

# HOLOMAKERS PROJECT

**Motivating secondary school students towards STEM careers through  
hologram making and innovative virtual image processing practices with  
direct links to current research and laboratory practices**

Erasmus+ KA2 2017-1-PL01-KA201-038420

---

## Output 1

### HOLOMAKERS Technical Reference Guide

---

**Lead Partner: WUT**

**Authors: Artur Sobczyk (WUT)**

**Circulation:** *Public*

**Version:** *01*

**Stage:** *Final*

**Date:** *2018*

## Contributions

Karol Kakarenko, Warsaw University of Technology  
Rene Alimisi, EDUMOTIVA

## Declaration

This report has been prepared in the context of the HOLOMAKERS project. Where other published and unpublished source materials have been used, these have been acknowledged.

## Copyright

© Copyright 2017 - 2019 the HOLOMAKERS Consortium  
All rights reserved.



This document is licensed to the public under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

## Funding Disclaimer

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Chapter 1.	Understanding the waves .....	6
1.1.	Waves in physics.....	6
1.2.	Features describing waves .....	7
1.2.1.	Amplitude .....	7
1.2.2.	Frequency .....	7
1.2.3.	Period .....	8
1.2.4.	Wavelength .....	8
1.2.5.	Phase .....	8
1.2.6.	Other features .....	9
1.3.	Properties of waves.....	9
1.3.1.	Diffraction.....	9
1.3.2.	Interference.....	10
1.3.1.	Coherence .....	11
1.4.	Waves in 3D space.....	12
1.4.1.	Spherical wave and plane wave .....	12
1.4.2.	Huygens–Fresnel principle .....	13
1.5.	Light as a wave .....	14
Chapter 2.	.....	15
2.1.	Holography in simple words.....	15
2.2.	The principle of the hologram formation.....	16
2.3.	Types and properties of holograms .....	19
2.3.1.	Types of holograms .....	19
2.3.1.	Properties of holograms.....	20
2.4.	Basic setups for hologram recording .....	20
2.4.1.	Gabor Hologram .....	21
2.4.1.	Leith-Upatnieks setup .....	21
2.4.1.	Rainbow (Benton) hologram .....	22
2.4.1.	Volume hologram.....	23
2.4.1.	Computer generated holograms.....	24
Chapter 3.	Image processing with the use of Octave environment .....	26
3.1.	Installation of the Octave .....	26
3.2.	Starting the program .....	27
3.3.	Command window .....	28
3.4.	Editor .....	30
3.5.	Basic programming issues in the Octave environment.....	32
3.5.1.	Variables.....	32
3.5.2.	Basic input / output operations .....	33
3.5.3.	Operators .....	34
3.5.4.	Conditional statement.....	35
3.5.5.	"For" loop .....	38
3.5.6.	Drawing charts .....	39
3.6.	Image processing.....	41
3.6.1.	Creating an image.....	41
3.6.2.	Reading / writing image and displaying it on the screen .....	41
3.6.3.	Color conversion.....	42

3.6.4.	Rotation .....	42
3.6.5.	Addition, subtraction, multiplication, division.....	43
3.6.6.	Fourier transform .....	43
3.7.	Algorithm for calculating Computer Generated Holograms (CGH) .....	47



Co-funded by the  
Erasmus+ Programme  
of the European Union

## Chapter 1. Understanding the waves

### 1.1. Waves in physics

In physics, wave is a disturbance that spreads in the medium or space. For example, if you throw a stone into the water, a wave will rise up on the water surface, spreading farther away from where the stone hits. In this case, the impact energy will be converted into oscillations of the medium (water). Another example can be sound wave. In this case, the wave also travels thanks to the oscillations of the medium (air).

We can also call a wave an oscillation that is limited in space. For example moving pendulum makes oscillatory motion returning from time to time to the same point. In this case, the wave will be a width of swing of the pendulum that changes over time. Fig. 1 illustrates the change in the pendulum's swing over time. The pendulum deflects in the range  $[-A, A]$ , so we can say that  $A$  is the amplitude of the swing.

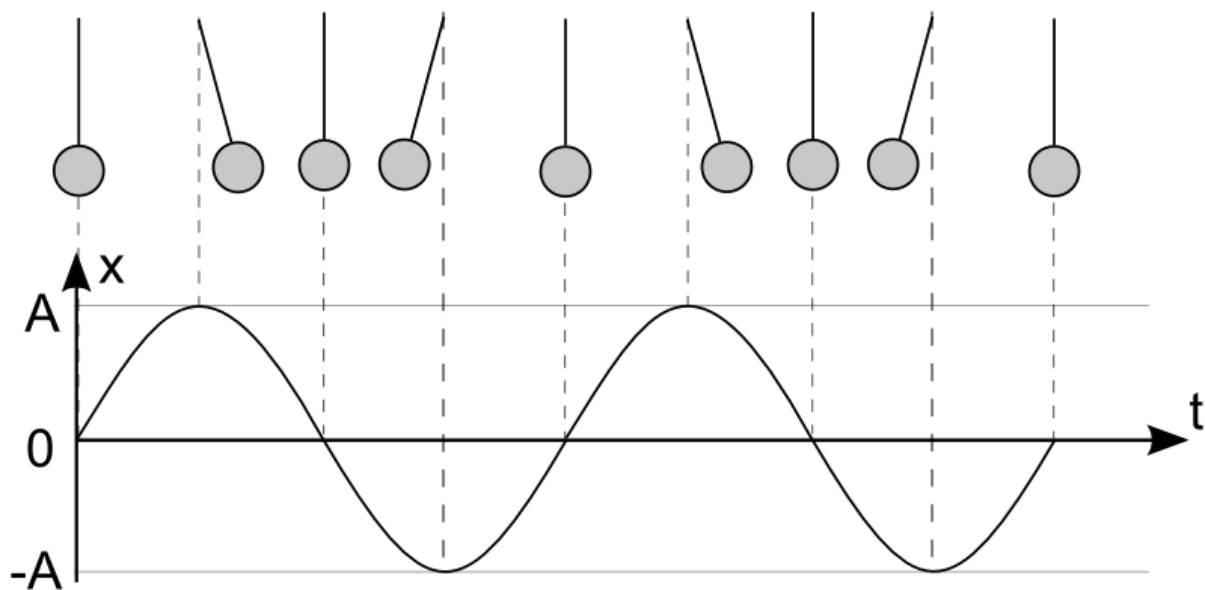


Fig. 1 Displacement of the pendulum at different times

Very often oscillations can be described using the sine function. This takes place in the pendulum already mentioned, but also in the case of a weight suspended on a spring, or in the case of a musical instrument string. This is the basic type of oscillation. This type of oscillation is called **harmonic** oscillations. As we will later learn, every other, more complicated type of oscillation consists of harmonic oscillations. In conclusion, the harmonic wave is one that can be described using the sine function. In the next chapter we will learn the basic properties used to describe the waves.

## 1.2. Features describing waves

### 1.2.1. Amplitude

The amplitude of the wave is the distance of the greatest deflection from the equilibrium position. In the case of a pendulum, the amplitude of the oscillation is its greatest deflection (see Fig. 2). The maximum wave value is called a crest and the minimum value is the trough.

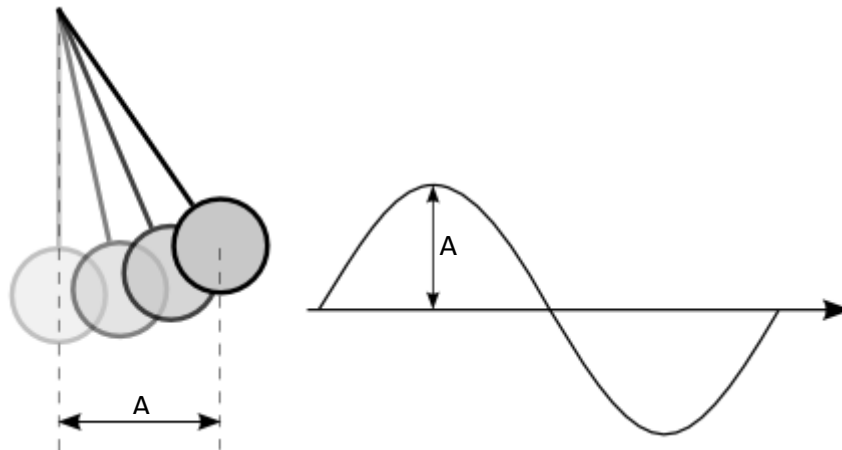


Fig. 2 Amplitude of the pendulum oscillations

### 1.2.2. Frequency

Frequency tells us how many oscillations have been made per unit of time (within one second). The unit of frequency is Hz (Hertz). For example, 10Hz is 10 oscillations per second, 15.5Hz is 15.5 oscillations per second. Fig. 3 presents two waves with different frequencies. The red line is one full oscillation. The frequency of the first wave is 3 Hz (3 oscillations per second) and the second one is 6 Hz (6 oscillations per second).

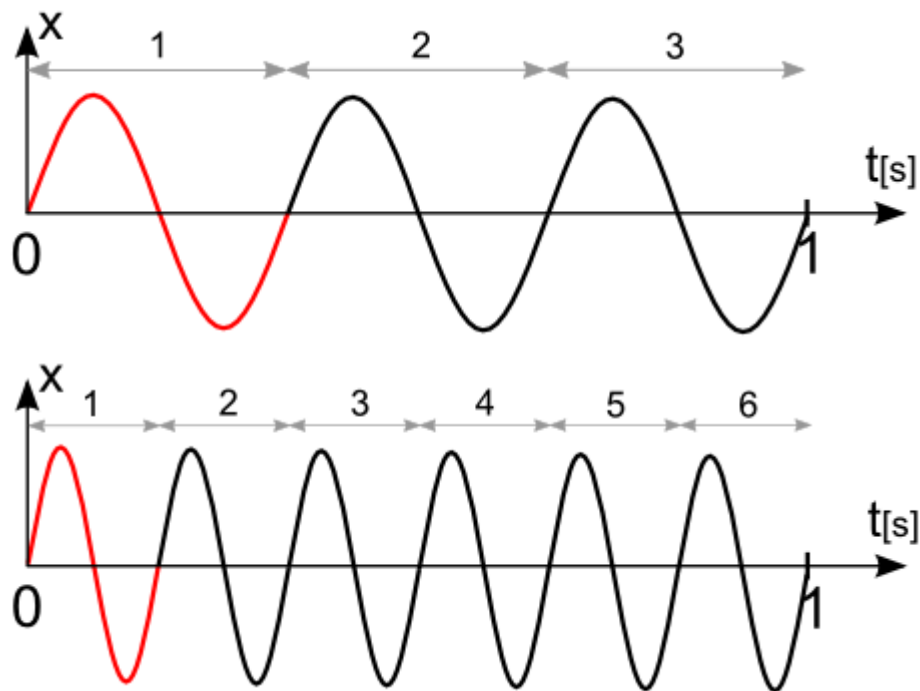


Fig. 3 Frequency of waves

### 1.2.3. Period

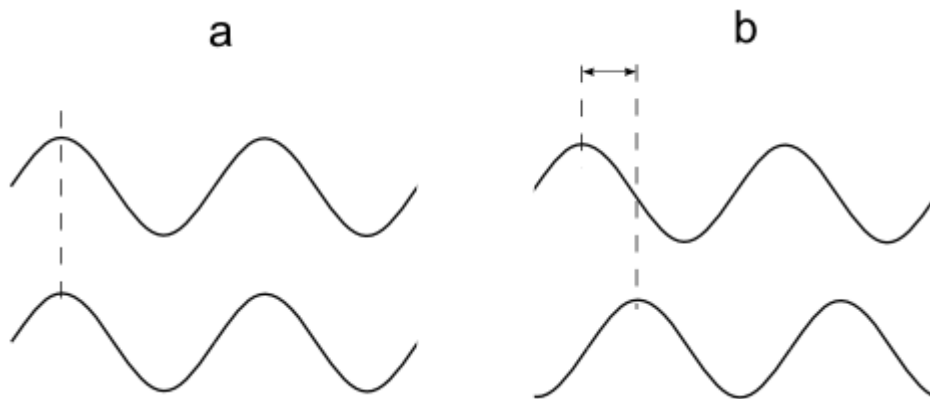
The wave period is directly related to its frequency. It is the time (measured in seconds) at which one full oscillation will be performed (marked in red in Fig. 3). The wave period ( $T$ ) is related to its frequency ( $f$ ) with dependency  $T = \frac{1}{f}$ . Thus, the period of the first wave in Fig. 3 is  $\frac{1}{3}s$  while period of the second wave is equal to  $\frac{1}{6}s$ .

### 1.2.4. Wavelength

If the wave propagates in space, one can determine its wavelength. This is the distance that wave will propagate during one period. Wavelength ( $\lambda$ ) is given an equation  $\lambda = vT$ , where  $v$  is the speed of wave propagation and  $T$  is a wave period.

### 1.2.5. Phase

The phase determines in which part of the wave period the given wave point is located. In practice, however, an important element is not so much the single-wave phase as the **phase shift** between waves. In other words, it is simply a shift from one wave to the other. Fig. 4a shows a situation where there is no phase shift, while Fig. 4b shows waves with a certain phase shift. The phase shift can be measured between any corresponding wave points. For example, in Fig. 4, for convenience, the phase shift is shown as the distance between the maximum of the waves.



**Fig. 4 Phase shift**  
a) no phase shift between waves  
b) waves having a phase shift

### 1.2.6. Other features

In addition to the features mentioned above, we can also distinguish group speed and phase speed of the wave. The waves can be longitudinal or transverse. In the case of transverse waves, we can also talk about polarization. These features are not necessary to understand holographic issues, so they will be omitted in this reference guide.

## 1.3. Properties of waves

Waves propagating in space (eg a wave on the water, sound wave, electromagnetic wave) show certain properties such as reflection, refraction, diffraction or interference.

### 1.3.1. Diffraction

The diffraction refers to changing the direction of the wave propagation as a result of encountering an obstacle or a slit. This effect is well visible when the wavelength is comparable to the size of the slit. Fig. 5 presents the situation in which the wave encounters a slit. Before the slit, the wave moves in a direction perpendicular to the slit (marked with red arrows). There are also areas of total wave extinction (marked with a blue line). Along these lines the amplitude of the wave is 0.

