross product,

$\vec{\mathbf{v}}_3 = \left|\vec{\mathbf{v}}_2\right|$, unit vector,

$\vec{\mathbf{v}}_4 = -\vec{\mathbf{v}}_2$, vector of the opposite direction

$$\mathrm{M}_1=\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 5 & 1 \end{bmatrix}+\begin{bmatrix} -1 & 2 & -3 \\ 5 & -2 & 7 \\ -4 & -1 & 3 \end{bmatrix}$$

$$\mathrm{M}_2=\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 5 & 1 \end{bmatrix}.\begin{bmatrix} -1 & 2 & -3 \\ 5 & -2 & 7 \\ -4 & -1 & 3 \end{bmatrix}^T$$, where *T* denotes the transpose

$$M_3 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 5 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 5 & -2 & 7 \\ -4 & -1 & 3 \end{bmatrix}^{-1}$$

2. Write a program which will allow the user to enter the information about three linear equations with three variables (it is not required to parse the input, simply assume that the variables are always x, y, z, and that the user will enter the first enter the data for the first equation, then for the second equation, and finally for the third equation). The program needs to write out the solution of this system. For example it the user enters 1, 1, 1, 6, -1, -2, 1, -2, 2, 1, 3, 13, the system is the defined as:

$$\begin{aligned} 1 \cdot x + 1 \cdot y + 1 \cdot z &= 6 \\ -1 \cdot x - 2 \cdot y + 1 \cdot z &= -2 \\ 2 \cdot x + 1 \cdot y + 3 \cdot z &= 13 \end{aligned}$$

And the program needs to give the following solution [x y z] = [1 2 3]

3. Write a program which will allow the user to enter the data for the vertices of a triangle (A, B and C) and an additional point T in the 3D space (it is required to enter the x, y, and z coordinated for each of the points). The program needs to output the barycentric coordinates of point T.

Remark: directly use the afore described procedure and ignore any possible problems which can occur if the points of the triangle are not specified correctly.

# 2. Line

## *2.1.Homogenous coordinates*

A point from the *n*-dimensional space can be mapped to a point in the (*n*+1) dimensional homogenous space. A homogenous point from the (*n*+1)-dimensional space can be projected to the a point in the *n*-dimensional space. Let's observe an example in the 2-dimensional space and the corresponding 3-dimensional homogenous space.

Mapping:

$$V(x\ y) \rightarrow V'(x'\ y'\ h),$$

Point V is located in the 2D space, while V' is located in the 3D-space, and:

$$x' = hx, \qquad (2.1)$$
$$y' = hy.$$

Projection:

$$V'(x'\ y'\ h) \rightarrow V(x\ y),$$

- calculated as:

$$x = x'/h, \qquad (2.2)$$
$$y = y'/h.$$

Component *h* is called the proportionality factor or the homogenous coordinate. The value of the homogenous coordinate *h* is arbitrary, and is usually set to $h = 1$. If $h = 0$, then the point would be located in infinity in the *n*-dimensional space. If the lines are parallel in the *n*-dimensional space then the lines are parallel in the homogeneous (*n*+1) dimensional homogeneous space, which is an important property.

Why is the homogenous space called that way?. Equation of a line in 2D is

$$ax + by + c = 0, \qquad (2.3)$$

if in 1.3 we place 1.2 then

$$\frac{a\,x'}{h} + \frac{b\,y'}{h} + c = 0,$$

which gives

$$ax' + by' + ch = 0. \qquad (2.4)$$

Expression 2.4 is the equation of the line in the homogenous space. By its form this is a homogenous equation, therefore the name homogenous space.

### *2.2. Line equation*

A line is defined by two points, for example points $V_1$ i $V_2$. We use the homogenous space

$V_1=(x_1\ z_1\ h_1)$, $V_2=(x_2\ y_2\ h_2)$.

The vector form of the line equation is given by the cross product

$$P=V_1\times V_2=\begin{bmatrix}\rho i & \rho j & \rho k\\ x_1 & y_1 & h_1\\ x_2 & y_2 & h_2\end{bmatrix}=\begin{bmatrix}y_1h_2-y_2h_1\\ -x_1h_2+x_2h_1\\ x_1y_2-x_2y_1\end{bmatrix}=\begin{bmatrix}a\\ b\\ c\end{bmatrix} \tag{2.5}$$

The analytic form of the equation, with $h_1=h_2=1$ is

$$\frac{y-y_1}{y_2-y_1}=\frac{x-x_1}{x_2-x_1}$$

or

$$\left(y_1-y_2\right)x+\left(-x_1+x_2\right)y+x_1y_2-x_2y_1=ax+by+c=0 \tag{2.6}$$

The parametric form is defined as

$$P=\left(V_2-V_1\right)t+V_1,\quad \text{or by coordinates}\quad \begin{aligned}x&=\left(x_2-x_1\right)t+x_1,\\ y&=\left(y_2-y_1\right)t+y_1,\\ h&=\left(h_2-h_1\right)t+h_1.\end{aligned} \tag{2.7}$$

In this case the parameter value $t = 0$ is assigned to point $V_1$ while the parameter value $t = 1$ is assigned to point $V_2$.

Figure 1.1. shows the assignment of parameter $t$ to various parts of the line.

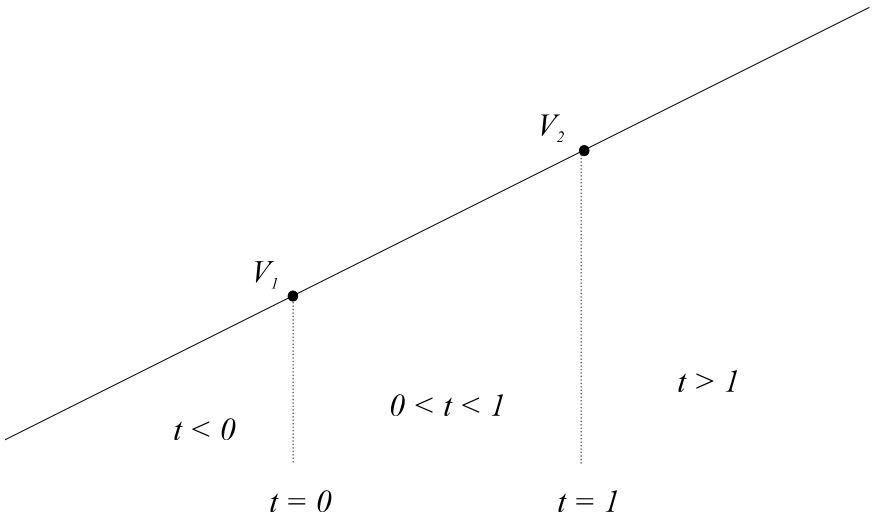



Figure 2.1. Value of parameter $t$ and various parts of the line.

### *2.3. Testing the relation between the line and point*

The scalar product of point $V(x\ y\ 1)$ and line $P(a\ b\ c)^{\tau}$ determines the relation between the point and line, with the convention

$$V\ P = [x \quad y \quad 1]\begin{bmatrix} a \\ b \\ c \end{bmatrix} = ax + by + c = 0 \quad \begin{matrix} > 0 \text{ point } V \text{ is "above" line P} \\ = \text{ point } V \text{ is "on" line P} \\ < 0 \text{ point } V \text{ is "below" line P} \end{matrix}$$

### *2.4. Drawing lines on the screen*

If we observe a screen similar to the TV screen (raster-scan). Between two points of the screen it is required to draw a straight line. For this purpose a procedure is required which selects the points (pixels) which need to be coloured. The validity of this procedure is based on:

- straightness of the line,
- correctness of the end of line,
- constant density of points in the line and its independence of the direction and length,
- drawing speed.

#### *2.4.1. Bresenham procedure*

The Bresenham procedure is most commonly used to draw lines on a screen similar to the TV screen. The criterion for choosing the raster points consists of calculating the distance between the neighbouring raster points and the line (Figure 1.2.)

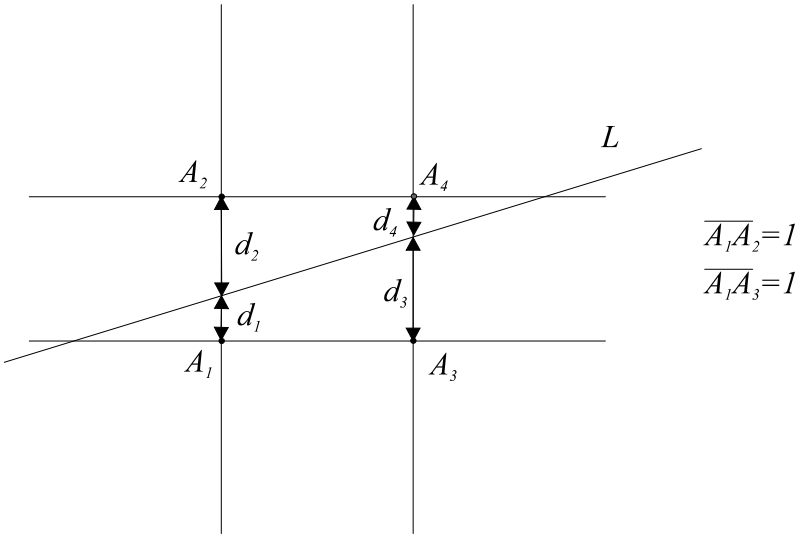



Figure 1.2. Bresenham procedure.