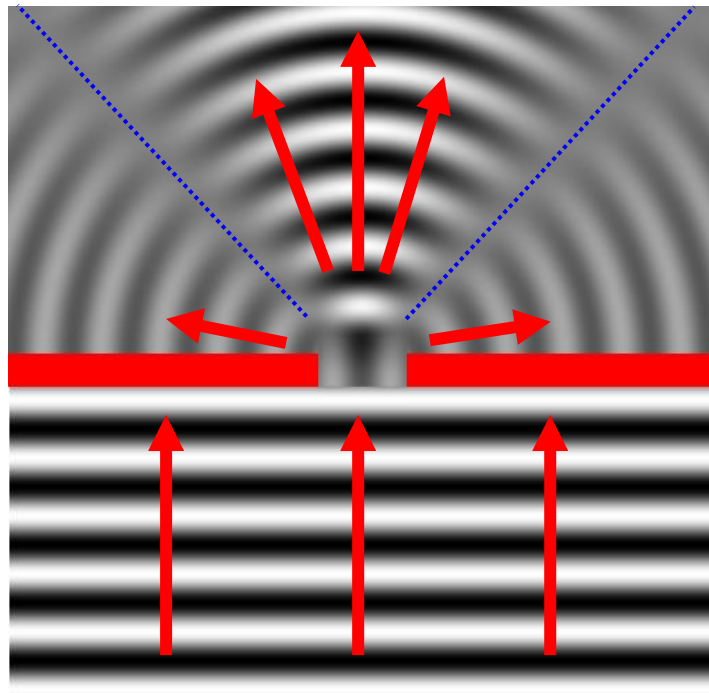


Fig. 5 Diffraction on a single slot

### 1.3.2. Interference

Wave interference is the phenomenon on which holography is based. Interference is overlapping (adding up) of waves. Propagating waves interact with each other, they can strengthen or weaken. Fig. 6 shows wave amplification and extinction. If the waves are "in phase" (the phase difference is 0) then the greatest possible wave amplification occurs. If, on the other hand, the waves are "out of phase" by half of the period (the phase difference is half way through the wave period), the waves are completely turned off. When the waves strengthen, it is constructive interference. When the waves weaken it is destructive interference.

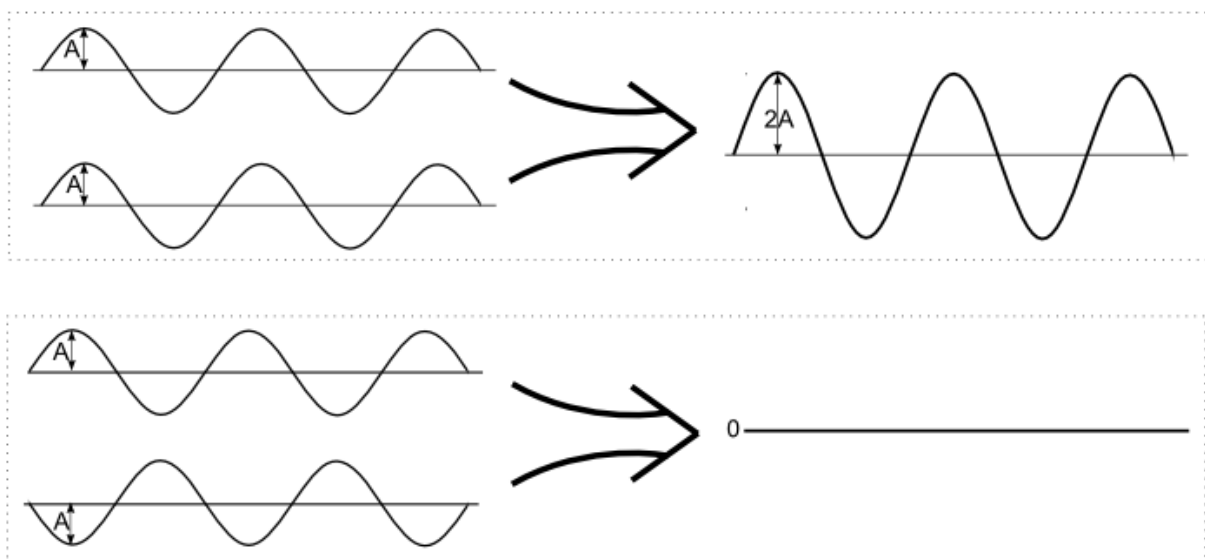
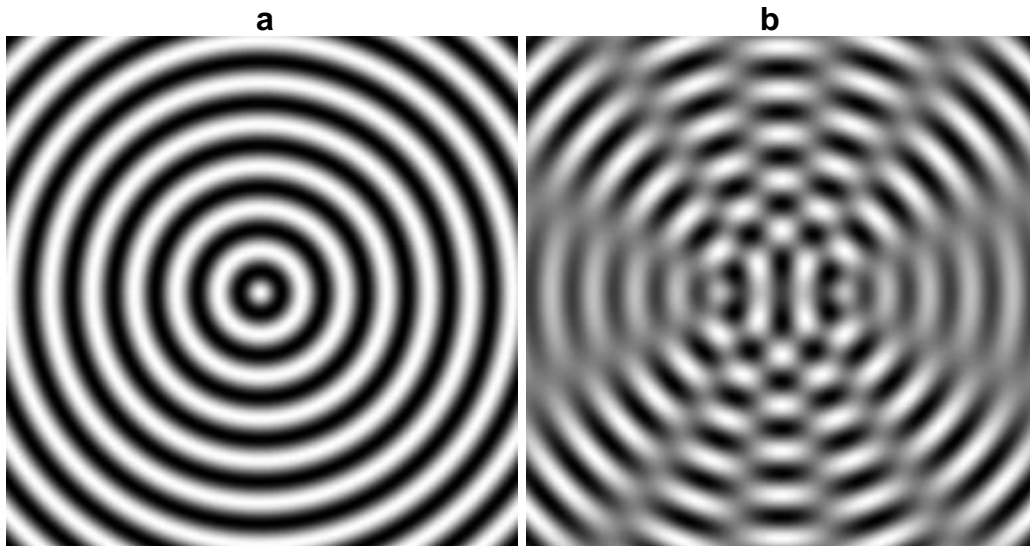


Fig. 6 Constructive and destructive interference

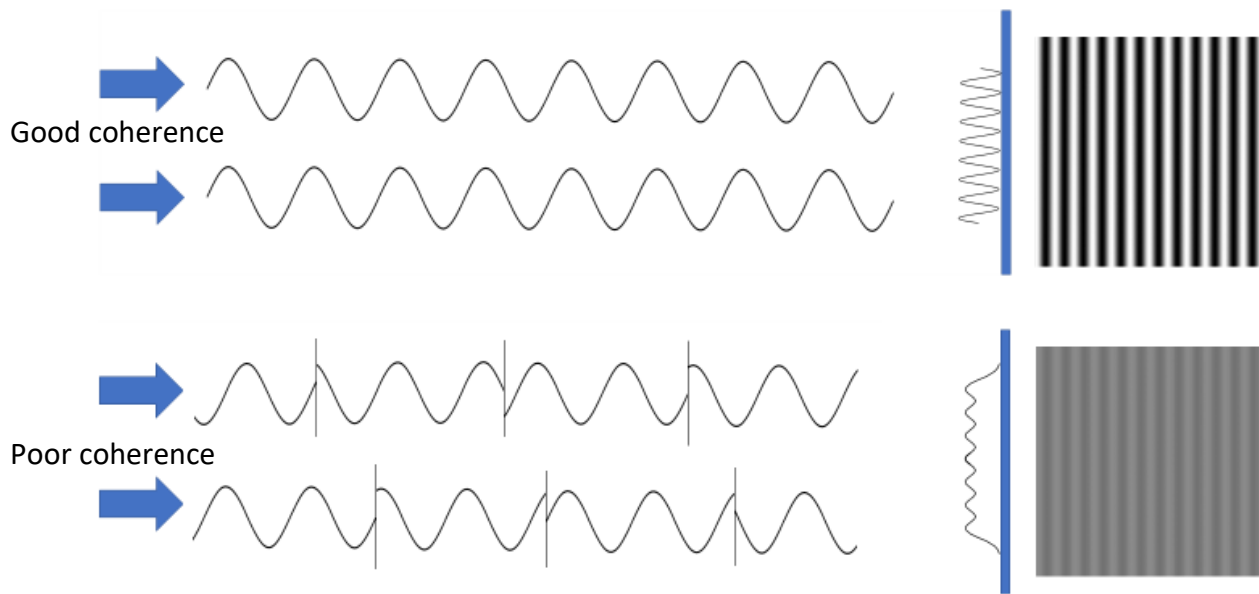
Interference can be illustrated by the example of waves on the water. If we stimulate the surface of water at one point with a constant frequency, we get a wave that propagates radially. This is shown on Fig. 7a. Black areas represent the troughs of the wave and white represent the crests. Fig. 7b presents two waves next to each other and interacting with each other. Gray areas are the extinction of the wave areas that are not blurred are the wave amplification.



**Fig. 7** An example of waves on the water

### 1.3.1. Coherence

Coherence of waves is necessary to obtain constant in time interference pattern. Two waves are coherent if they have a constant phase difference. On Fig. 8 it is shown how in a simplified way one can imagine the difference between coherent and incoherent waves. The upper part of the figure shows highly coherent waves. As a result of overlapping of these waves, formed interference fringes are constant in time. We see, therefore, an interference pattern with good contrast (fringes are clearly visible). In the case where the waves have a low degree of coherence, the resulting fringes at any moment of time are somehow shifted from each other. Our eyes see this as a blurred image of the interference pattern (with reduced contrast). The lower degree of coherence, the interference image is more blurred. In the case of fully incoherent waves, we will not observe interference pattern. Most of the light in nature are incoherent waves. However, a good source of coherent waves is a laser.



**Fig. 8 Good coherence is necessary for obtaining a high contrast interference pattern**

## 1.4. Waves in 3D space

### 1.4.1. Spherical wave and plane wave

The wave in 3D space can be represented as a surfaces where phase is constant. The most common example is a spherical wave or so-called plane wave (Fig. 9). In the first case (Fig. 9a), the surfaces of the constant phase (that is, the surfaces in which the wave has the same value, eg it reaches the maximum value) have the shape of spheres (hence the name of the spherical wave). In the second case (Fig. 9b), the surfaces of the constant phase have the shape of planes (hence the name of the plane wave).

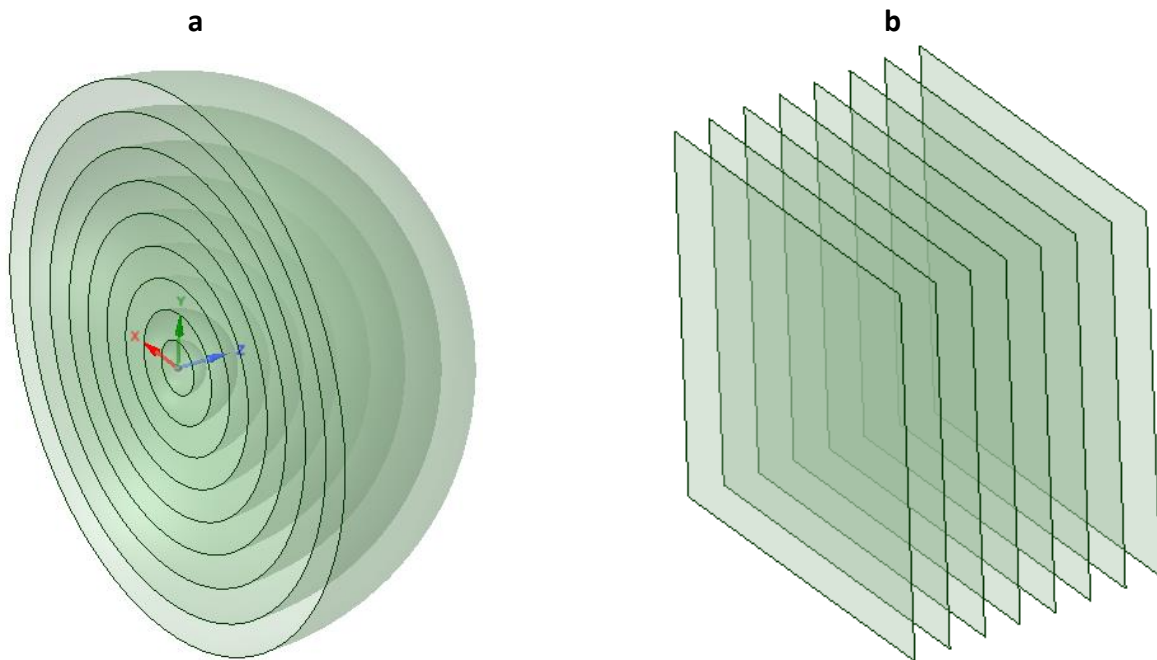


Fig. 9 Spherical wave (a) and plane wave (b)

### 1.4.2. Huygens–Fresnel principle

Consider the case when the wave reaches an object (Fig. 10). It can be assumed that every point the wave reaches becomes the source of a new spherical wave. The sum of all these spherical waves (i.e. interference) determines the shape of the new wave. In Fig. 10 the incident wave is marked in black. The object to which the wave arrives can be represented as a set of infinitely many points shown in the figure as green dots. Each of these points is the source of the secondary spherical wave. After adding up all the secondary waves, we get a new wave (red dotted line in the figure) which then propagates in space. This is the principle of Huygens-Fresnel, which reads as follows:

*Every point to which a wave reaches becomes a source of a spherical wave; the sum of these secondary waves determines the form of the wave at any subsequent time.*

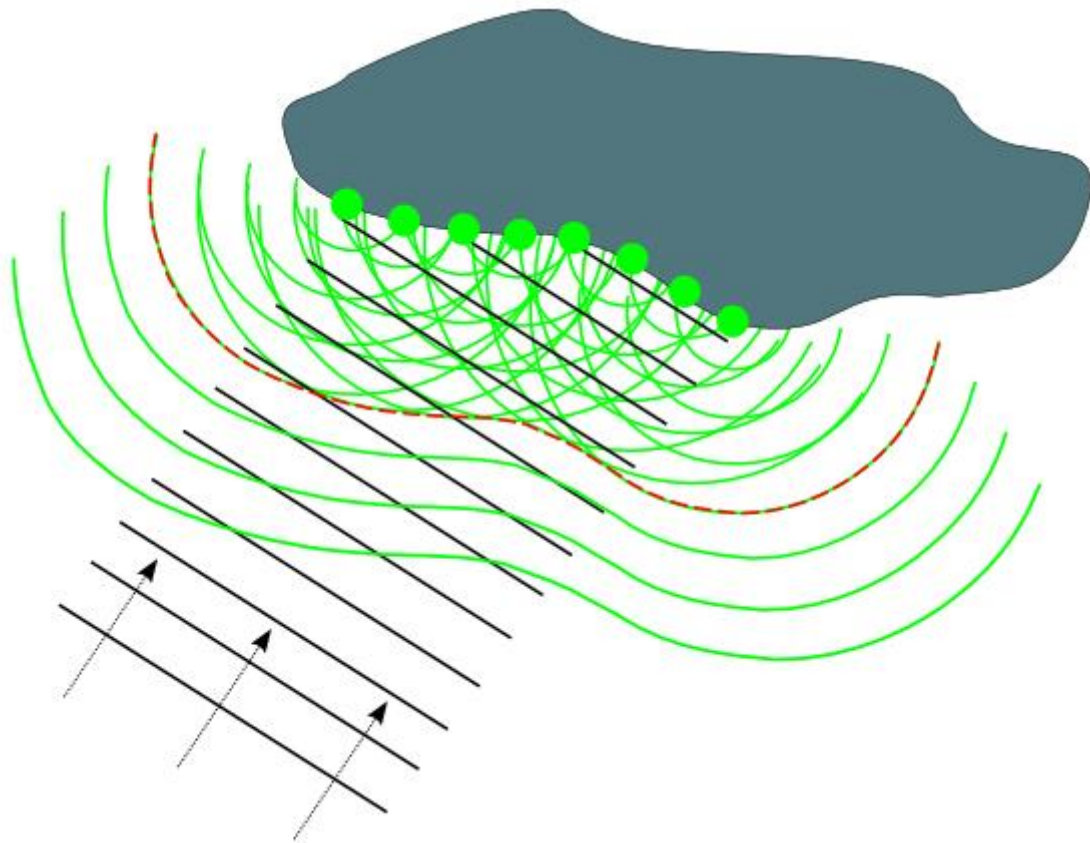


Fig. 10

## 1.5. Light as a wave

Light is also a wave. If so, then what really waves there? What is waving is the electric field and the magnetic field. That is why it is said that light is an electromagnetic wave (EM). Such a wave can be imagined as increasing and decreasing the electric and magnetic fields that are propagating in space (Fig. 11). The electric field is marked in red and the magnetic field in blue.

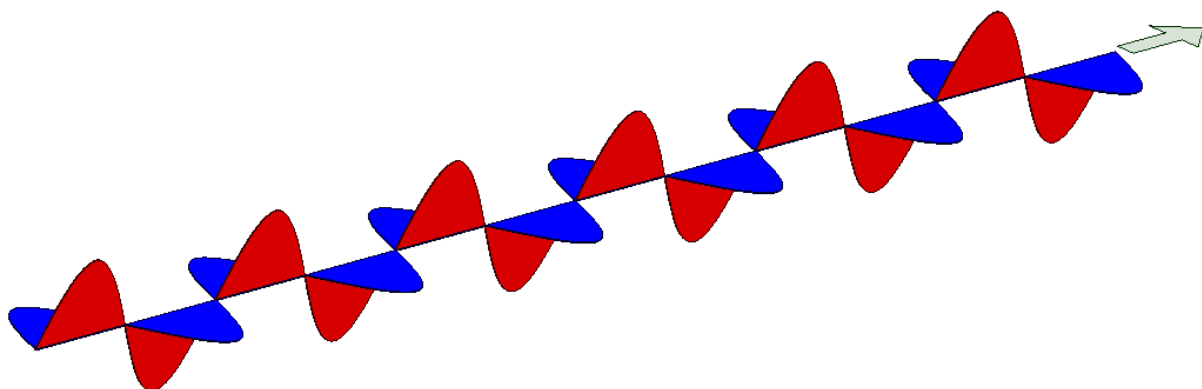


Fig. 11 Electromagnetic wave

In nature, there are many types of electromagnetic radiation. They are classified due to the speed of electromagnetic field changes in space (i.e. due to the frequency of waves). EM waves with the highest frequency are gamma rays, then we have X-rays, ultraviolet, visible light, infrared, microwaves and radio waves as the lowest EM radiation frequency (Fig. 12).

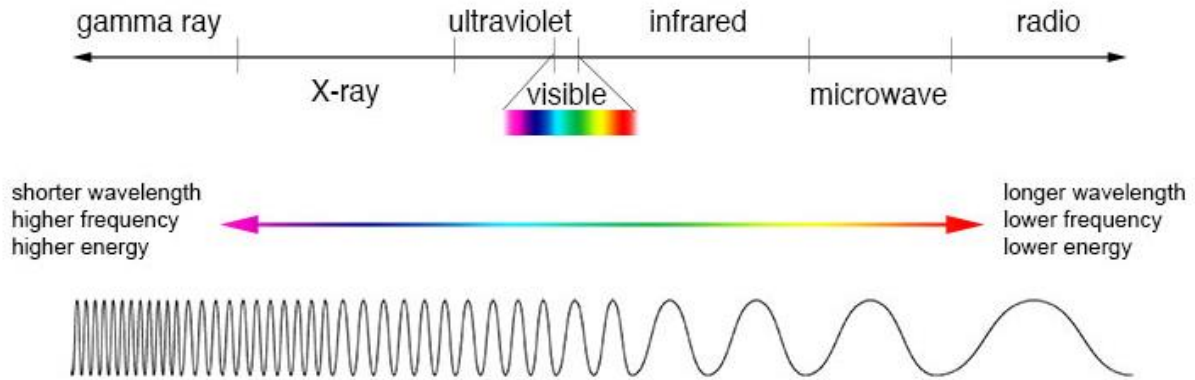


Fig. 12 The spectrum of electromagnetic waves. Source: NASA's Imagine the Universe (<https://imagine.gsfc.nasa.gov/science/toolbox/emspectrum1.html>)

## Chapter 2.

### 2.1. Holography in simple words

A hologram is a plate or a glass plate on which surface or in its volume an interference pattern is recorded. The light passing through or reflected from the hologram creates an image identical to the light reflected from the real object (see Fig. 13).

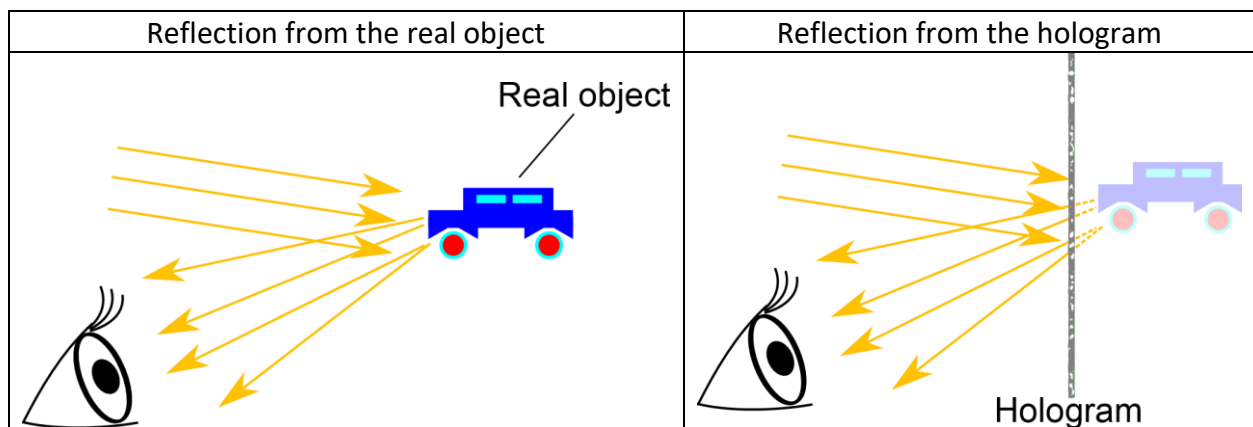
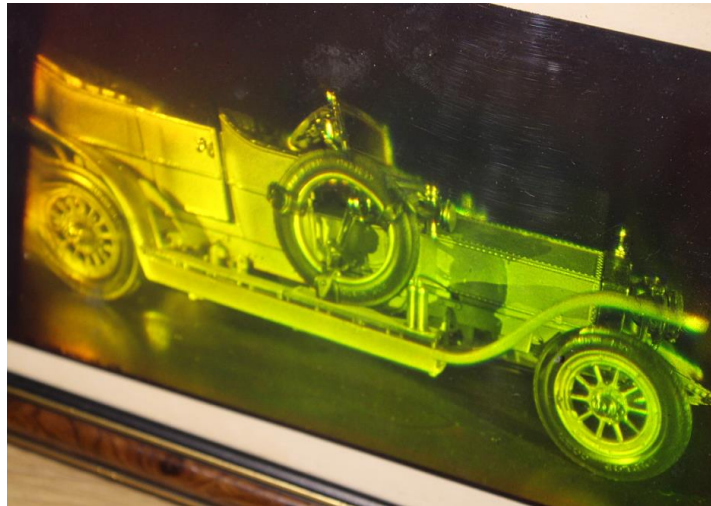
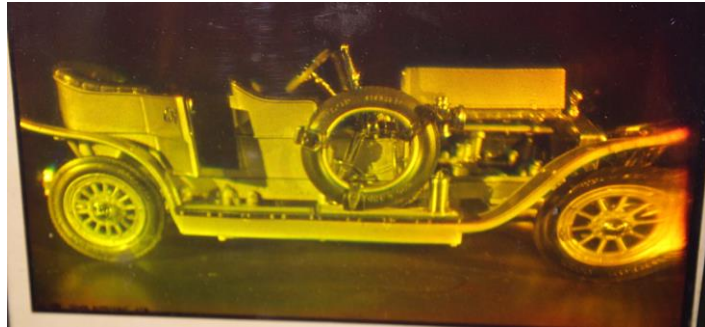


Fig. 13 The light reflected from the hologram behaves identically as if it came from a real object

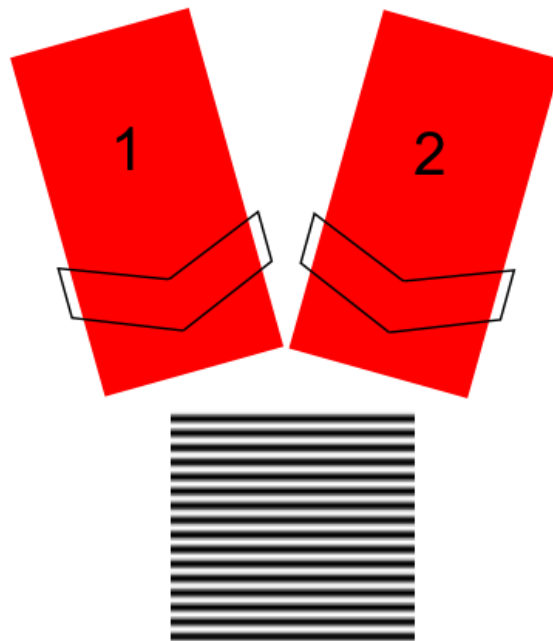
A characteristic feature of the hologram is that the observed object is fully three-dimensional. This means that by looking at the hologram at a certain angle, we can see details that are invisible when looking at the same hologram from a different angle (see Fig. 14). This type of property does not have ordinary photos; the photos look the same regardless of the angle of observation.



**Fig. 14** The image visible in the hologram changes with the angle of observation

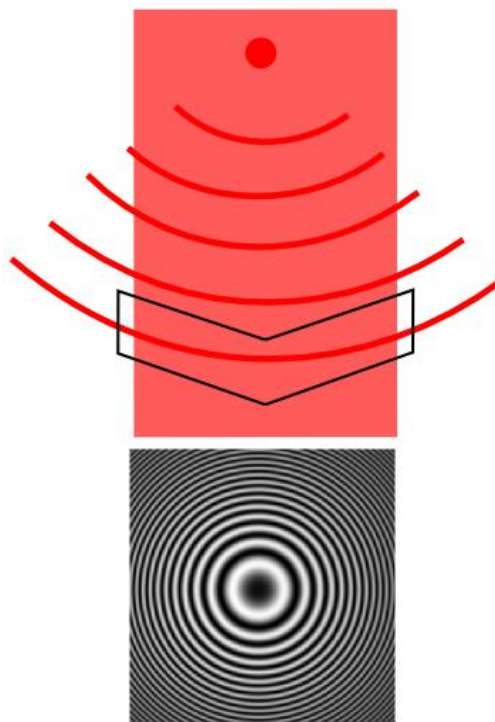
## **2.2. The principle of the hologram formation**

The physical phenomenon that creates a hologram is interference. We already know that two coherent waves interfere with each other to create interference fringes. As a simple example, Fig. 15 shows two plane waves that interfere with each other to form straight fringes.



**Fig. 15 Interference of two plane waves**

The interference of a plane wave with a point source is depicted on Fig. 16. Interference fringes have the shape of circles. We can say that such an interference pattern contains information about a point in the space. If we manage to record this distribution, for example on a holographic film, then it is possible to reconstruct decoded point by illuminating the pattern with a plane wave.



**Fig. 16 The interference of a plane wave with a point source**

Any illuminated object can be treated as a set of many secondary point sources (Huygens-Fresnel principle). Interference pattern resulting from the interference of many point sources with a plane wave is generally complicated (see Fig. 17). We can say that the object is encoded as an interference pattern.

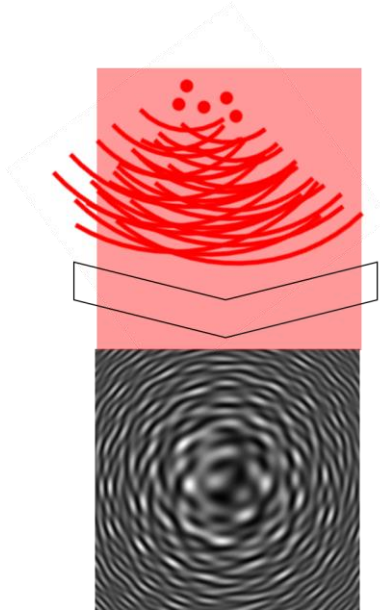
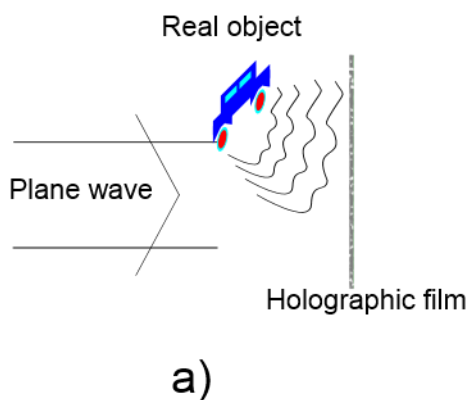


Fig. 17 The interference of a plane wave with many point sources

Such pattern is a hologram. This hologram can be reconstructed by illuminating it with a plane wave. Fig. 18 illustrates an example of the recording and reconstruction of a hologram. In this case, during the reconstruction of the the hologram, we obtain a real image (visible on the screen) and a virtual image (visible when we look directly at the illuminated hologram).

## Hologram recording



## Hologram reconstruction

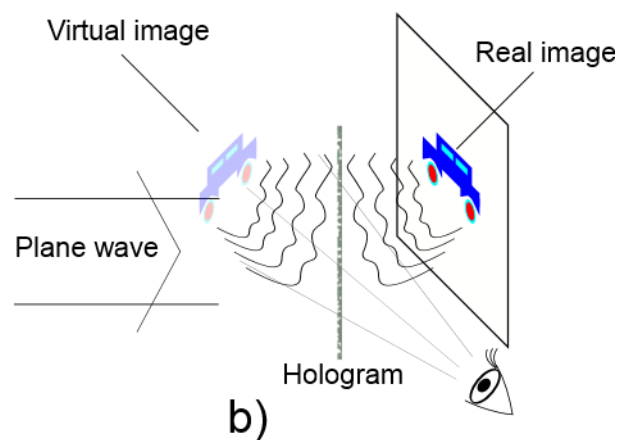


Fig. 18 Recording a hologram (a) and its reconstruction (b)

## 2.3. Types and properties of holograms

### 2.3.1. Types of holograms

Holograms can be classified in different ways based on their properties. One of the divisions can be how the light passes through the hologram. We say that the hologram can be amplitude or phase modulated. The amplitude modulation means that the interference pattern is recorded in the form of more or less transparent areas (Fig. 19a). The phase modulated hologram is fully transparent and the interference pattern is recorded in the form of areas with different refractive index (Fig. 19b).