For line L it is required to draw points $A_1$ and $A_4$ since $d_1 < d_2$ i $d_4 < d_3$.

### *2.5. Assignment*

l. Enter coordinates of points $V_1(x_1\ y_1)$ i $V_2(x_2\ y_2)$.

2. By using the Bresenham procedure the a line between points $V_1$ and $V_2$.

3. Comparison with the *LINE* method from OpenGL.
Draw a line with coordinates $(x_1\ y_1+20)$ i $(x_2\ y_2+20)$ with the *LINE* command.

### *2.6. Assignment solution*

#### *2.6.1. Procedure:*

1. Enter the *x y* coordinates for the starting and ending point of the line $(x_1\ y_1)$, $(x_2\ y_2)$.
2. Calculate $x_0$, $y_0$ intervals of the line $x_0=x_2-x_1$, $y_0=y_2-y_1$.
3. Calculate the distance $D = y_0/x_0 - 0{,}5$.
4. Set the coordinates of the current point $x=x_1, y=y_1$.
5. For $i = 0$, and $< x_0$ repeat steps 6-8. Else go to step 9.
   6. Color the current point *x, y*.
   7. For $D > 0$ calculate $y=y+1$, $D=D-1$.
   8. Add $x=x+1$, $D=D+y_0/x_0$.
9. Comparison with the *LINE* command.
   Draw the line specified by the following coordinates $(x_1\ y_1+20)$ and $(x_2\ y_2+20)$.
10. *End.*

#### *1.6.2. Results*

Enter *x, y* of the starting point ? 10 10

Enter *x, y* of the ending point? 200 100

Comparison with the *LINE* command.

The described procedure works well for lines with an angle $0\text{-}45^0$. Do the required modification which will ensure that the procedure works for lines with any angle.

### *2.7. For exercise*

Which changes need to be done in the algorithm so that all variables become integer?

# 3. Drawing and filling a convex polygon

## *3.1. Specifying a polygon*

The geometric and topological data define a polygon (Figure 3.1). The geometric data is defined as the coordinates of the *n* vertices of the polygon,

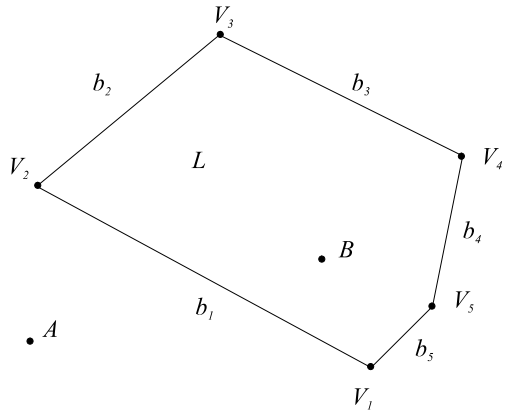
$V_i=(x_i\ y_i\ h_i)$, $\quad i=1\ ..\ n.$

The topological data is defined as the list of the vertices of a polygon,

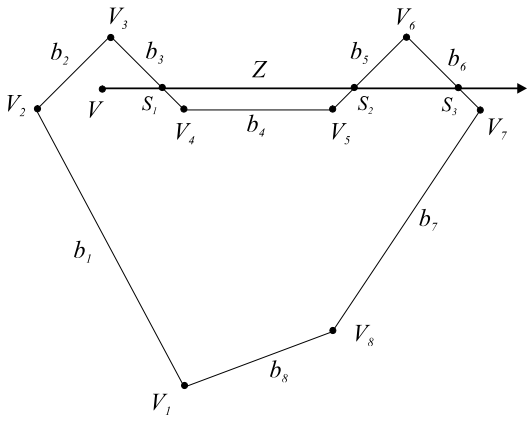
$L=(V_i)$, $\quad i=1\ ..\ n.$

The order of the vertices in the list *L* needs to be specified in the clockwise or counter clockwise direction. For example for the polygon in Figure 3.1.a. that would be

$L=(V_1\ V_2\ V_3\ V_4\ V_5)$ *or* $L=(V_1\ V_5\ V_4\ V_3\ V_2)$.




a) Konveksan poligon.




b) Konkavan poligon.

Figure 3.1. Polygons.

The equation of the line on which the edge $b_i$ of polygon *L* lies, or shorter the equation of edge *b*, is defined with the cross product of the starting and ending vertex of the edge,

$$b_i = V_i \times V_{i+1}, \quad i = 1\ ..\ n\text{-}1,$$
$$b_n = V_n \times V_1, \quad i = n. \qquad (3.1)$$

## *3.2. Checking the orientation of the edges*

The order of the vertices in the list can be requested as an input parameter, but it is not mandatory. With the assumption that the order of the vertices is given in the clockwise fashion, for a convex polygon the following criterion holds:

$$(\forall i)(V_j\, b_i < 0), \quad \begin{matrix} j=i+2 \ \text{za} \ \ i \le n-2, \\ j=i+2-n \ \text{za} \ \ i > n-2. \end{matrix} \quad i=1..n, \qquad (3.2)$$

If the order of the vertices is required to be in the clockwise direction, and criterion 3.2. does not hold, it is necessary to do the following corrections:

- reverse the order of the vertices in the list,
- repeat the calculation of the edge equations.

## 3.3. Testing the relation between a point and the polygon

For a convex polygon we can perform the check for the orientation of its vertices in the following was. Point *A* is outside the polygon *L* (Figure 3.1.a) if the following criteria do not hold

$$(\exists i)(Ab_i > 0),\ i = 1..\ n. \tag{3.3}$$

Point *B* is inside the polygon *L* since the following criterion is satisfied

$$(\forall i)(Bb_i < 0),\ i = 1..\ n. \tag{3.4}$$

## 3.4. Colouring the convex polygon

The edge $b_i$ of the polygon is determined by the starting vertex $V_i$ and the ending vertex $V_{i+1}$. The edges of the polygon need to be sorted into „left“ and „right“ edges by the following principle:

- if $y_i < y_{i+1}$ holds, the edge is classified as left,
- if $y_i > y_{i+1}$ holds, the edge is classified as left.

Determine all the intersections between the left and right edges with the line $y_p$(Figure 3.2). Determine $L_{max}$, the intersection with the left edges which has the largest *x* coordinate value. Determine $D_{min}$, the intersection with the right edges which has the smallest *x* coordinate value. Colour the line from $L_{max}$, $D_{min}$. Repeat the procedure for all lines of the screen. In the case of $L_{max} > D_{min}$ the polygon is bellow or above the polygon of the line $y_p$. Therefore, the area for selecting the line $y_p$ can be reduced to the area of the polygon, in between $y_{min}$,and $y_{max}$,

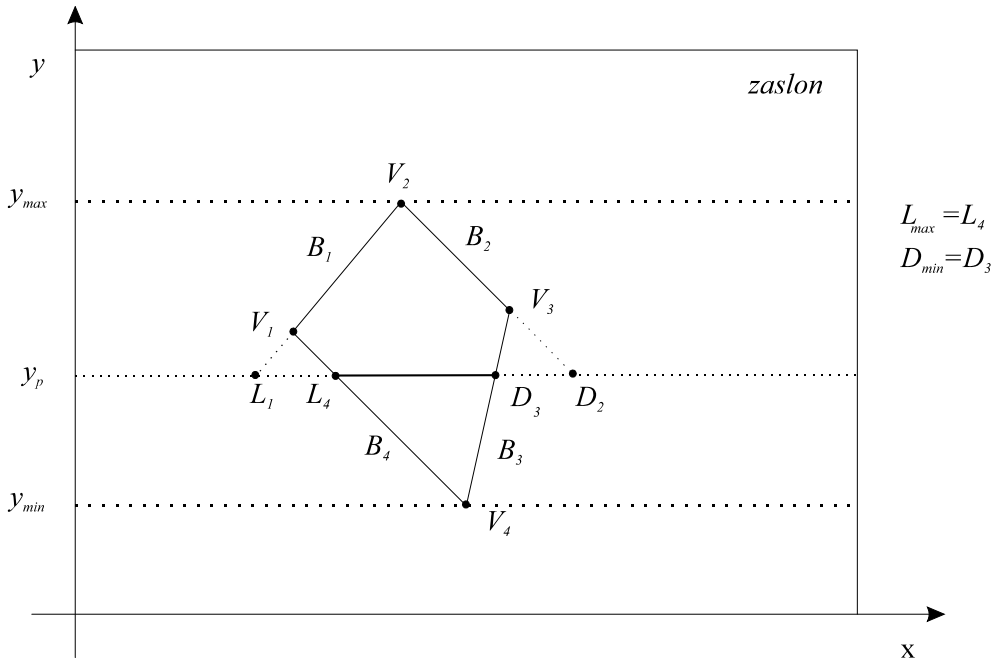



Figure 3.2. Colouring the convex polygon.