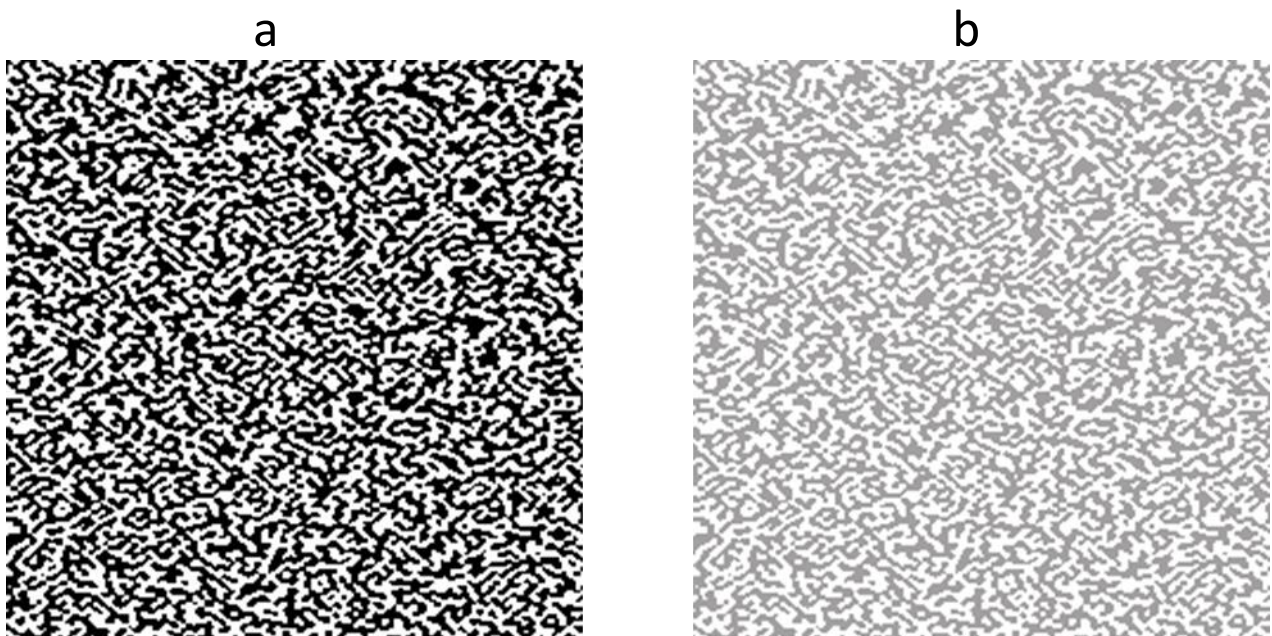


Fig. 19 Amplitude (a) and phase (b) modulated hologram

Holograms can also be divided into so-called thin and thick holograms. Holograms thin are those that behave as if the interference pattern was recorded only on the surface of the holographic plate. Thick holograms are those in which the interference pattern is recorded in the entire volume of the hologram.



Fig. 20 Thin (a) and thick (b) hologram

The next way of describing holograms is transmission and reflection holograms. The transmission hologram can be observed by transmitting light through a hologram while the reflection hologram is observed by reflecting the light from the hologram.

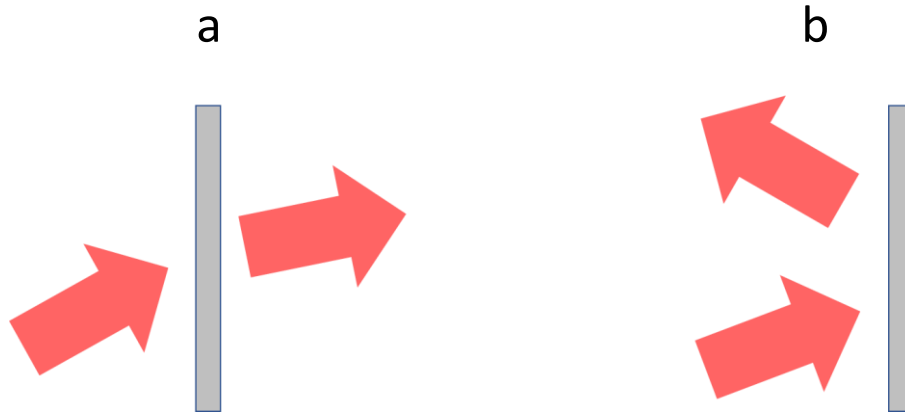


Fig. 21 Transmission (a) and reflection (b) hologram

### 2.3.1. Properties of holograms

Holography is a photographic method - information about the image is recorded on the holographic plate. Nevertheless, the differences between the holograms and the regular photos are essential:

- In the case of holography, the film must have a much higher resolution (measured in lines per millimeter).
- Photographic negative contains information only about the intensity of light, while in the hologram (in the form of an interference pattern), information about both light intensity and phase (ie the distance of the object's points from the hologram) is encoded.
- Each fragment of the photographic negative contains information about a different part of the image, while in the case of a hologram, each piece of it contains information about the whole picture. This means that if we cut the hologram in two parts, each of them will reconstruct the whole picture. Although there will be a difference in the brightness of the image and the angle range at which we can observe the hologram will decrease.

## 2.4. Basic setups for hologram recording

### 2.4.1. Gabor Hologram

Historically, the first setup for recording holograms was proposed by Denis Gabor in 1948. Lasers were not yet invented at that time. This setup uses a point light source, thanks to which it is partially coherent and possible to record a hologram. The object is a translucent film (e.g. a sign or a simple picture). Some of the light passes straight through the film and some is scattered. Both of these beams interfere with each other by recording on the film as interference pattern (see Fig. 22).

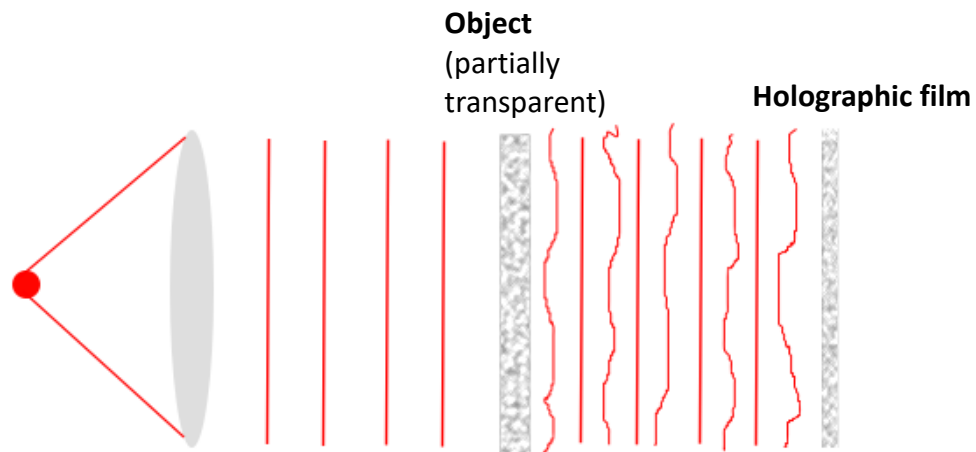


Fig. 22 Gabor hologram

### 2.4.1. Leith-Upatnieks setup

In this setup, one can record so-called Fresnel hologram, which can then be reconstructed with the laser light. The laser beam is divided in two (so-called reference beam and object beam). The light scattered from the object interferes with the reference beam and records as an interference pattern on the holographic film (Fig. 23). An example of a Fresnel hologram is shown in Fig. 24.

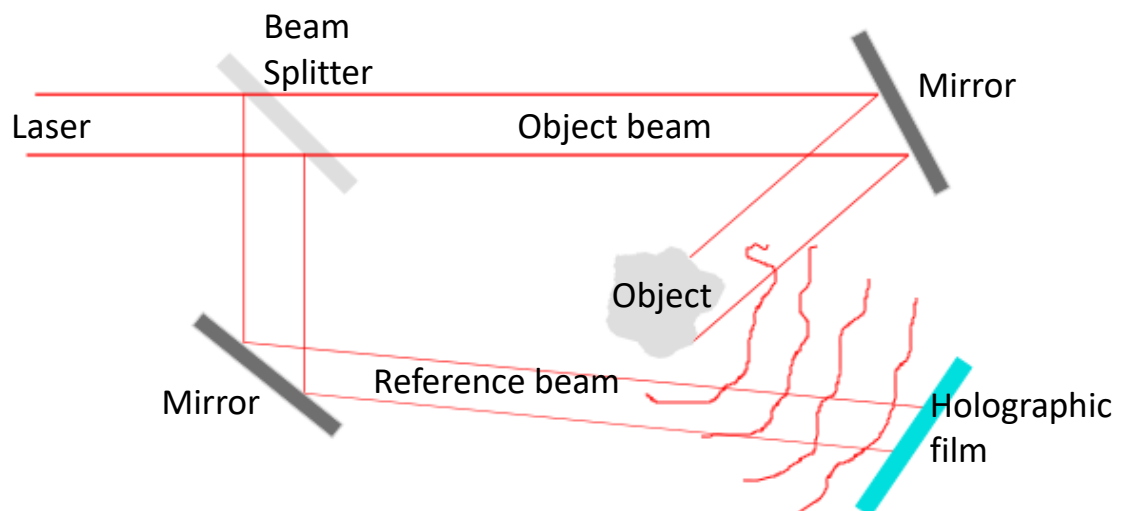


Fig. 23 Fresnel hologram recorded in Leith-Upatnieks setup

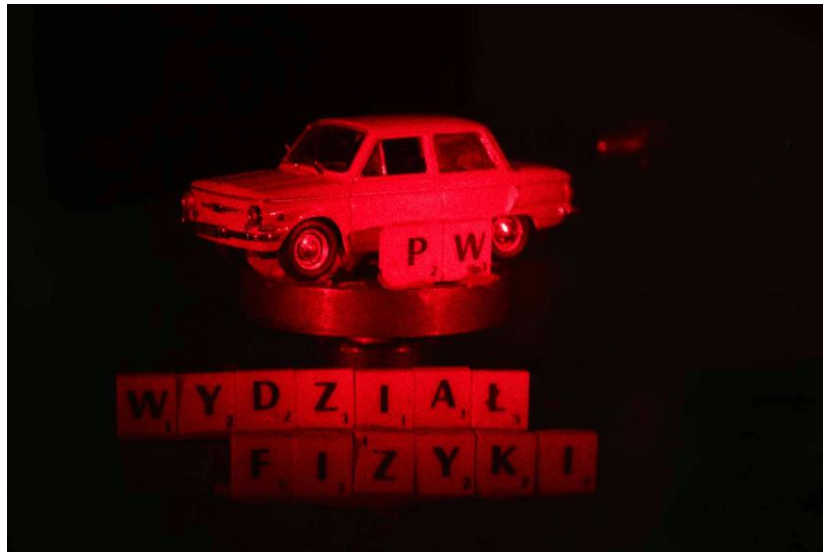


Fig. 24 An example of Fresnel hologram

### 2.4.1. Rainbow (Benton) hologram

In this setup (Fig. 25), a Fresnel hologram (eg previously recorded in the Leith-Upatnieks system) is used as the object. Obtained hologram has the advantage that it can be viewed in white light. By turning the hologram slightly in one direction, the 3D effect is visible (ie so-called parallax) while in the other direction one can observe the change of colors. This is illustrated in Fig. 26.

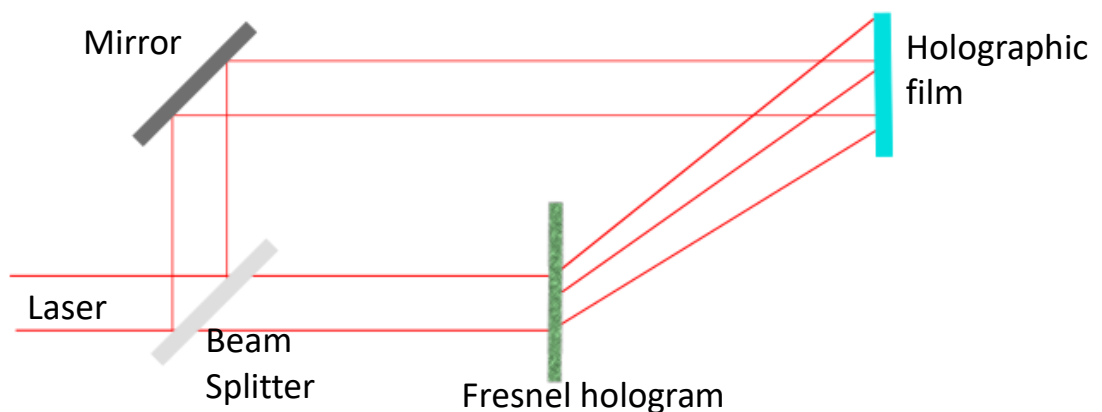


Fig. 25 Recording a rainbow hologram

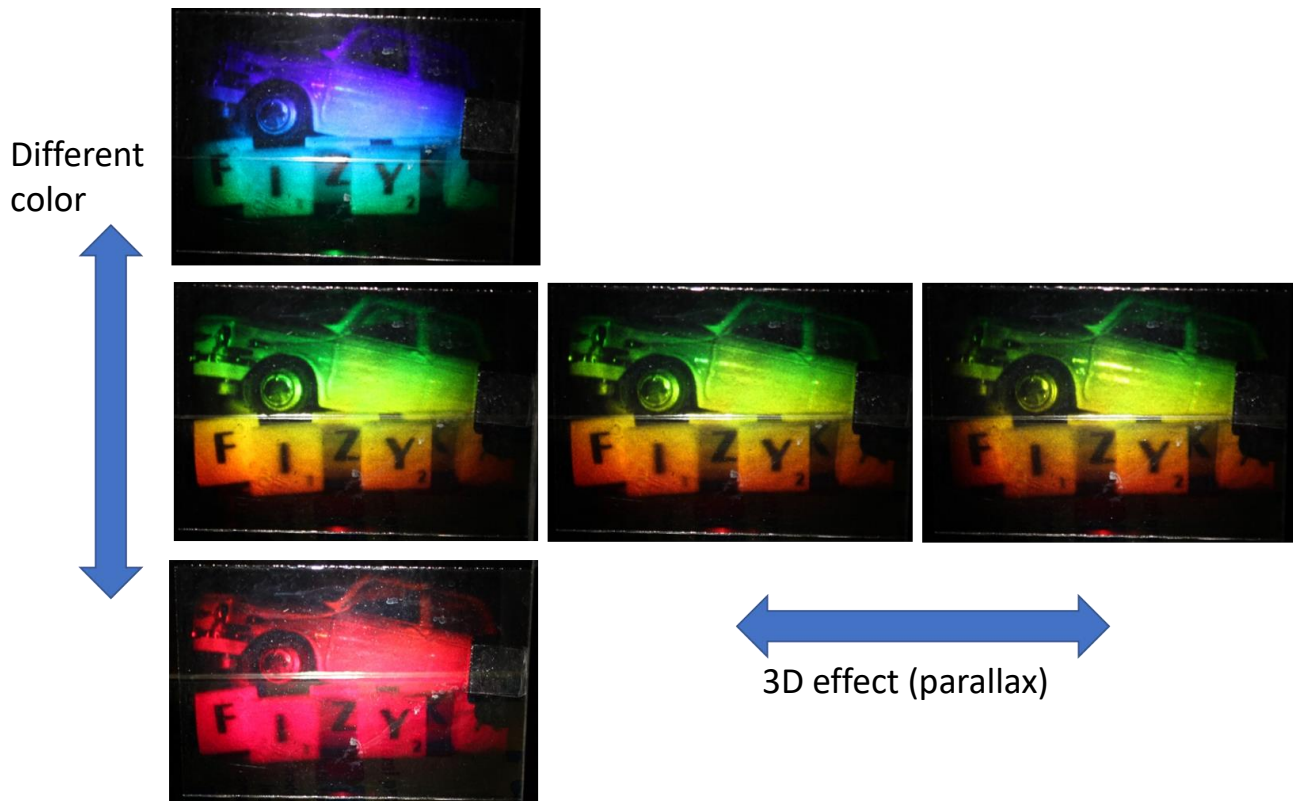


Fig. 26 An example of rainbow hologram

#### 2.4.1. Volume hologram

This hologram can be recorded in the setup shown in Fig. 27. The volume hologram can be viewed in transmission mode in white light. An example of a volume hologram is shown in Fig. 28.