## 3.5. Assignment

1. Specify $n$ coordinates of the convex polygon, with the order given in the clockwise direction
2. Draw the polygon on the screen.
3. Calculate the coefficients for all polygon edges.
4. Specify the coordinates of a point $V$ and test the relation between it and the polygon.
5. Fill the polygon.

## 3.6. Solution of the assignment

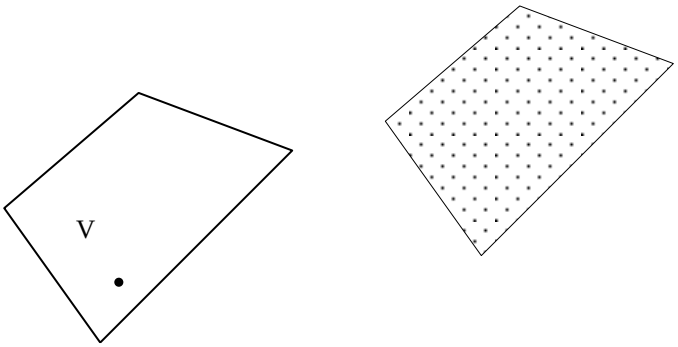
1. Enter the $n$ vertices of the convex polygon
2. Enter the $x$ $y$ coordinates of the vertices, $x(i)$, $y(i)$, $i = 0, n\text{-}1$. They should be provided in clockwise order. Determine $y_{min}$, $y_{max}$. and $x_{min}$, $x_{max}$. (entering the points can also be done dynamically with the use of the mouse).
3. Set $x(n)=x(\mathbf{0})$, $y(n)=y(0)$,
4. Draw the polygon.
5. Calculate the coefficients of the polygon edges.

   $a(i) = y(i) - y(i+1)$,
   $b(i) = -x(i) + x(i+1)$, $\quad i = 0, n\text{-}1$
   $c(i) = x(i) \cdot y(i+1) - x(i+1) \cdot y(i)$.
6. Enter the $x$ $y$ coordinates of point $V(x_1\, y_1)$.
7. Test the relation of point $V$ and polygon.
   If at leas for one of the edges the following relation holds
   $x_1\, a(i) + y_1\, b(i) + c(i) > 0, \quad i = 0, n\text{-}1$ then point $V$ is outside the polygon
   otherwise, the point $V$ is inside the polygon.
8. Filling the polygon.
   For all test lines $Y_o = y_{min}, y_{max}$ perform the steps 9-15. Then go to step16.
9. Set $L = x_{min}$, $D = x_{max}$ .
   10. For $i = 0, n\text{-}1$ repeat the steps 11-14. Then go to step 15.
      11. If $A(i)=0$ do not perform steps 12 -14.
      12. Calculate the $x$ coordinate of the intersection between the line $y_0$ and the $i$-th edge, $x_1=[\, -b(i)\, y_0 - c(i)] \,/\, a(i)$.
      13. Left edge.
          If $y(i) < y(i+1)$ then if $x_1 > L$ set $L = x_1$.
      14. Right edge.
          If $y(i) \geq y(i+i)$ then if $x_1 < D$ set $D = x_1$.
   15. If $L < D$ draw line $(L\, y_0)$, $(D\, y_0)$.
16. End.

*Results*

Enter the number of polygon vertices? *4*

| | | |
|---|---|---|
| Enter coordinate? | *50* | *200* |
| Enter coordinate? | *150* | *350* |
| Enter coordinate? | *300* | *150* |
| Enter coordinate? | *100* | *50* |
| Enter coordinate of point? | *150* | *200* |




POINT *V* IS INSIDE OF POLYGON !

POINT *V* IS OUTSIDE OF POLYGON !

# 4. Plane, body

## *4.1. Plane equation*

A plane is specified by three noncolinear points

$$V_1 = \begin{pmatrix} x_1 & y_1 & z_1 & h_1 \end{pmatrix},$$
$$V_2 = \begin{pmatrix} x_2 & y_2 & z_2 & h_2 \end{pmatrix},$$
$$V_3 = \begin{pmatrix} x_3 & y_3 & z_3 & h_3 \end{pmatrix}.$$

There are two basic plane equation forms: the analytic form and the parametric form.

## *4.2. Analytic form of the plane*

The analytic form of the plane is given as:

$$Ax + By + Cz + D = 0.$$

(with $h_1 = h_2 = h_3 = 1$) gives

$$\text{determinanta} \begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = Ax + By + Cz + D = 0. \qquad (4.1)$$

The coefficients of the plane equation, in the case when $n$ points lie exactly or approximately in the plane, are given with the following procedure.

$$A = \sum_{i=0}^{n-1} (y_i - y_j)(z_i + z_j)$$
$$B = \sum_{i=0}^{n-1} (z_i - z_j)(x_i + x_j) \qquad \begin{matrix} j = i+1 \quad \text{za} \;\; i \neq n \\ j = 0 \quad \text{za} \;\; i = n-1. \end{matrix}, \qquad (4.2)$$
$$C = \sum_{i=0}^{n-1} (x_i - x_j)(y_i + y_j)$$

For the $i$-th point with coordinates ($x_i$, $y_i$, $z_i$) which lies in the plane the following equation holds:

$$Ax_i + By_i + Cz_i + D = 0,$$

Which gives

$$D = -Ax_i - By_i - Cz_i.$$

The equation of the plane $R$ can be determined based on three points, specifically based on three dot products.

$$V_1 R = 0, \quad V_2 R = 0, \quad V_3 R = 0,$$

Or in matrix form:

$$\begin{bmatrix} & V_1 & \\ & V_2 & \\ & V_3 & \\ 0 & 0 & 0 & 1 \end{bmatrix} R = \begin{bmatrix} 0 \\ 0 \\ 0 \\ D \end{bmatrix}$$

from which follows

$$R = \begin{bmatrix} x_1 & y_1 & z_1 & h_1 \\ x_2 & y_2 & z_2 & h_2 \\ x_3 & y_3 & z_3 & h_3 \\ 0 & 0 & 0 & D \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \tag{4.3}$$

where *A, B, C* are the coefficients of the normal vector of that plane. In case when the three specified points for which we want to determine the plane are colinear (they lie on the same line) there is no single solution. If the plane is defined with points which are coplanar with planes *xy*, *xz*, *yz*, it will be necessary to specially consider and determine the normal plane vector (0 0 1), (0 1 0), (1 0 0).
The coefficients *A, B, C* of the normal vector on the plane can be determine as the cross product of two vectors ($V_2$ - $V_1$) and ($V_3$ - $V_1$):

$$\vec{\mathbf{n}} = \begin{pmatrix} A & B & C \end{pmatrix} = (V_2 - V_1) \times (V_3 - V_1) \tag{4.4}$$

where the order of the vertices determines the direction of the normal vector (towards „up“ or „down“). This formula is equivalent to formula 4.1

## *4.3. Parametric equation form of the plane*

The parametric form of the plane equation by the coordinates is a linear function of tow parameters: *u* and *w*.

$$\begin{aligned} x &= a_1 u + b_1 w + c_1, \\ y &= a_2 u + b_2 w + c_2, \\ z &= a_3 u + b_3 w + c_3, \\ h &= a_4 u + b_4 w + c_4, \end{aligned} \tag{4.5}$$

From 4.5 follows the matrix form

$$V = \begin{bmatrix} x & y & z & h \end{bmatrix} = \begin{bmatrix} u & w & 1 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \end{bmatrix}, \tag{4.6}$$

or

$$V = \begin{bmatrix} x & y & z & h \end{bmatrix} = \begin{bmatrix} u & w & 1 \end{bmatrix} K, \tag{4.7}$$

where *K* represents the characteristic matrix of the plane.
If points $V_1$, $V_2$ i $V_3$ are assigned the parameter values

$$V\,R = Ax + By + Cz + D = 0 \quad \begin{matrix} > 0 \text{ point } V \text{ is "above" plane R} \\ = \text{ point } V \text{ is "in" the plane R} \\ < 0 \text{ point } V \text{ is "below" pane R} \end{matrix}$$

$$V\,R = Ax + By + Cz + D \begin{cases} > 0 \text{ točka je iznad ravnine,} \\ = 0 \text{ točka u ravnini,} \\ < 0 \text{ točka je ispod ravnine,} \end{cases}$$

Then from 4.6 and 4.7 follows

$$K = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_1 & y_1 & z_1 & h_1 \\ x_2 & y_2 & z_2 & h_2 \\ x_3 & y_3 & z_3 & h_3 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \end{bmatrix} \tag{4.8}$$

## 4.4. Testing the relation between a point and a plane

The dot product between a point and a plane (similarly as in the case of a point and a line) determines the relation between the point and the plane, with the following agreement:

$$V\,R = Ax + By + Cz + D \begin{cases} > 0 \text{ točka je iznad ravnine,} \\ = 0 \text{ točka u ravnini,} \\ < 0 \text{ točka je ispod ravnine,} \end{cases} \tag{4.9}$$

## 4.5. Intersection of a line and a plane

The intersection is a common point $V$ between the line $P$ and the plane $R$,

$$V \cdot P = 0 \quad \text{i} \quad V \cdot R = 0,$$

Based on these two conditions we can calculate a point for which the line in its parametric form intersects with the given plane. We will determine the parameter value $t$ for which the point simultaneously belongs to the line and the plane.