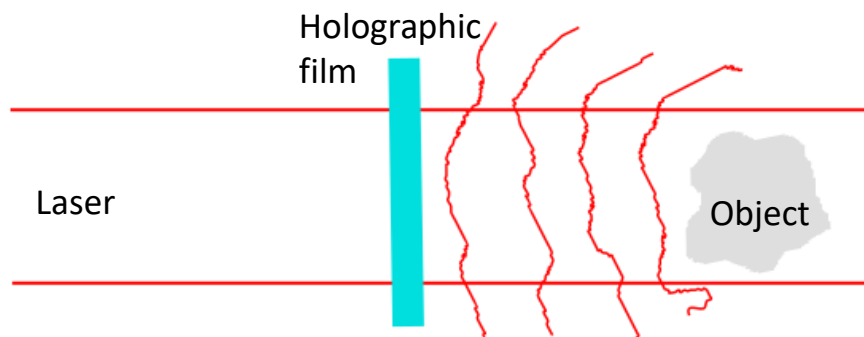


Fig. 27 Setup for recording volume hologram



**Fig. 28 An example of volume hologram**

### **2.4.1. Computer generated holograms**

No physical object is needed to make computer generated hologram (CGH). In the simplest case, the role of the object plays a graphic file with an sign or two-dimensional shape. On the basis of such a file, an interference pattern (also a graphic file) is computed using the computer. An iterative algorithm is used for calculations (Gerchberg's Saxton algorithm is the most popular one). The calculated interference pattern can then be made in the form of an optical element (slide). If we light the slides with a laser beam (eg with a laser pointer) then the passing light will form a previously designed shape. An example of such a hologram can be popular ends for laser pointers which generate desingned shape. Making of a high quality hologram requires sophisticated equipment. Nevertheless, in home conditions, a CGH suitable for educational purposes can be made. For this purpose, the calculated interference pattern should be printed, for example, on A4 paper. Then a photo of the printed pattern should be taken with an analog camera. After that, all you have to do is develop the film, thus receiving a computer generated hologram. The process of obtaining such holograms is presented in Fig. 29.

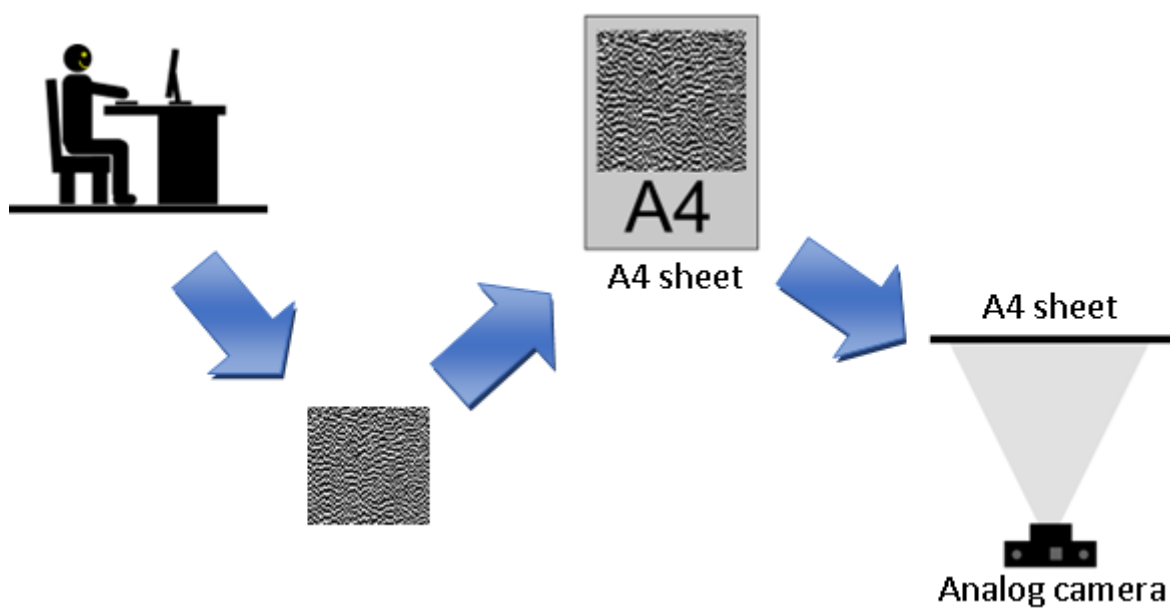


Fig. 29 The process of obtaining CGH in home conditions

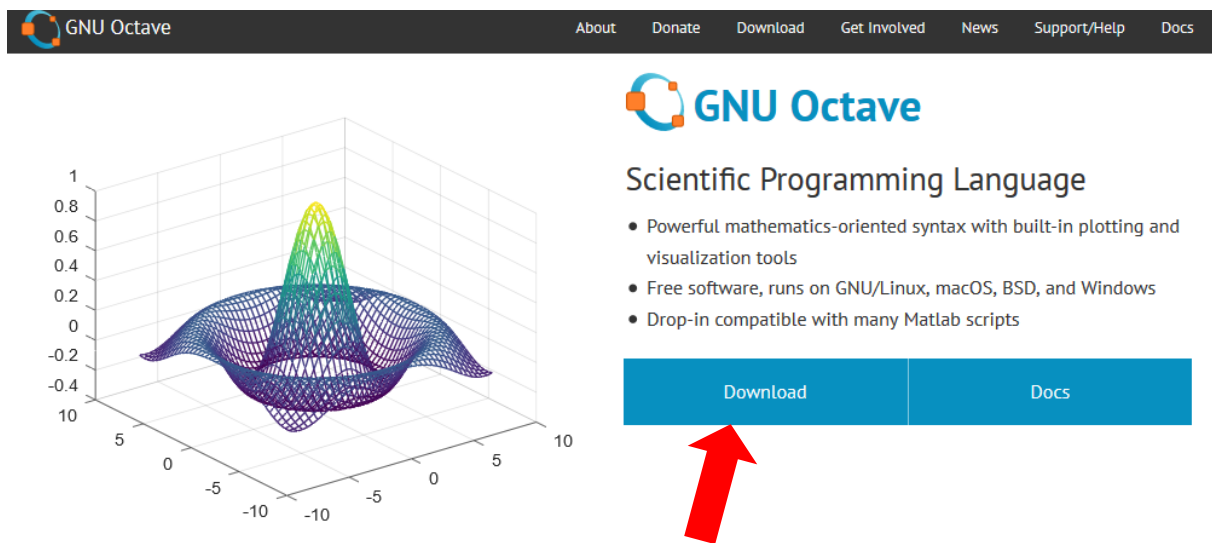
## Chapter 3. Image processing with the use of Octave environment

### 3.1. Installation of the Octave

The Octave program should be downloaded from the website:

<https://www.gnu.org/software/octave/>

To do this, click the download button in accordance with Fig. 30:



### Syntax Examples

The Octave syntax is largely compatible with Matlab. The Octave interpreter can be run in GUI mode, as a console, or invoked as part of a shell script. More Octave examples can be found in [the wiki](#).

Solve systems of equations with linear algebra operations on **vectors** and **matrices**.

```
b = [4; 9; 2] # Column vector
A = [ 3 4 5;
      1 3 1;
```

Fig. 30 Downloading Octave

Then go to the Windows tab and download the file [octave-4.2.1-w32-installer.exe](#) or [octave-4.2.1-w64-installer.exe](#) depending on whether it should be a 32 bit or 64 bit version (Fig. 31).

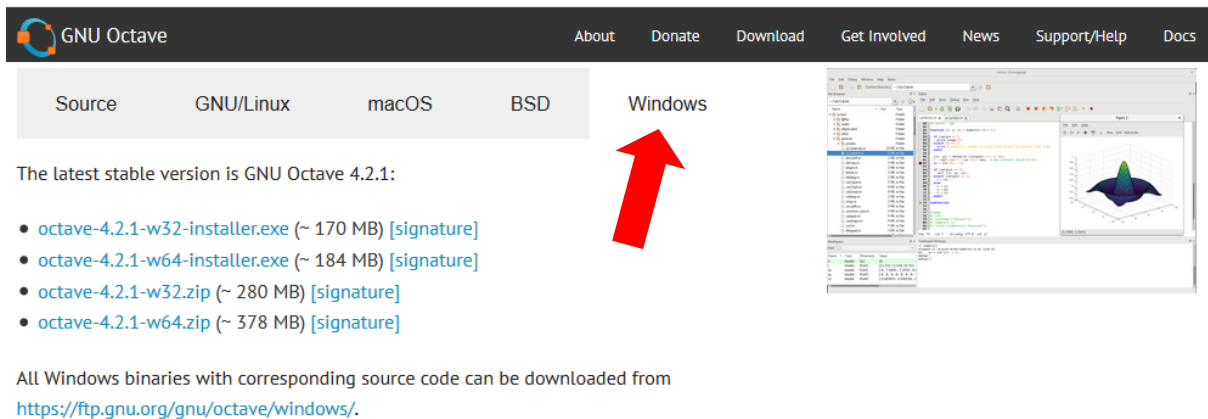


Fig. 31 Downloading Octave

## 3.2. Starting the program

If you work with Windows, click on the Start menu and start writing "Octave" in the search field. Then open the "Octave (GUI)" file. The main view will appear, in which there are several windows (Fig. 32). The most important of them is the area for entering commands (marked with the number 1), then a window allowing to choose a folder in which we will work (marked with the number 2) and lastly a file browser in the current folder (number 3).

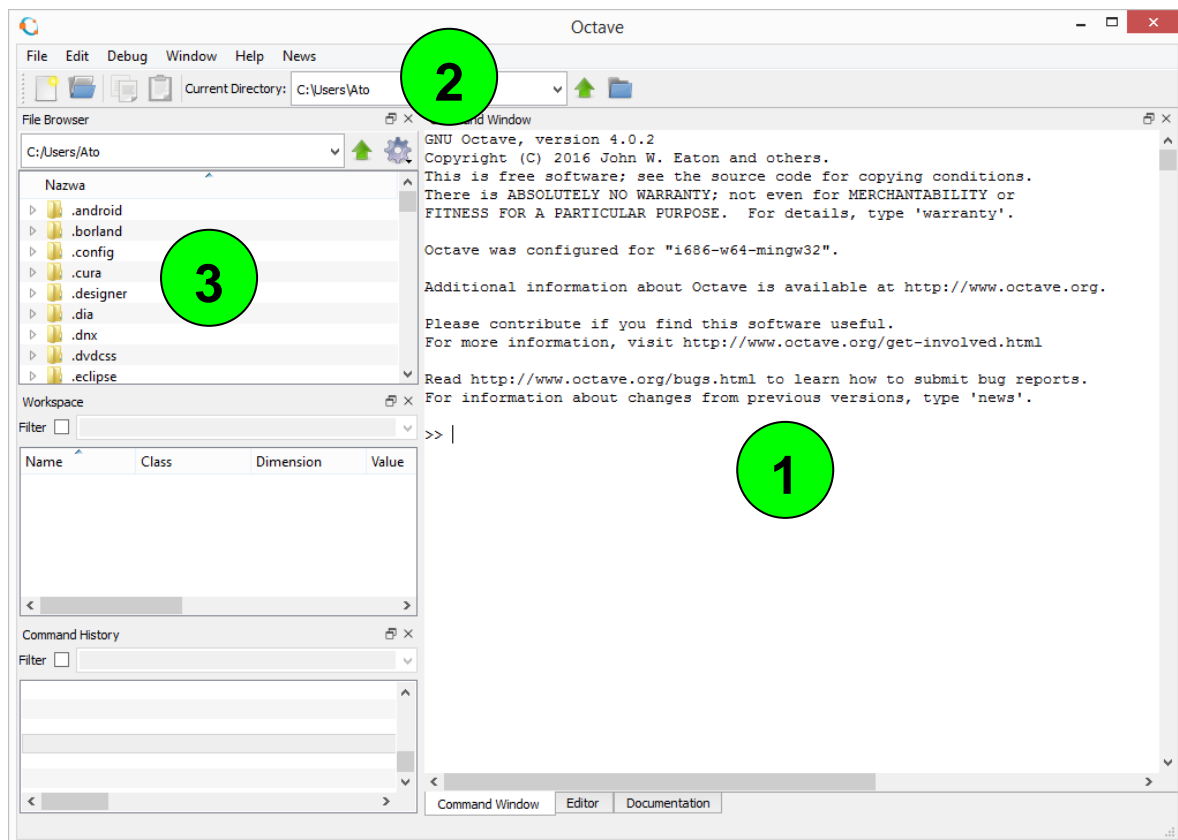


Fig. 32 Starting the Octave program

### 3.3. Command window

Entering commands can be done in two ways: using the "Command Window" and using the "Editor". In the first case, each command is executed immediately after pressing the Enter key. In the second case, so-called script should be written (a series of commands), which then after running the script will be executed line by line. At the moment, it can be a bit incomprehensible, therefore a few examples are presented below.

Switching between the command window and the editor is shown on Fig. 33.