## 4.6. Specifying a body

The body is specified by the geometric and topological data. Geometric data are vertices, or their coordinates. Topological data represents connection of vertices, i.e. the lists that determines the surface, the polygons, the edges, and the corresponding vertices. In this example, the organisation shown in Figure 4.1 was selected. This is one of the ways of organising topological and geometric data, but depending on the further procedures that will be performed on the object, different ways of storing the object's data structure can be used.

Triangles are the most commonly used polygons in the body's description, because a plane is determined by exactly three points. For example, if the body would be specified by using quadrangles, then all four points to not necessarily line in the same plane, which can cause certain problems.

It is important to specify the data structure in a way that the sequence of edges for a particular polygon is known. The order of edges can be specified in the counter clockwise direction, when observed from outside the body. The order of the edges determines the orientation of the normals of the polygons. For a certain body the normals of all polygons must be oriented in the same direction, for example all should be oriented towards the interior of the body or towards outside the body.

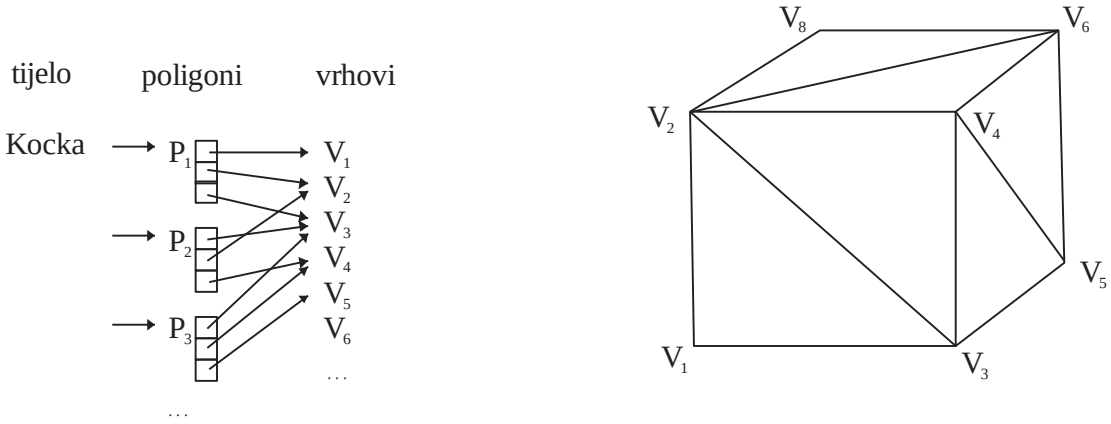



Figure 4.1. The body and the adherent data structures.

The body can consist out of several surfaces of different colours. In that case the polygons with the same properties would have to be grouped into the same group.

For the example in Figure 4.1. the list of vertices (given that the order of the vertices is defined to be counter clockwise when observed from outside the body) would be:

| **Cube:** | | **Data in the file about the vertices and polygons** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $P_1$ | $V_0\ V_1\ V_2$ | v | 0.0 | 0.0 | 0.0 | f | 1 | 3 | 2 |
| $P_2$ | $V_2\ V_1\ V_3$ | v | 0.0 | 0.0 | 1.0 | f | 3 | 4 | 2 |
| $P_3$ | $V_2\ V_3\ V_4$ | v | 1.0 | 0.0 | 0.0 | f | 3 | 5 | 4 |
| $P_4$ | $V_4\ V_3\ V_5$ | v | 1.0 | 0.0 | 1.0 | f | 5 | 6 | 4 |
| $P_5$ | $V_4\ V_5\ V_6$ | v | 1.0 | 1.0 | 0.0 | f | 5 | 7 | 6 |
| $P_6$ | $V_6\ V_5\ V_7$ | v | 1.0 | 1.0 | 1.0 | f | 7 | 8 | 6 |
| $P_7$ | $V_6\ V_7\ V_0$ | v | 0.0 | 1.0 | 0.0 | f | 7 | 1 | 8 |
| $P_8$ | $V_0\ V_7\ V_1$ | v | 0.0 | 1.0 | 1.0 | f | 1 | 2 | 8 |
| $P_9$ | $V_0\ V_2\ V_4$ | # | comment | | | f | 1 | 5 | 3 |
| $P_{10}$ | $V_0\ V_4\ V_7$ | | | | | f | 1 | 7 | 5 |
| $P_{11}$ | $V_1\ V_5\ V_3$ | | | | | f | 2 | 4 | 6 |
| $P_{12}$ | $V_1\ V_7\ V_5$ | | | | | f | 2 | 6 | 8 |

## *4.7. Orientation of the plane normal*

The normal vector **n** of the plane *R* can be oriented

- towards the inside of the body,
- towards outside the body.

Three neighbouring non-colinear vertices $V_{i-1}$, $V_i$, $V_{i+1}$ in the vertex list determine the coefficients of the equation of plane *R*,

$$Ax + By + Cz + D = 0, \tag{4.10}$$

And with that also the component of normal **n** of plain *R*

$$\mathbf{n} = \begin{bmatrix} A & B & C \end{bmatrix}. \tag{4.11}$$

With the given assumptions:

- the order of vertices $V_{i-1}$, $V_i$, $V_{i+1}$ is specified in the counter clockwise direction, when observed from outside the body
- the middle vertex is located on a convex part of the polygon (always true for triangles)

the normal vector **n** of plane *R* is directed towards the outside of the body.

## 4.8. Testing the relation between a point and a convex body

In the case of a convex body and when all normal vectors are directed towards outside of the body, point V is inside of the body if

$$(\forall i)(VR_i < 0),\ i = 1..n, \tag{4.12}$$

that is, point V outside if

$$(\exists i)(VR_i > 0),\ i = 1..n. \tag{4.13}$$

If the body is concave, then a line is pulled from the point and the number of intersections between the body and the line is determined. If the number of intersections is:

- even, the point is outside of the body,
- odd, the point is inside of the body.

## 4.9. Recording polygonal objects

Recording three-dimensional objects is usually defined by the specifications of the program equipment manufacturers. For example, those would be records with extensions .3ds, .max, .dxf, .ply, .obj, .xgl, .stl, .wrl, .iges … .Depending on the record, it is possible to record only triangles or polygons with multiple vertices, groups of polygons, colours, textures, normals of polygons and vertices, curves, surfaces, object construction details, scene data, light sources, animation data, and many other. In this assignment the .obj record will be used.

## 4.10. Assignment

A simple example is given which consists out of a subset of the Wavefront object record (.obj). The records contain the list of vertices and their coordinates, as well as the list of polygons with the corresponding vertex indices.

1. Load the default body from the file (for the example in Figure 4.1)
   - during the first pass count the number of vertices and polygons (first letter *v*, *f* or #)
   - allocate the necessary memory space
   - read the necessary geometric and topological data (vertices and polygons)
   - set the coordinates of the test point *V*.
2. Determine the $A_i$, $B_i$, $C_i$, $D_i$ coefficients of the plane equation for each polygon of the body. Use the formula 4.1.:

$$A = (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1)$$
$$B = -(x_2 - x_1)(z_3 - z_1) + (z_2 - z_1)(x_3 - x_1)$$
$$C = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$
$$D = -x_1 A - y_1 B - z_1 C$$

3. For the test point *V* determine whether it is located inside or outside of the convex body. Use the conditions 4.12, 4.13. Set the *z* coordinate to *z*=0 and draw the body (if required scale the image).

## 5. View transformation and perspective projection

The system of the scene is three dimensional (Figure 5.1). The system of the eye is a three dimensional system where the *z* axis is oriented in the direction of the view, so to represents the depth of the image. The display system is a two dimensional system located in the projection plane *R*.

Mapping the points from the scene system to the eye system is called the view transformation. The projection of the points from the eye system to the display system can be done through a parallel or perspective projection.