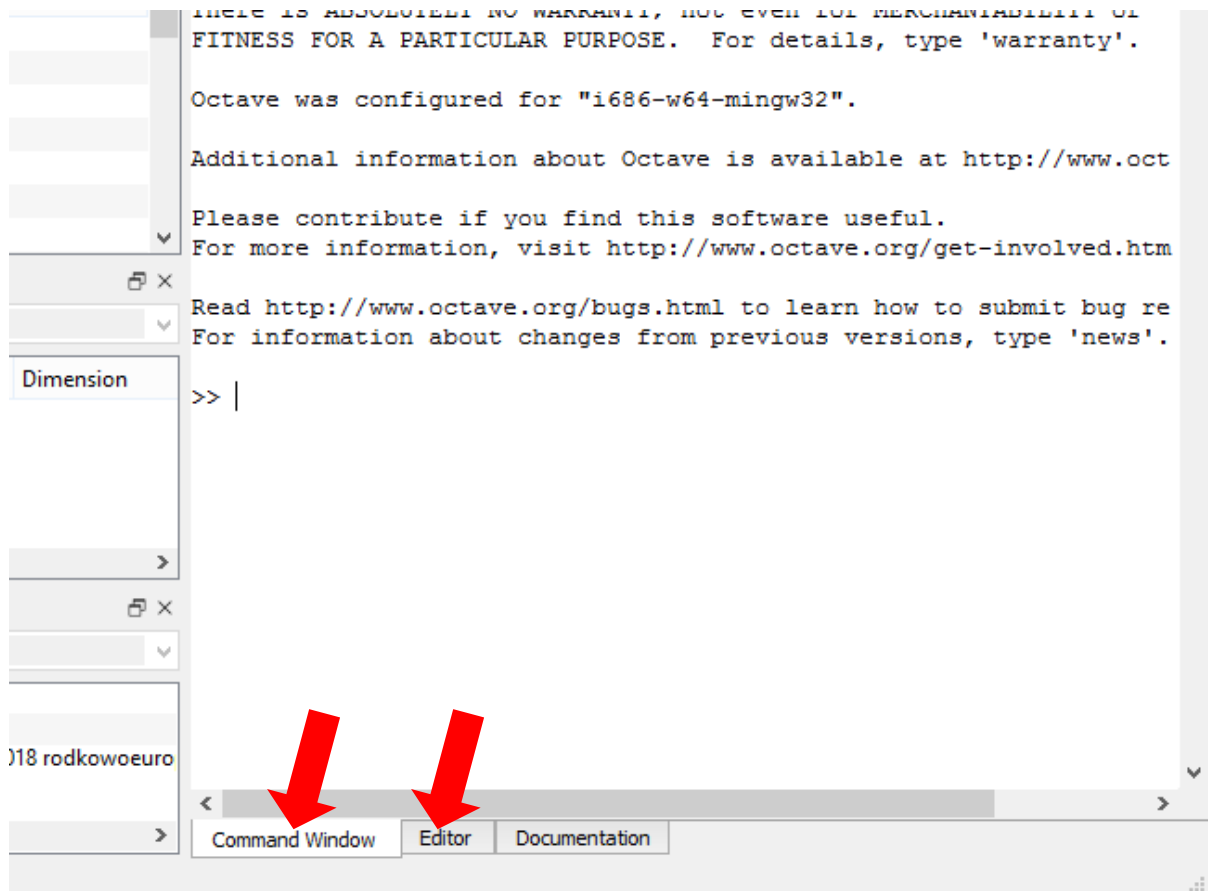
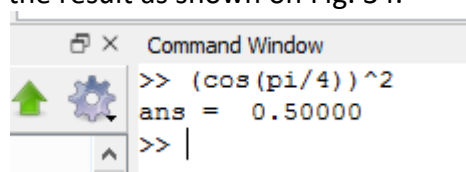


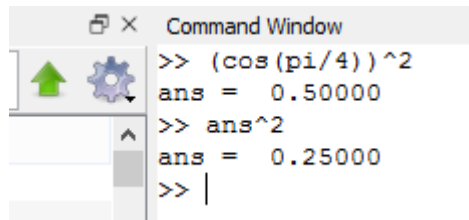
Fig. 33 Command Window and Editor

The Command Window is used to perform short and quick calculations. For example, if we want to calculate the value of the expression  $\cos^2\left(\frac{\pi}{4}\right)$  then in the window just enter the expression `(cos(pi/4))^2`. The ^ operator means raising to the power (in our case to the power of 2). After entering the above expression and confirming it with the Enter key, we will immediately receive the result as shown on Fig. 34.



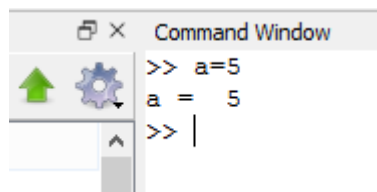
**Fig. 34 Calculation result**

The result of our expression is automatically assigned with the variable named "ans". We can use this variable in a very simple way for further calculations. For example, if we want to get the result again raised to the power of 2, just type on the keyboard expression `ans ^ 2` as shown on Fig. 35. Then the variable "ans" will take on a new value equal in this case to 0.25.



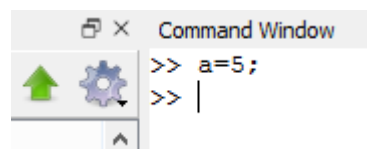
**Fig. 35 Further calculations**

Of course, we can store different values in independently created variables. If, for example, we would like to remember a value of 5 in the variable which we will call for example "a", then we can just write `a = 5` as shown on Fig. 36.



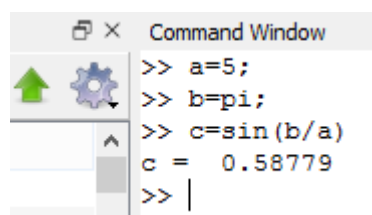
**Fig. 36 Declaration of variable**

In this way, we declared a new variable named "a" and at the same time we assigned it a value of 5. After pressing the Enter key, the result of our command was printed on the screen (`a = 5`). If we want the results not to be displayed on the screen, the command should be ended with a semi-colon as shown on



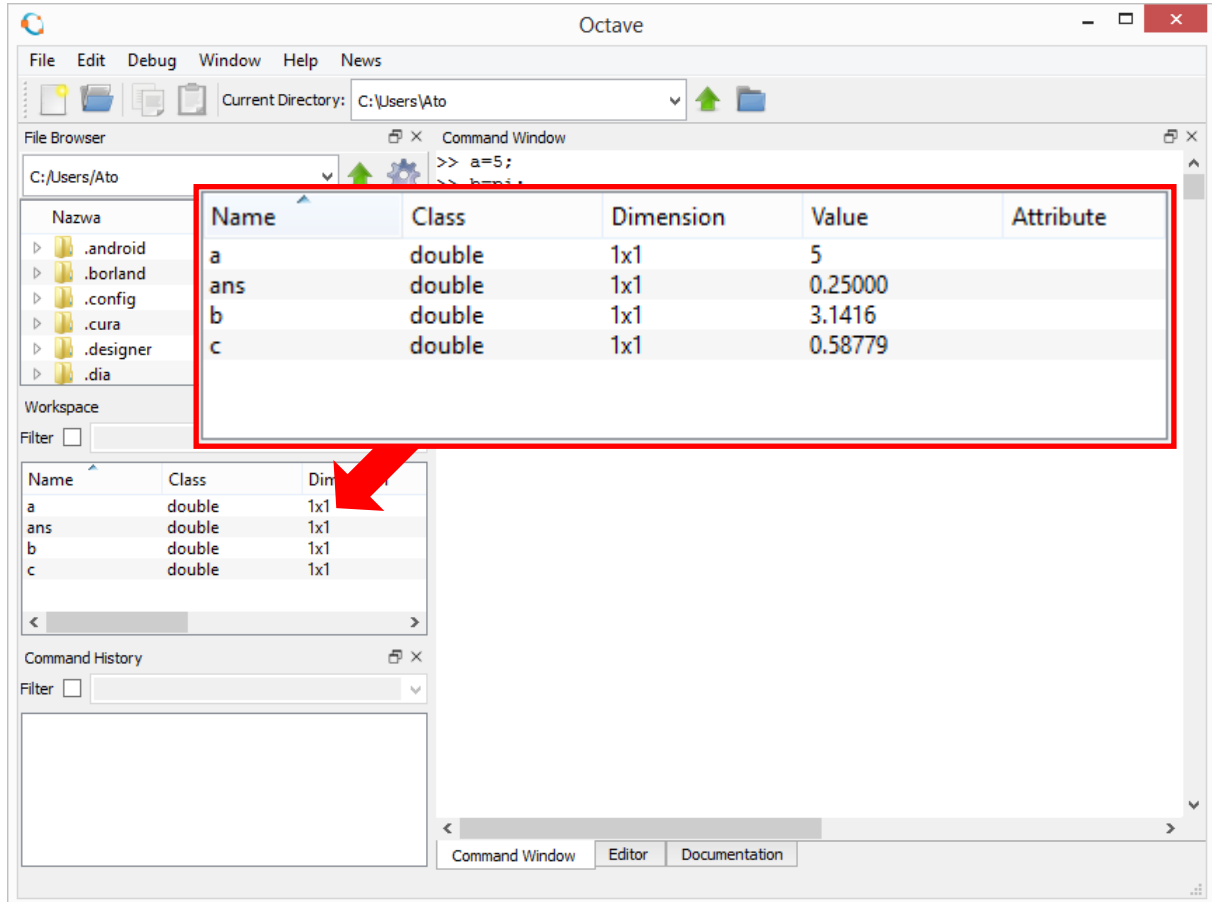
**Fig. 37 The result of the command is not displayed explicitly.**

You can create several variables in this way and perform various calculations (Fig. 38).



**Fig. 38 The calculations**

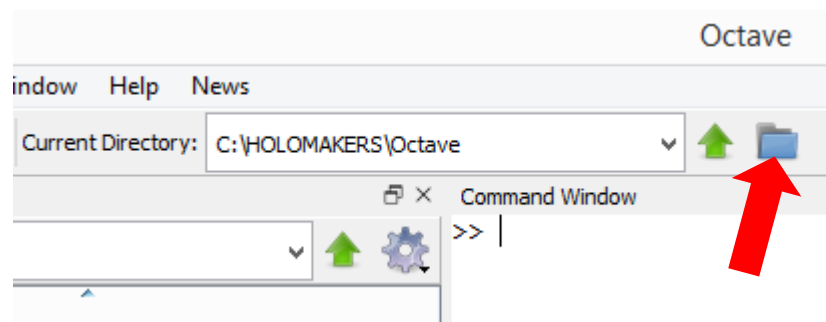
All variables declared so far can be seen in the "Workspace" window (Fig. 39).



**Fig. 39 "Workspace" window**

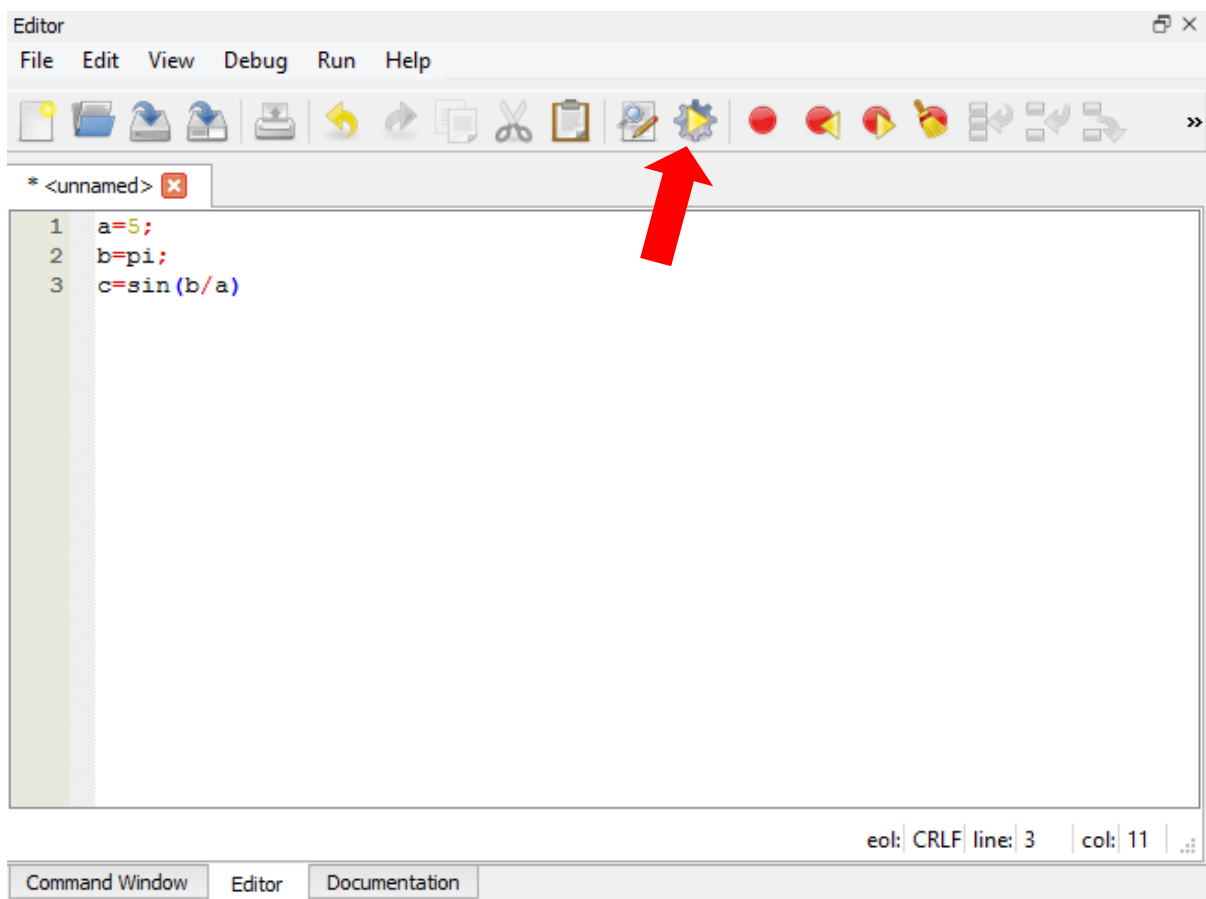
### 3.4. Editor

As it has been said before, the second option to enter commands is to use the editor. In this case, you must first write all the commands and then "run" the written script. Go to the "Editor" tab (Fig. 33) and enter the commands (so-called source code). First, however, it is worth setting the folder in which you want to work. To do this, click the blue icon in accordance with Fig. 40 and then select the appropriate folder (you can, for example, create and choose the folder C: \ HOLOMAKERS \ Octave).



**Fig. 40 Changing the working folder**

Now we can write our own script. Fig. 41 presents a few simple commands. Each command is in a separate line (this allows you to keep the code legible and to avoid possible mistakes). In order for the calculation to take place, you must run the script using the button "Save File and Run" highlighted in Fig. 41 with a red arrow. During the first run, the program will ask you to save the file. The file will be saved in the previously set Current Directory. Give the file the name "test01". The file will be saved as "test01.m", and then the script will be executed (all commands will be executed in the order from the first to the last line).



**Fig. 41 Editor**

To see the result of our actions, return to the "Command Window" (Fig. 42). Only the variable "c" was displayed because the command  $c = \sin(b / a)$  was not ended with a semicolon. The first two commands are ended with a semicolon so they did not display.

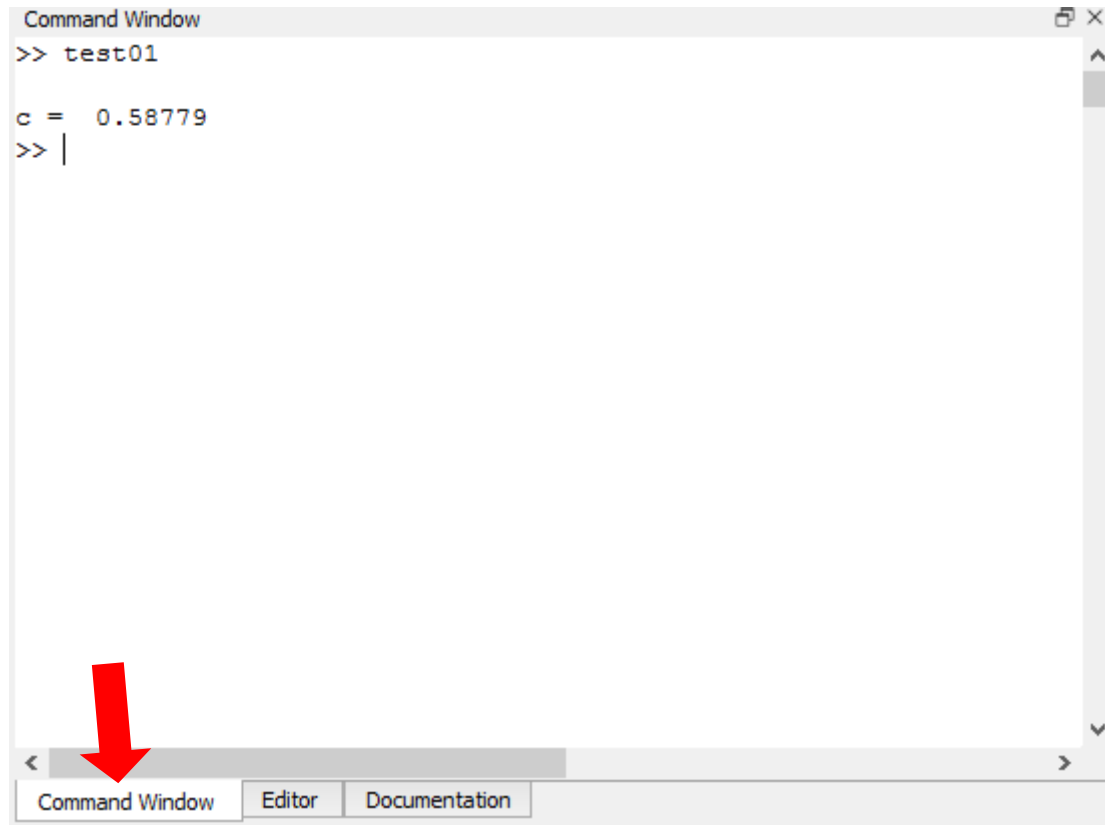


Fig. 42 Calculation result

## 3.5. Basic programming issues in the Octave environment

### 3.5.1. Variables

Variables are used to store the data we need (numbers, characters, text) in the computer's memory. Thanks to this, we can read the value of the variable at any time. In Octave, it is very easy to declare a variable. Just write its name and then assign it a value as it was mentioned in the last chapter. In Octave, one variable can store many numbers. Then such a variable is called an array. The arrays can be one-dimensional or two-dimensional, as illustrated in Fig. 43.

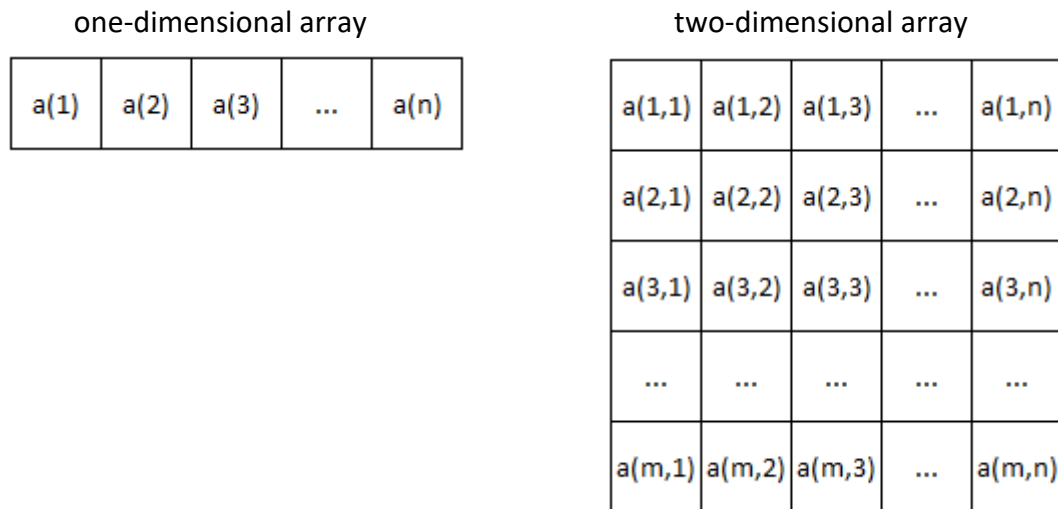


Fig. 43 Array data structure

As you can see, each element of a one-dimensional array is defined by one index and in the case of a two-dimensional array - two indexes. Imagine that we want to declare one variable named `tab1`, in which we will store 5 numbers. In Octave, just write:

```
tab1 = zeros(5,1)
```

It is an array that has 5 rows and one column. Each of the five elements of the array has a value of 0. Now let's declare a 3x3 two-dimensional array also filled with zeros:

```
tab2 = zeros(3,3)
```

Enter the above instructions in Octave in the "Command Window" to see the arrays you created.

The following is an interesting example of creating a one-dimensional array consisting of equidistant numbers from a certain range. Let's create an array and call it `X`, which contains 5 elements in range from 0 to  $\pi$ . We will use the function `linspace(begin of range, end of range, number of elements)`:

```
X=linspace(0,pi,5)
```

Enter the above instruction in Octave in the "Command Window" to see the result.

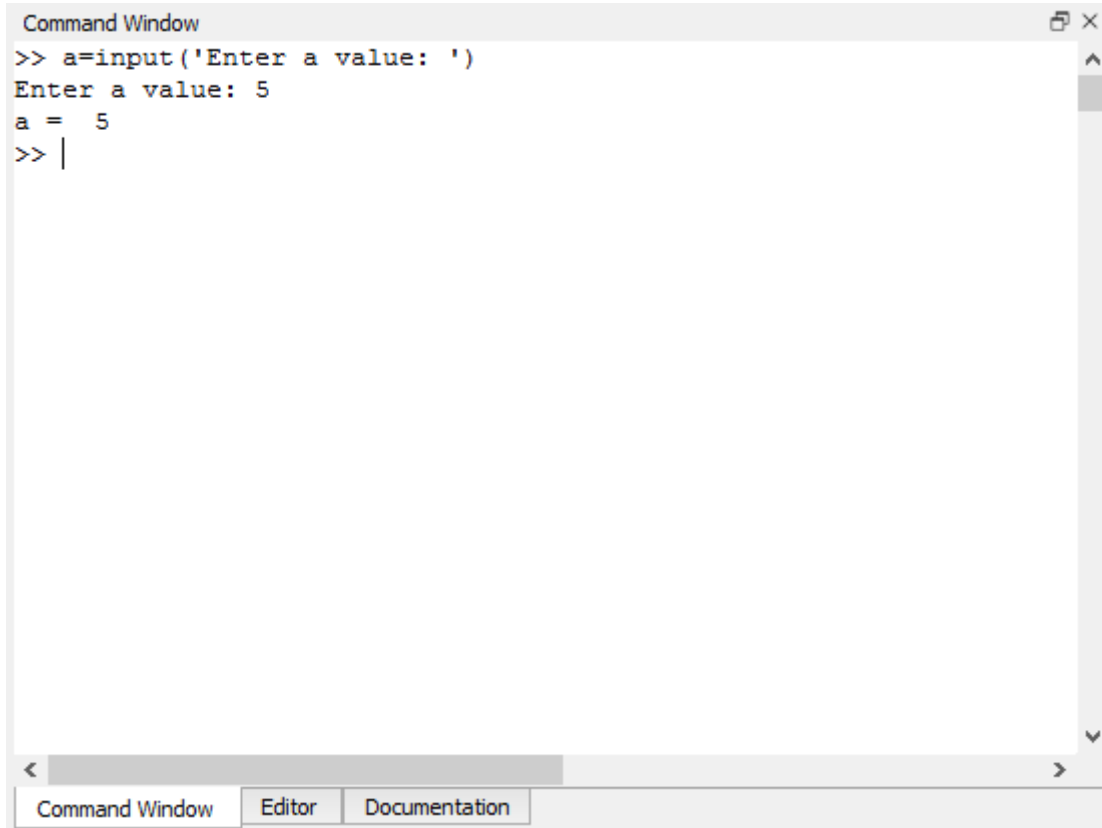
### 3.5.2. Basic input / output operations

The input operation we will discuss here is entering data from the keyboard (the user gives some value from the keyboard, which will then be remembered by the program in the appropriate variable. The output operation is, for example, displaying the calculation result on the screen.

To read a number from the keyboard and save it in a variable named `"a"`, use the following command: `a=input('User text')`. Type the following command in the Command Window and then confirm with Enter.

```
a=input('Enter a value: ')
```

The following line will display the text: `Enter a value:` and the program will wait until you enter a value from the keyboard. Type the number 5 and confirm with Enter. From now, the variable "a" contains the value 5. You should get a result like on Fig. 44.



```
Command Window
>> a=input('Enter a value: ')
Enter a value: 5
a = 5
>> |
```

**Fig. 44 Loading data from the keyboard**

The simplest output operation is to display text on the screen with the command `disp`. In the "Command Window" enter the following command and confirm it with the Enter key:

```
disp('The variable "a" contains the value: '), disp(a)
```

Following text will be displayed: The variable "a" contains the value: and in the next line the value of variable "a" will be displayed. Thus, using the `disp` function, you can display both text and variable values on the screen. If we want to display only the text, we can also use the `puts` function.

### 3.5.3. Operators

We will discuss three classes of operators: arithmetic, relational and logical operators.

1. Arithmetic operators are nothing else than the operator of adding (+), subtracting (-), multiplying (\*), dividing (/) and operator assigning values (=). Thus, for example, the expression `a = c + 2 * b` contains three operators (=, +, \*).
2. Relational operators are used to compare two values (eg, `a > b`). These operators are: equality operator (==), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), different from (!=). The result of the comparison of two values is the so-

called logical value, which is true or false. For example, if we write the expression  $2 == 5$ , it will be false, while the expression  $2 < 5$  will be true.

3. Logical operators as the name suggests are used to perform basic logical operations such as logical sum ( $||$ ), logical conjunction ( $\&\&$ ) and negation operator ( $!$ ). The result of a logical expression is either true or false. For example, the expression  $(5 > 2) \&\& (2 < 3)$  is true because the terms  $5 > 2$  and  $2 < 3$  are both true. An expression used here contains both relational operators ( $<, >$ ) and the logical operator ( $\&\&$ ).

A noteworthy concept is the so-called priority of operators. Although it sounds quite mysterious, every student has met with this from the first years of learning mathematics. Think the result of the following expression:  $5 + 2 * 3 = ?$ . That's right, the result is 11. Note that you intuitively first multiplied and then added. Why did not you add the numbers 5 and 2 first and then you did not multiply the result by 3? This is because you are aware that the multiplication operation is performed first and then the addition. This is the priority of the operators. It tells us which operations in the expression are carried out first and which ones follow. Table 1 shows the operators ordered according to their priority. The logical negation operator has the highest priority. This means that if such an operator is somewhere in the expression it will be executed first. Relation operators have the lowest priority (they are performed at the very end).

Priority	Operator
1	!
2	$\&\&, *, /$
3	$  , +, -$
4	$==, <, >, <=, >=, !=$

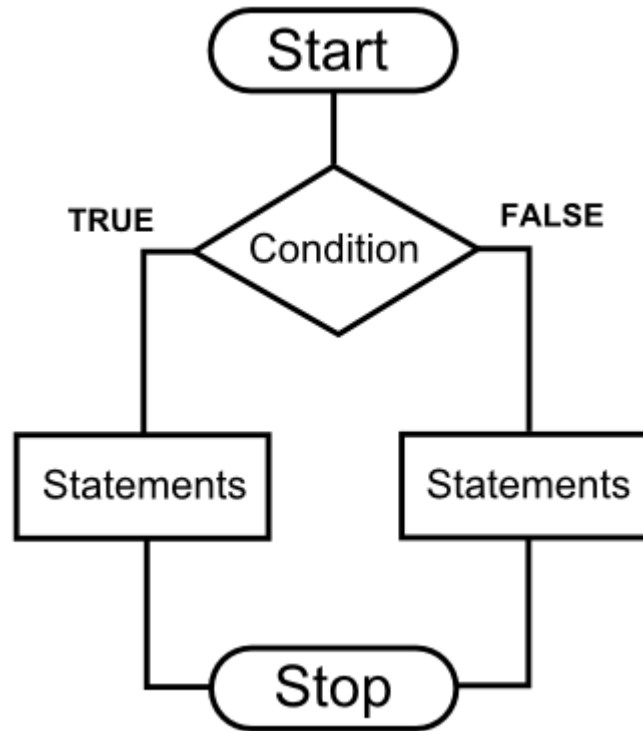
**Table 1 Priority of operators**

A certain obvious thing also known from the maths classes is the fact that if we want to change the order in which operators are executed, we have to put the brackets in the right place. Thus, the expression  $5 + 2 * 3$  gives a value of 11 while the result of the expression  $(5 + 2) * 3$  will be the number 21. Sometimes inserting brackets is necessary for the expression to make sense. This example was already given above:  $(5 > 2) \&\& (2 < 3)$ .

### 3.5.4. Conditional statement

Each of us makes different decisions every day. For example, if the sun shines, we can decide that we will go on a bike and we will stay at home otherwise. This is an example of a simple conditional statement. First, we check some condition (eg whether the sun is shining) and then depending on what is the result of the condition we take the appropriate action (eg we go on a bike or stay at home). The condition is represented by a logical expression that is either true or false. Logical expression can be eg. the sentence "Now the weather is sunny." This sentence (logical expression) is either true or false depending on the weather conditions.

In programming, conditional statements are just as necessary as in everyday life. Fig. 45 presents the structure of a simple conditional instruction. Depending on whether the condition is true or not, other instructions (statements) are executed.