The used labels:

| | |
|---|---|
| $x_S\ y_S\ z_S$ | - scene system, |
| $x_O\ y_O\ z_O$ | - eye system, |
| $x_P\ y_P$ | - screen system, |
| *O* | - eye, position of the observer in the scene, |
| *G* | - gaze, the point towards which the gaze of the observer is directed, |
| *H* | - distance of the projection plane to the eye, *H*=d(*O*,*G*), |
| *R* | - projection plane, point *G* lies in the projection plane, |

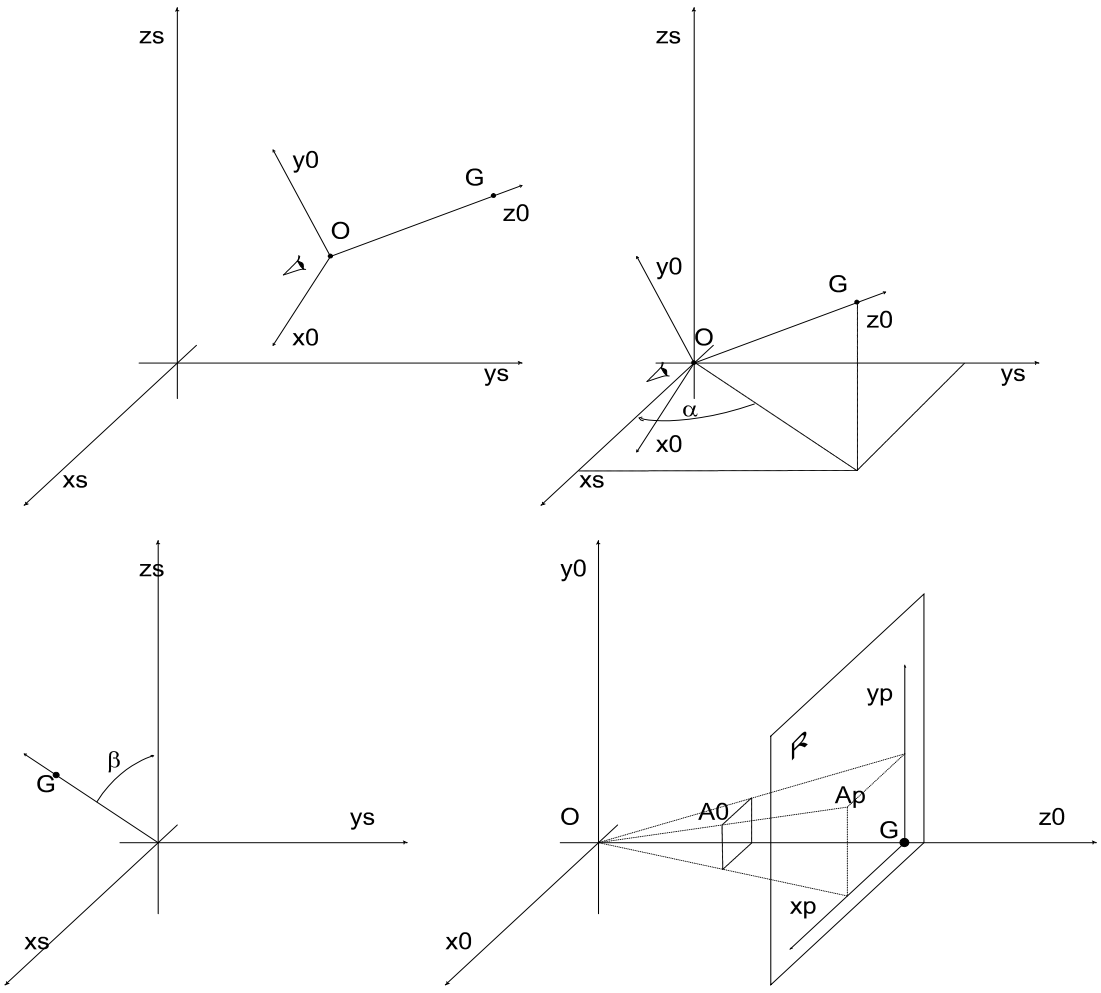



Figure 5.1. Scene, eye and display scenes and systems.

## *5.1. View transformation*

For the view transformation it is required to determine a matrix T so that the following holds

$$A_0 = A_S T \tag{5.1}$$

Matrix $T$ is composed of five matrices of elementary transformations which are:
$T_1$ – shift of the origin towards point $O$,
$T_2$ – rotation for the angle α around the $z$ axis,
$T_3$ - rotation for the angle β around the $y$ axis,
$T_4$ - rotation for the angle 90° around $z$ axis,
$T_5$ – change of the sign on the $x$ axis.

Points $O\begin{pmatrix} x_o & y_o & z_o \end{pmatrix}$ and $G\begin{pmatrix} x_g & y_g & z_g \end{pmatrix}$ are measured in the scene system. The coordinates of point $O$ are determined by matrix $T_1$

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_o & -y_o & -z_o & 1 \end{bmatrix}. \tag{5.2}$$

The effect of $T_1$ on $G$ gives

$$G_1 = GT_1 \text{ or } \begin{aligned} x_{g1} &= x_g - x_o \\ y_{g1} &= y_g - y_o \\ z_{g1} &= z_g - z_o \end{aligned}$$

Matrix $T_2$ is defined as

$$T_2 = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.3}$$

where

$$\sin\alpha = \frac{y_{g1}}{\sqrt{x_{g1}^2 + y_{g1}^2}}, \qquad \cos\alpha = \frac{x_{g1}}{\sqrt{x_{g1}^2 + y_{g1}^2}}$$

The effect of matrix $T_2$ on $G_1$ gives

$$G_2 = G_1 T_2 \text{ **or** } \begin{aligned} x_{g2} &= \sqrt{x_{g1}^2 + y_{g1}^2} \\ y_{g2} &= 0 \\ z_{g2} &= z_{g1} \end{aligned}.$$

Matrix $T_3$ is defined as

$$T_3 = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.4}$$

where

$$\sin\beta = \frac{x_{g2}}{\sqrt{x_{g2}^2 + z_{g2}^2}}, \qquad \cos\beta = \frac{z_{g2}}{\sqrt{x_{g2}^2 + z_{g2}^2}}.$$

The effect of matrix $T_3$ on $G_2$ gives

$$G_3 = G_2 T_3 \text{ or } \begin{aligned} x_{g3} &= 0 \\ y_{g3} &= 0 \\ z_{g3} &= \sqrt{x_{g2}^2 + z_{g2}^2} \end{aligned}$$

After these transformations the *z* axis of the eye coordinate system $z_o$ now coincides with the *z* axis of the scene coordinate system $z_s$. Additionally, it is required to achieve that the *x* and *y* axes also coincide between the two system. The display coordinate system is usually set in a way that the origin is places in the left upper corner of the screen, the *x* axis is set horizontally towards right, the *y* axis is set vertically towards the bottom of the screen, while the *z* axis is orientated towards the observer. The coordinate system of the eye is therefore left-handed, while the scene coordinate system is right handed. The matrices required for coinciding the remaining two axes between the two systems are:

$$T_4 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_5 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

Matrix *T* is the obtain by multiplying all the matrices

$$T = T_1 T_2 T_3 T_4 T_5 . \tag{5.6}$$

## 5.2. Perspective projection

The goal is to determine the P matrix which will project the points from the eye system into the projection plane (Figure 5.1), i.e. the projection system

$$A_P = A_0 P . \tag{5.7}$$

The distance between the projection plane and the eye is

$$H = \sqrt{\left(x_o - x_g\right)^2 + \left(y_o - y_g\right)^2 + \left(z_o - z_g\right)^2} = z_{g3} . \tag{5.8}$$

From the similarity of triangles follows

$$x_p = \frac{x_0}{z_0} H, \quad y_p = \frac{y_0}{z_0} H . \tag{5.9}$$

Expression 5.9 can be written in a matrix form

$$A_p' = A_0 P$$

Or by coordinates

$$\begin{bmatrix} x_p' & y_p' & z_p' & h_p' \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\begin{aligned} x_p' &= x_0 \\ y_p' &= y_0 \\ z_p' &= 0 \\ h_p' &= \frac{z_0}{H} \end{aligned} \qquad (5.10)$$

From 5.10 follows 5.9, which represents the return into the non-homogenous space

$$x_p = \frac{x_p'}{h_p'} = \frac{x_0}{z_0} H, \qquad y_p = \frac{y_p'}{h_p'} = \frac{y_0}{z_0} H.$$

## *5.3. Assignment*

Specify a polygon and perform the view transformation and perspective projection.

1. From a file read the coordinates of the eye, gaze, and the vertices of the polygon in the scene system. The gaze is usually set to the origin of the scene G = (0 0 0) or in the centre of the body (polygon). The eye is a point from which the scene is observed and which will depend on the size of the object. If the coordinates are in the range (-1, 1) the eye can be in O = (1 1 3). The eye must not be specified inside of the object.
2. Calculate the projection matrix T by using the formula 5.6.
3. Calculate the perspective projection matrix P.
4. Perform the transformation and projection of the specified polygon vertices.
5. Draw the polygon
6. Repeat the steps 1-5 for the body from the previous assignment

# 6. Drawing Bezier spatial curves

Bezier curves can be defined in two ways: by using Bezier weight functions and by using Bernstein polynomials.

## *6.1. Bezier curves defined by Bezier's weight functions*

The Bezier curve is defined through the Bezier's weight functions:

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{a}_i f_{i,n}(t) \tag{1}$$

where:

**p**(t) points on the curve (vector notation x(t), y(t), z(t)),
**a**$_i$ vectors of the control polygon (between the specified points),
$f_{i,n}$ base (weight) functions of n-th degree.

The base functions are defined through the polynomials:

$$f_{i,n} = \frac{(-t)^i}{(i-1)!} \frac{d^{i-1}\Phi_n(t)}{dt^{i-1}}, \qquad \Phi_n(t) = \frac{1-(1-t)^n}{-t} \qquad i = 1..n \tag{2}$$

Or by the recursive function:

$$f_{i,n}(t) = (1-t) f_{i,n-1}(t) + t\, f_{i-1,n-1}(t), \qquad f_{0,0}(t) = 1,\ f_{k+1,k}(t) = 0,\ f_{-1,k}(t) = 1 \tag{3}$$

## *6.2. Bezier curves defined by Bernstein polynomials*

The Bezier curve is defined by the Bernstein weight functions as:

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{r}_i b_{i,n}(t) \tag{4}$$

where:

**p**(t) point on the curve (vector notation x(t), y(t), z(t)),
**r**$_i$ vertices of the control polygon (the specified points),
$b_{i,n}$ base (weight) functions of n-th degree.

The base functions are defined by the Bernstein polynomials

$$b_{i,n} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} \tag{5}$$

Example of the weight functions for four control points (cubic curve):

$$b_{0,3} = (1-t)^3$$
$$b_{1,3} = 3t(1-t)^2$$
$$b_{2,3} = 3t^2(1-t)$$
$$b_{3,3} = t^3$$

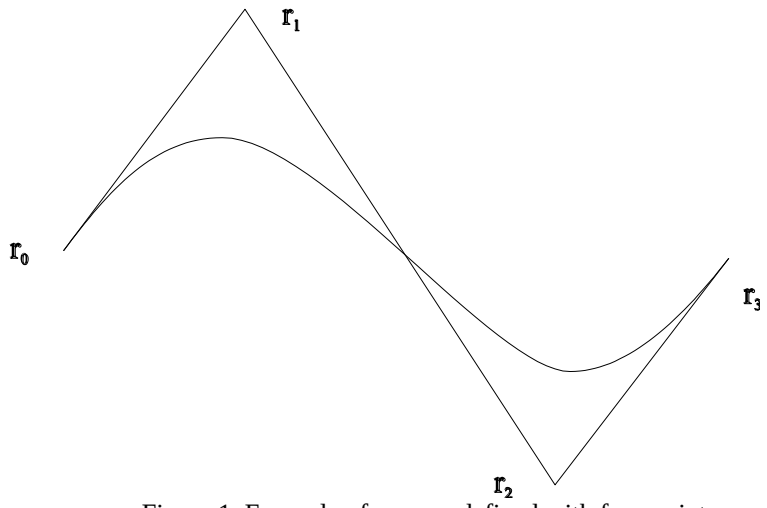



Figure 1. Example of a curve defined with four points.

## *6.3. Assignment:*

1. From a file read the n+1 points specifying the control polygon.
2. Draw the polygon
3. Change the parameter *t* from zero to one with the step 0.01.
4. For each value *t* determine the coordinates x(t), y(t) and z(t) of the curve by using formulas (4) and (5), and plot the points.
5. Use the obtained procedure to determine the path of the eye (or gaze or body) from the previous assignment. Plot the wireframe of the object with the removal of the rear polygons. Perform the removal of the polygons based on the angle between the normal vectors and the vector towards the observer.

# 7. Shading

All the procedures from the previous chapters will be included in the shading process. Specifically, these will be the procedure for building the body described in Chapter 4, as well as the view transformation and projection from Chapter 5. In this assignment, the function of OpenGL or (Direct3D) will be used for the view transformation to compare it with the procedure from the fifth assignment. The aim of this assignment is to show the body without shading, by using constant and Gouraud shading, as well as to remove rear polygons
For constant shading it is necessary to know the normal vectors of polygons. For Gouraud's shading, it is necessary to determine the normal vectors in the vertices of the object. Based on the normal vector, the intensity is determined and the polygon is coloured. To improve the time required to display the object it is recommend to remove all rear polygons.

## *7.1. Removing hidden lines and polygons*

Various methods exist for the removal of hidden lines and surfaces. These methods are based on geometric calculations in the space of the scene, or in the projection plane. The basic procedures use multiple calculations and are thus computationally expensive. For the purpose of improving the efficiency of the methods, fast procedures which will reduce the duration of the basic procedures will be used. In these fast procedures we include:

- removing rear polygons,
- minimax tests.

Removing rear polygons

Removing the rear polygons can be done in the scene system or in the display system.

Removing rear polygons in the scene system can be done based on the observation of the angle between the normal vector of the polygon and the vector from the polygon (the centre of the polygon) towards the eye (Figure 7.1.). If normal vectors of all polygons are oriented towards inside of the body, then the angle between the considered vectors determines whether the polygon is a rear polygon

- 0°- 90° if the polygon is not a rear polygon,
- 90°-180° if the polygon is a rear polygon.

We can observe that it is not necessary to calculate the angle between the normal of the plane and the vector towards the eye. It is sufficient to test on which side of the plane (on which the considered polygon lies) is the eye point located. Therefore, if the dot product between the eye point and the plane equation is (4.9):

- positive, the polygon is visible,
- negative or zero, the polygon is not visible.

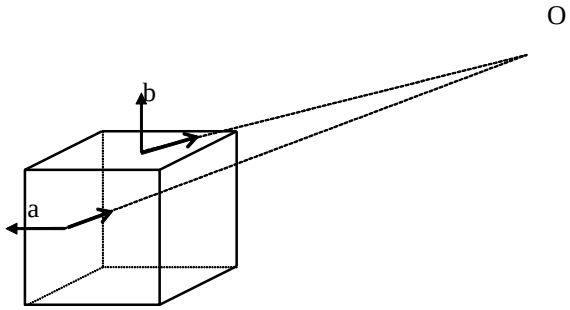



Figure 7.1. Rear polygons (a) the polygon is a rear polygon since the angle is larger than 90°, (b) the polygon is not a rear polygon since the angle is less than 90°.

The angle between two vectors **a** i **b** can be determined based on the dot product

$$\mathbf{ab} = |\mathbf{a}||\mathbf{b}|\cos\alpha \tag{7.1}$$

The other way of removing rear polygons is in the display system, i.e. in the projection plane. It is assumed that the order in which the polygon vertices are specified in clockwise order when observed in the display scene and from outside of the body. In this procedure, in the projection plane the order of the vertices from rear polygons will be counter clockwise (Figure 7.2.).

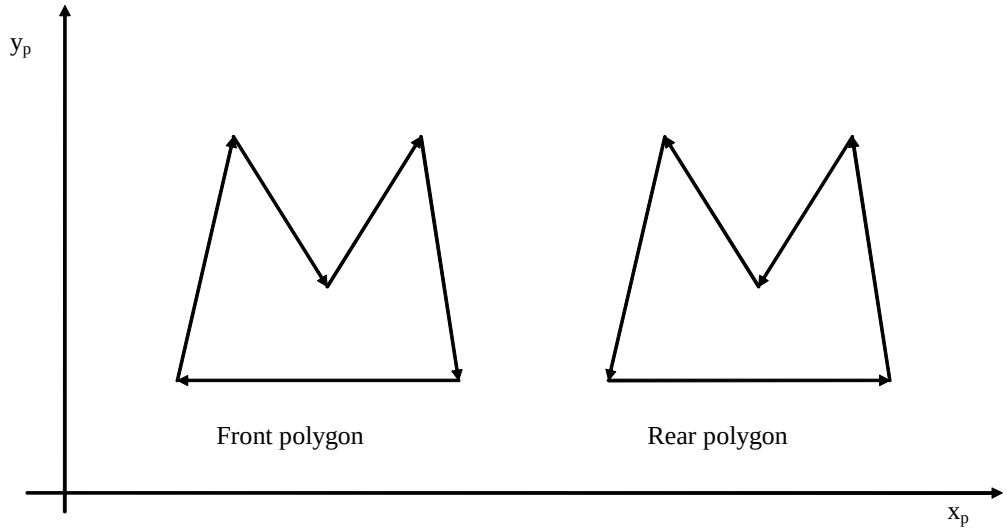



Figure 7.2. Determining rear polygons based on their orientation in the projection

Let *P* be the list of polygon vertices in the display system,

$$P = \left(V_1 \quad V_2 \quad \ldots \quad V_n\right), \tag{7.2}$$

polygon P is a rear polygon if

$$\left(\exists B_i\right)\left(\forall_j\right)\left(B_i V_j\right) \geq 0, \qquad i = 1\ldots n, \;\; j = 1\ldots n. \tag{7.3}$$

**Minimax tests**

Minimax test are performed in the display system, in the projection plane. The possible test include:

- body- body,
- polygon- body,
- edge- body,
- polygon- polygon,
- edge - polygon
- edge -edge.

In Figure 7.3. the minimax test for body-body is shown.

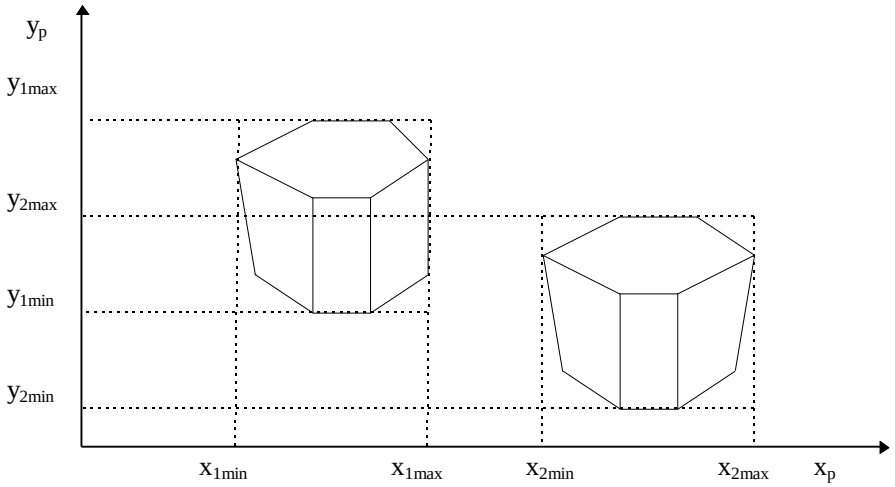



Figure 7.3. Minimax test for a body.

For bodies $T_1$ and $T_2$ it is required to determine the minimum and maximum *x, y* coordinates. These are:

- ($x_{1min}$, $y_{1min}$) ($x_{1max}$, $y_{1max}$) for body $T_1$,
- ($x_{2min}$, $y_{2min}$) ($x_{2max}$, $y_{2max}$) for body $T_2$.

Minimax coordinates of a body define rectangles. If the rectangles in the projection plane do not overlap, then body $T_1$ does not cover body $T_2$.

## *7.2. Range of the coordinates and the position of the body*

The body can be defined with different ranges of coordinates and on different positions in the scene. Minimax coordinates of the body are required when normalising the size of the body to a given working space, as well as for the translation of the body into the origin of the scene (or some other position)

$size\_x = x_{max} - x_{min}$ $center\_x = (x_{max} + x_{min})/2$
$size\_y = y_{max} - y_{min}$ $center\_y = (y_{max} + x_{min})/2$
$size\_z = z_{max} - z_{min}$ $center\_z = (z_{max} + z_{min})/2$

If we want to position the body into the working area [-1, 1], we have to translate the entire body for (-center_x-center_y -center_z), and to scale it with 2/max(size_x, size_y, size_z).

If we do not know the position and the range of the body coordinates, but we want to place it in our working are (for example [-1, 1]) we will perform the abovementioned procedure, so that we are not required to change the parameters of the view transformation or projection.

If the scene contains several objects, a similar sub procedure can be performed on each of the bodies (so that all objects would be in the same range), and then we can scale and move the objects in the given frame of the scene.

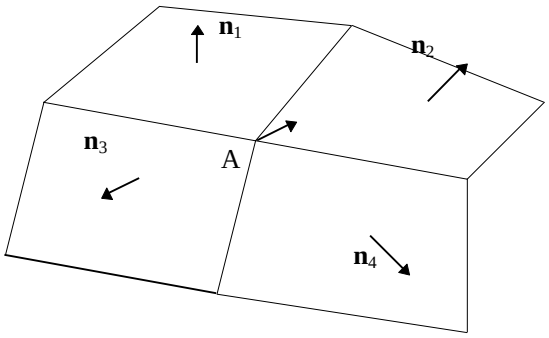



Figure 7.4 Determining the normal of vertex A.

## 7.3. Determining the normals in vertices

For the shading models it is required to know the normal vectors in the vertices of the body. The normals in the vertices are calculated based on the normals of the polygon (Figure 7.4.). The normals of the polygon have to be in unit length (all the components divided by the root of the sum of their components raised to the power of two). The normal of the vertex is calculated as the sum of the normals of the polygons which contain that vertex, divided by the number of those polygons:

$$\mathbf{n}_{Ax} = \frac{1}{4}\left(\mathbf{n}_{1x} + \mathbf{n}_{1x} + \mathbf{n}_{3x} + \mathbf{n}_{4x}\right), \tag{7.4}$$

where $\mathbf{n}_{1x}$, $\mathbf{n}_{2x}$, $\mathbf{n}_{3x}$, $\mathbf{n}_{4x}$, are $x$ components of normals $\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$, $\mathbf{n}_4$, which are used to determine the normal of the vertex. The other components are calculated in the same way.

### 7.4. Illumination model

Based on the ***illumination model*** we determine the intensity in a concrete point. We will use Phong's illumination mode. The shading procedure is then added to the illumination model. The basic shading procedures are constant shading, Gouraud shading and Phong shading.

Phong's illumination modes performs a simple approximation of the physical model. The Phong illumination model is defined as a linear combination of three component: ambient, diffuse and specular.

***Ambient component***. Usually, the calculation of the reflection contributions from other surfaces in the scene is not performed, therefore the ambient component represents and approximation of that contribution. In case when the ambient component is not present, read polygons will be coloured black. However, due to the reflection of the light from other surfaces, these surfaces are not completely black. Since these polygons do not have to be rear polygons for the observer, they will be visible. The ambient component is defined as

$$I_g = I_a k_a \tag{7.5}$$

where $I_a$ the intensity we want for the rear surfaces (0-255), and $k_a$ a coefficient ($0 \le k_d \le 1$).

***Diffuse component*** is

$$I_d = I_i k_d \cos\varphi = I_i k_d (\mathbf{LN}), \qquad 0 \le \varphi \le \pi, \tag{7.6}$$

where $I_i$ is the intensity of the light source, φ the angle between the normal vector of the surface and the vector towards the light source, $k_d$ and empiric reflection coefficient ($0 \le k_d \le 1$). **LN** denotes the dot product between the unit vectors $|\mathbf{L}| = 1$ i $|\mathbf{N}| = 1$. When the angle φ is larger than 90° the dot product is negative, which means that the surface is a rear surface and should not be illuminated. In that case the diffuse component is equal to zero. The total contribution of the ambient and diffuse component is

$$I_d = I_a k_a + I_i k_d (\mathbf{LN}), \quad 0 \le \varphi \le \pi, \tag{7.7}$$

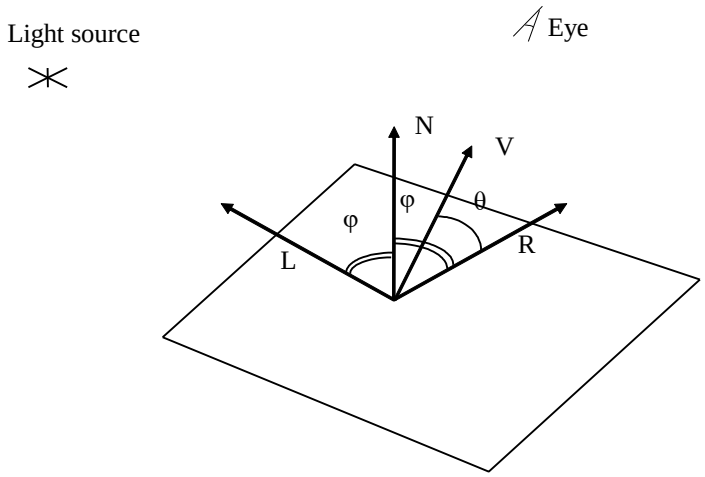



Figure 7.4Position of the normal **N**, vector towards the light source **L**, The vector towards the eye **V** and the reflected ray **R**.

***Specular component*** depends on the angle θ between the reflected ray **R** and the ray towards the observer **V**

$$I_s = I_i k_s \cos^n \theta = I_i k_s (\mathbf{RV})^n \tag{7.8}$$

where $n$ is the index of the roughness of the surface $n \in (0, \infty)$. The reflected rays lie in the plane which is defined by the normal and the entering ray, and encloses the angle φ with the normal (Figure 7.4.)
The complete intensity is given with the following formula

$$I = I_a k_a + I_i \left( k_d (\mathbf{LN}) + k_s (\mathbf{RV})^n \right) \tag{7.9}$$

In the given model we can also introduce the dependence on the distance between the object and the light source. However, due to simplification it is presumed that the light source in somewhere in infinity, so usually the distance towards the observer is used. Introducing the distances results in the following formula

$$I = I_a k_a + \frac{I_i}{r + k} \left( k_d (\mathbf{LN}) + k_s (\mathbf{RV})^n \right) \tag{7.10}$$

With the given formula the intensity is calculated separately for the *R, G, B* components $I_r$, $I_g$, $I_b$ , but in this assignment it is sufficient to implement only the ambient and diffuse component without the dependence to the distance, and calculate the value for only a single component ($I_r$).

The parameters in the formula for calculating the intensity are selected in a way that the total intensity is within a specified range. For example, in the range from 0 to 1, or 0 to 255. The coefficients $k_s$ i $k_d$ are given in the range from 0 to 1, the dot product of the vector towards the light source and the normal **LN** is from the range between 0 and 1 if both of them are of unit

length. If there are several light sources in the scene, then designing the ranges can be a bit problematic, since the result can exceed the defined ranges. If the result is beyond the maximum value, then the maximum value is associated to that result. The second way is to calculate all the intensity values for the image. When all intensities are known, the maximum intensity can be determined, based on which the other intensities can be calculated linearly or non-linearly. (This procedure is not appropriate for animations)

### *7.5. Shading procedures*

The basic shading methods are constant shading, Gouraud shading and Phong shading.
***Constant shading*** is the simplest and uses the normals of polygons to calculate the illumination intensity of the polygons.The entire polygon has the same intensity which is determined based on the illumination model (expression 7.8. or 7.9.). For triangles this will mean that each vertex will have the same colour associated to it.
***Gouraud shading*** calculates the intensities in the vertices based on the normals in the vertices. Such an intensity is interpolated linearly on the polygon. This means that based on the three different normals calculated by expression 7.4 we will determine the intensities on the vertices.
***Phong shading*** interpolates the calculated normals of the vertices of the polygon, so that for each pixel a normal can be calculated, which will be used to calculate the intensity for each pixel by using the illumination model.

## *7.6. Assignment*

With the assignment there is an example which draws a 3D triangle with colour intensities in the vertices. Instead of the triangle it is required to display a body by using the different shading models and by removing the rear polygons.

1. Read a body as described in Assignment 4. Place the body in the centre of the working space [-1, 1] as described in point 7.2. Perform the view transformation and perspective projection as in Assignment 5, and compare the results to the code which uses OpenGL functions.
2. Determine the rear polygons of the body by using formula 7.1 (Figure 7.1)
3. Show the wireframe form of the body with the rear polygons removed.
4. Define the position of the light source.
5. Based on the polygon normals and the vector **L** towards the light source from the centre of the polygon, calculate the intensities of the polygon by using formula 7.7 (the ambient and diffuse component)
6. Draw the body so that all three polygon vertices have the same intensity (constant shading)
7. Determine the normals in the vertices for the surfaces by using formula 7.4 and normalise them.
8. Based on the normals in the vertices and the vectors towards the light source, determine the intensities in the vertices by using formula 7.7 (the ambient and diffuse component)
9. Display the body so that all three vertices have different intensities which were determined in step 9 (Gouraud shading)

# 8. Fractals – Mandelbrot and Julia fractal sets

## *8.1. The complex plane and the display plane*

A function of the complex variable $f(z_n)$ is observed in the complex plane with axes (u,v). The display plane (x, y) is a plane in which we display the considered complex function. The translation from system (O u v) to system (O' x y) depends on the considered area in the individual systems. Let the observed area in the complex plane be specified with $u_{max}$, $u_{min}$, $v_{max}$ i $v_{min}$. Let the display system be defined with $rez_x$ i $rez_y$ (Figure 8.1).

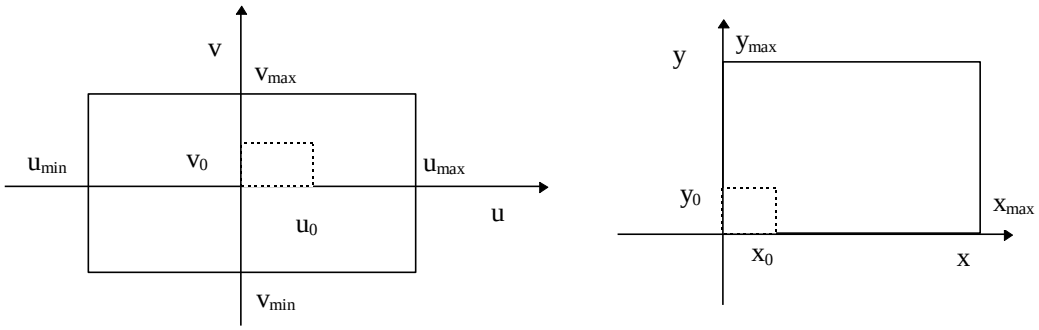



Figure 8.1. The plane of the complex function and the display plane

Since the display system is the screen, the values of *x* and *y* are discrete. The coordinates $u_0$ i $v_0$ of a point in the complex plane which correspond to the values of $x_0$ i $y_0$ are:

$$u_0 = \frac{u_{max} - u_{min}}{x_{max}} x_0 + u_{min},\ v_0 = \frac{v_{max} - v_{min}}{y_{max}} y_0 + v_{min} \qquad (1)$$

With the given expressions we can define the transition from one system to the other.

## *8.2. Mandelbrot and Julia sets*

Let the recursive mapping be defined as:

$$z_{n+1} = f(z_n), \qquad (2)$$

where $f(z_n)$ is for example $z_{n+1} = f(z_n) = z_n^2 + c,\ z, c \in C$, and *c* is a selected point in the complex plane for which we test the convergence of the generated series. For such a defined iterative mapping we can observe if the generated sequence ($z_0$, $z_1$, $z_2$, ..) converges or not. The stopping criterion in the implementation can be different. One example of the criterion with which we can determine whether the sequence converges is by testing the absolute value:

$$|z_n| = \sqrt{u^2 + v^2}, |z_n| < \varepsilon, \quad n > n_0$$

If the iterative mapping $z_{n+1} = f(z_n)$ after *n* iterations does not satisfy the condition $|z| > \varepsilon$ we will determine that the sequence converges, otherwise we will say that the sequence diverges. The „convergence speed“ is defined as the number of iterations which are required so that condition $|z| > \varepsilon$ will be satisfied. The procedure is performed so that for each pixel in the display ($x_0$, $y_0$) the appropriate point in the complex plane is determined, for which the convergence of the

sequence is tested. The area of the complex plane inside of which the iterative mapping generates sequences which converge are called the Mandelbrot set.

For the Julia set it the point $c \in C$ (point in the complex plane) needs to be selected, and $z_0$ is the point of the complex plane for which the convergence of the sequence is tested. If $c \in C$ is selected so that it belongs to the Mandelbrot set, then the Julia set will be connected, otherwise it will not be connected.

## 8.3. Assignment

Procedure for the Mandelbrot set:

1. Enter the threshold value *eps* and the maximum number of iterations *m*.
2. Enter the area of the complex plane which is considered ($u_{min}$, $u_{max}$), ($v_{min}$, $v_{max}$).
3. Enter the resolution of the screen $x_{max}$, $y_{max}$.
4. For each point on the screen $x_0$, $y_0$:
   a) determine $u_0$, $v_0$ (using formula 1).
      a) Set: k = -1, $c_{real} = u_0$, $c_{imag} = v_0$ , $z_0 = 0$.
      b) do:

$$k = k + 1 ,$$

$$z_{n+1} = z_n^{\,2} + c$$

$$r = \sqrt{z_{real}^{\,2} + z_{imag}^{\,2}}$$

      while the conditions $r < eps$ and $k < m$ are satisfied:
5. On the position $x_0$, $y_0$ draw the pixel element in colour k .

Example: *eps*=100, *m*=16, ($u_{min}$, $u_{max}$) = (-1.5, 0.5), ($v_{min}$, $v_{max}$)=(-1, 1)

Procedure for the Julia set:

The procedure is similar to the previous one with the differences being

1. Additionally enter the complex number $c \in C$.
   a) Set: k = -1, $z_{real} = u_0$, $z_{imag} = v_0$.

Example: *eps*=100, *m*=16, ($u_{min}$ $u_{max}$)=(-1 1), ($v_{min}$ $v_{max}$)=(-1.2 1.2), ($c_{real}$ $c_{imag}$)=(0.32 0.043).

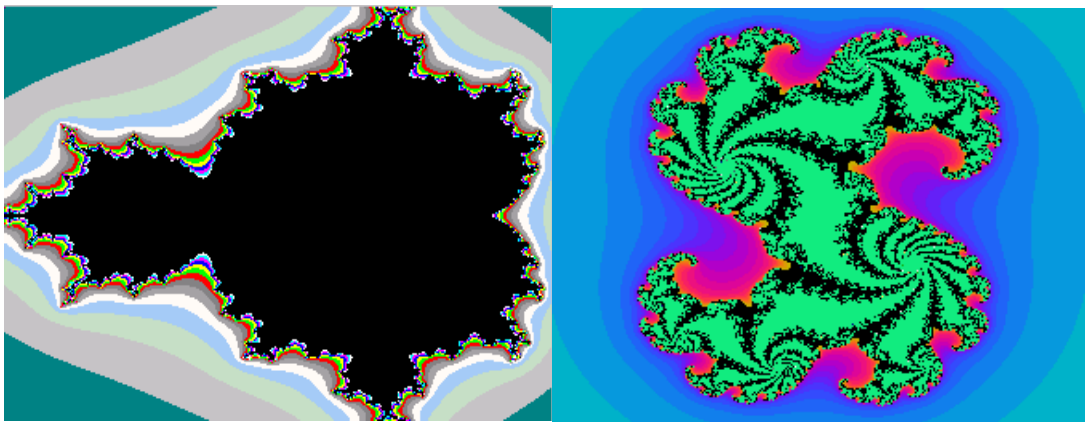


Figure 8.2: The Mandelbrot and Julia sets