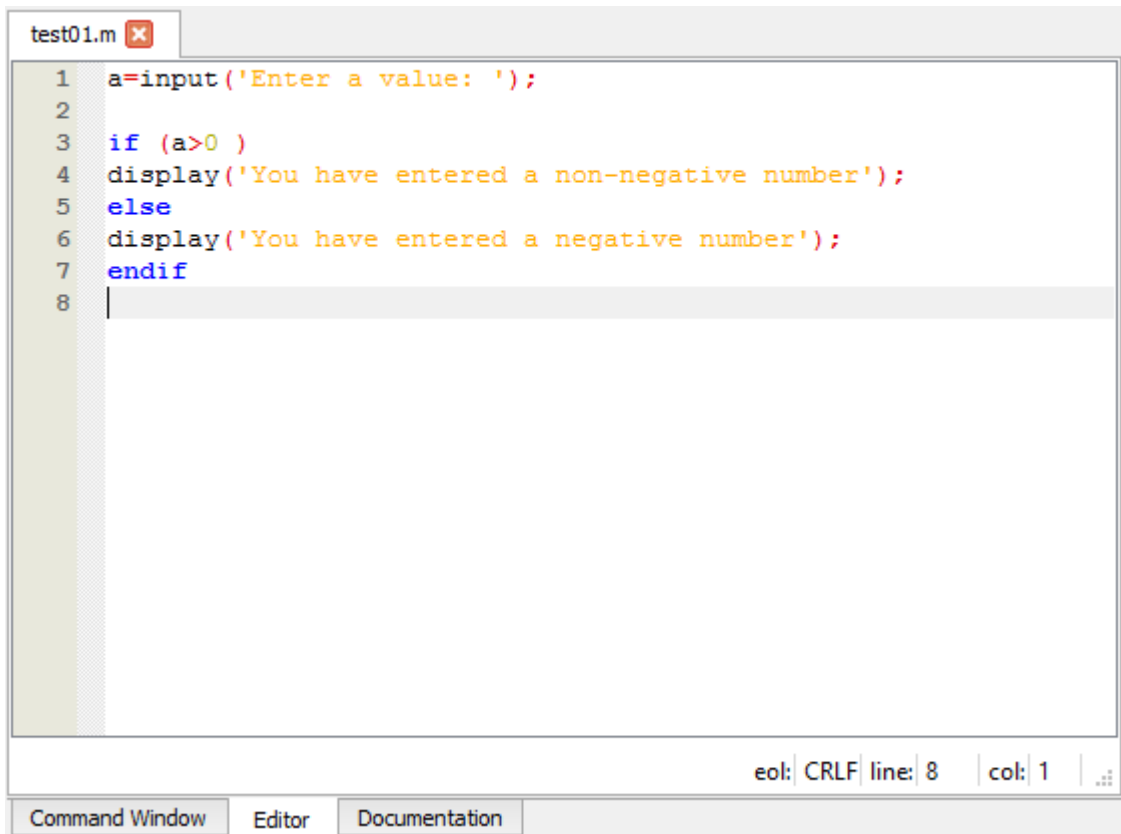
**Fig. 45 Algorithm of a simple conditional statement**

In Octave, simple conditional statement looks as follows:

```

if (condition)
Other statements that will execute if the condition is true
else
Other statements that will execute if the condition is false
endif
  
```

As an example of using the conditional statement, we write a short script that will check if the number entered by the user is negative or non-negative. Write in the Editor window the script as in Fig. 46. Then press the "save file and run" button  and go to the Command Window. After entering the number and pressing the Enter key, the corresponding sentence will appear.



```
test01.m
1 a=input('Enter a value: ');
2
3 if (a>0 )
4 display('You have entered a non-negative number');
5 else
6 display('You have entered a negative number');
7 endif
8
eol: CRLF line: 8 col: 1
Command Window Editor Documentation
```

Fig. 46 Example of using the conditional statement

Often, however, we have a situation where after checking one condition we want to check another one. In Octave, therefore, we should write:

```
if (condition1)
Other statements that will execute if the condition1 is true
elseif (condition2)
Other statements that will execute if the condition1 is false and condition2 is true
else
Other statements that will execute if both condition1 and condition2 are false
endif
```

To illustrate it better, let's modify the last script a bit so that the program also indicates whether the user has provided the number zero (Fig. 47).

```
1 a=input('Enter a value: ');
2
3 if (a>0 )
4 display('You have entered a positive number');
5 elseif (a<0)
6 display('You have entered a negative number');
7 else
8 display('You have entered 0');
9 endif
```

Fig. 47 Example of using the conditional statement


### 3.5.5. "For" loop

Often there is a need to execute some commands many times. In order to avoid entering the same commands many times, the loop is used in programming. We will briefly discuss only one type of loop - the "for" loop. It allows you to execute a set of commands a certain number of times. The structure of the for loop is shown in Fig. 48. It is very simple and tells us that the statements will be carried out 10 times in this case. How do you know that just 10 times? This is defined at the beginning of the loop. A variable, which we called "i" in each subsequent loop run, increases its value by 1, from 1 up to 10. When the variable "i" reaches the end value (in this case 10), the loop ends its operation.

```
1 for i=1:10
2
3 statements...
4
5 end
```

Fig. 48 "For" loop

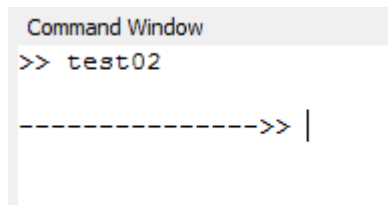
A very simple example of using the "for" loop may be to write a number of characters on the screen. Imagine that we want to be able to display a line consisting of the characters "-". We can write a very simple script for this purpose, shown on Fig. 49. Create a new script using

the icon . During first run, you save it under the name test02. You can see that our line will be 15 characters long (the "for" loop will be executed 15 times).

```
1 for i=1:15
2
3 puts('-');
4
5 end
```

Fig. 49 Displaying a 15-character line

The result of the script operation can be seen in the Command Window (Fig. 50).



```
Command Window
>> test02
----->> |
```

Fig. 50 A 15-character line

We can modify the script a bit so that it allows the user to decide what length of the line to display (Fig. 51). Please note that the loop will be executed n times, where n is the number provided by the user.

```

1 n=input ('enter the length of the line: ');
2
3 for i=1:n
4
5     puts ('-');
6
7 end

```

Fig. 51 Displaying a line with any number of characters

The for loop is very often used for various types of calculations. Can you write a script in which you enter 5 numbers and the program will calculate its average? If not then look at Fig. 52.

```

1 for i=1:5
2
3     a(i) = input('Enter a value: ');
4
5 end
6
7 sum = 0;
8
9 for i = 1:5
10
11     sum = sum + a(i);
12
13 end
14
15 average = sum/5

```

Fig. 52 Calculation of the arithmetic mean

Let's analyze this script now. First, the user enters 5 numbers from the keyboard. These numbers are written to an array named "a". Next, the variable named sum is declared and its value is set to 0. The next loop executes the sum of 5 numbers entered by the user. The last line is the calculation of the arithmetic mean. Since it is not ended with a semicolon, the result will be displayed on the screen.

### 3.5.6. Drawing charts

Octave allows you to draw a graph of a function in a very simple way. To draw a function  $y = f(x)$  we must first create an array of function arguments  $x$  and the corresponding array of function values  $f(x)$ . Suppose you want to draw a function  $y = \sin(x)$  in a range of  $[0, 2\pi]$ . Octave allows declaring an array with  $n$  equidistant elements from the range  $[a, b]$ . This is done using the `linspace(a, b, n)` function. Let's assume that we want to divide our  $[0, 4\pi]$  range into 100 equidistant values. In this case, we must write:

```
x = linspace(0, 4*pi, 100);
```

In this way, we declared the array of 100 elements. Obtaining an array of function values is even simpler, because it is enough to write:

```
y=sin(x);
```

Because x is a 100-element array then y will automatically become an array with the same number of elements. Now just display the graph. To do this, write:

```
plot(x, y)
```

The entire script is presented on Fig. 53.

```
1 x = linspace(0, 4*pi,100);
2 y = sin(x);
3 plot(x, y)
```

Fig. 53 Data preparation and graph display

After starting the script a new window with a graph will appear (see Fig. 54).

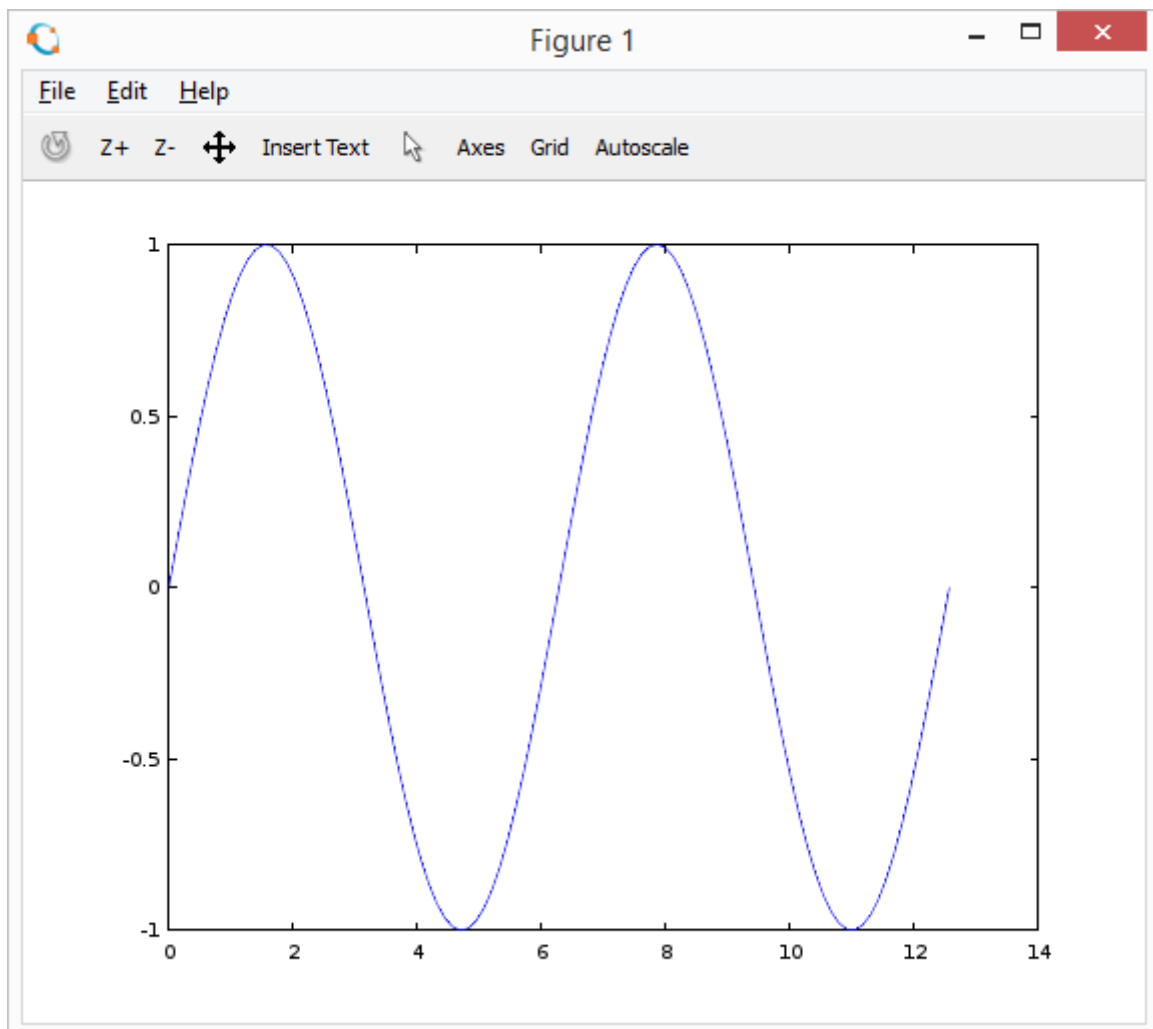


Fig. 54 Chart

## 3.6. Image processing

### 3.6.1. Creating an image

Gray images are represented in Octave as a two-dimensional array, where each element corresponds to one pixel (Fig. 55).

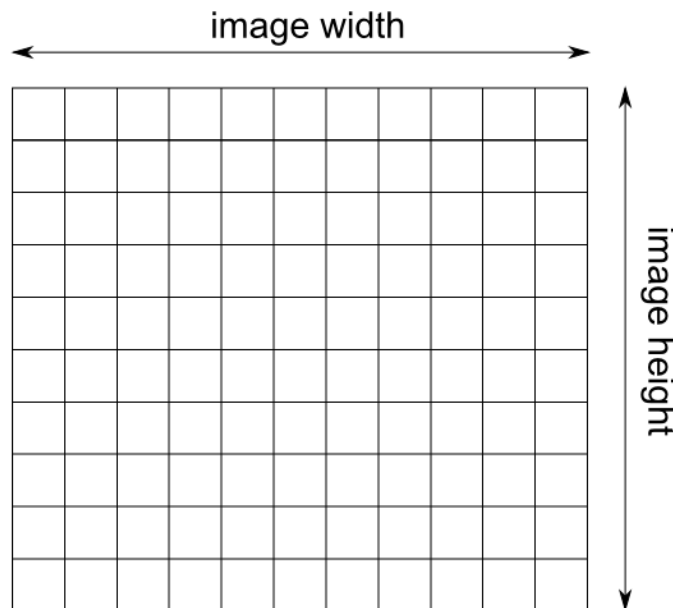


Fig. 55

Creating a new image is simply declaring a two-dimensional array. If you want to create an image with a size of 200x200 pixels, just write:

```
image1 = zeros(200,200);
```

In this way, the variable `image1` was assigned with a 200x200 array filled with zeros (each element of the array is zero). We used the `zeros` function here.

### 3.6.2. Reading / writing image and displaying it on the screen

Often there is a need to load a image file already existing on the disk. To do this, first copy the image to the previously set up "current directory". Then just write the command:

```
image1 = imread('filename');
```

The image file with the specified name is now stored in the variable `image1`.

To display the image from the variable `image1`, use the command:

```
imshow(image1);
```

To display several images, each image should be assigned a different number using the command `figure(number)`. It is also worth to make the window titled using command `title('Image Title')`. Below is an example of displaying two different images in two different windows:

```
figure(1);imshow(image1);title('Image no 1');
figure(2);imshow(image2);title('Image no 2');
```

The image is saved to a file using the command:

```
imwrite(image1,'name.extension');
```

The image can be saved in various formats, eg bmp, png, jpg, tiff, gif.

As an example, upload a image file with the bmp extension to the current working directory and then write a script that will display this image on the screen and save it with various extensions

```
1 image1=imread('test.bmp');
2 imshow(image1);
3 imwrite(image1,'test.gif');
4 imwrite(image1,'test.tiff');
5 imwrite(image1,'test.png');
6 imwrite(image1,'test.jpg');
```

Fig. 56 Converting the image into different formats

### 3.6.3. Color conversion

Octave has many different functions for color conversion. Among them there is a function that converts a color image to gray and a function that converts an image (gray or colored) into a binary image that is composed only of two colors - black and white. These functions, however, are not available by default when the Octave environment is started. To be able to use these functions you need to load the so-called package called `image`. So, at the beginning of the script, write:

```
pkg install image
pkg load image
```

From now, we can freely use the many functions included in this package. To convert a color image to gray, use the `rgb2gray` function. For example, if the `colorImage` variable stores a color image, we can replace it with gray and save it in the `grayImage` variable as follows:

```
grayImage= rgb2gray(colorImage);
```

Similarly, the image can be converted to binary one:

```
bwImage= im2bw(colorImage,0.5);
```

### 3.6.4. Rotation

You can rotate the Octave image relative to the center with any angle using the `imrotate` function. To use this function, we need to indicate the rotated image and the angle of rotation. For example, if you want to place the `Image1` rotated by 45 ° in the `Image2` variable, write:

```
Image2 = imrotate(Image1,45);
```

### 3.6.5. Addition, subtraction, multiplication, division.

As we remember, the image in Octave is represented as a two-dimensional array with dimensions corresponding to the width and height of the image in pixels. Having two images with the same dimensions, we can easily perform operations such as adding or multiplying images. For example, adding two images to each other will simply add the corresponding pixel values to each other (Fig. 57).

Image A		Image B		Image C
8 3 0 8 14		20 12 2 19 15		28 15 2 27 29
18 9 10 13 0		15 11 5 17 15		33 20 15 30 15
17 9 10 7 6	+	16 6 5 9 11	=	33 15 15 16 17
9 19 17 4 19		8 15 13 19 17		17 34 30 23 36
7 1 16 19 7		8 0 9 5 3		15 1 25 24 10

Fig. 57 Adding two images

In a similar way, subtraction is performed as well as multiplication and division. Table 2 presents the names of particular functions in the Octave environment.

Operation	Octave
Add	imadd
Substract	imsubtract
Multiply	immultiply
Division	imdivide

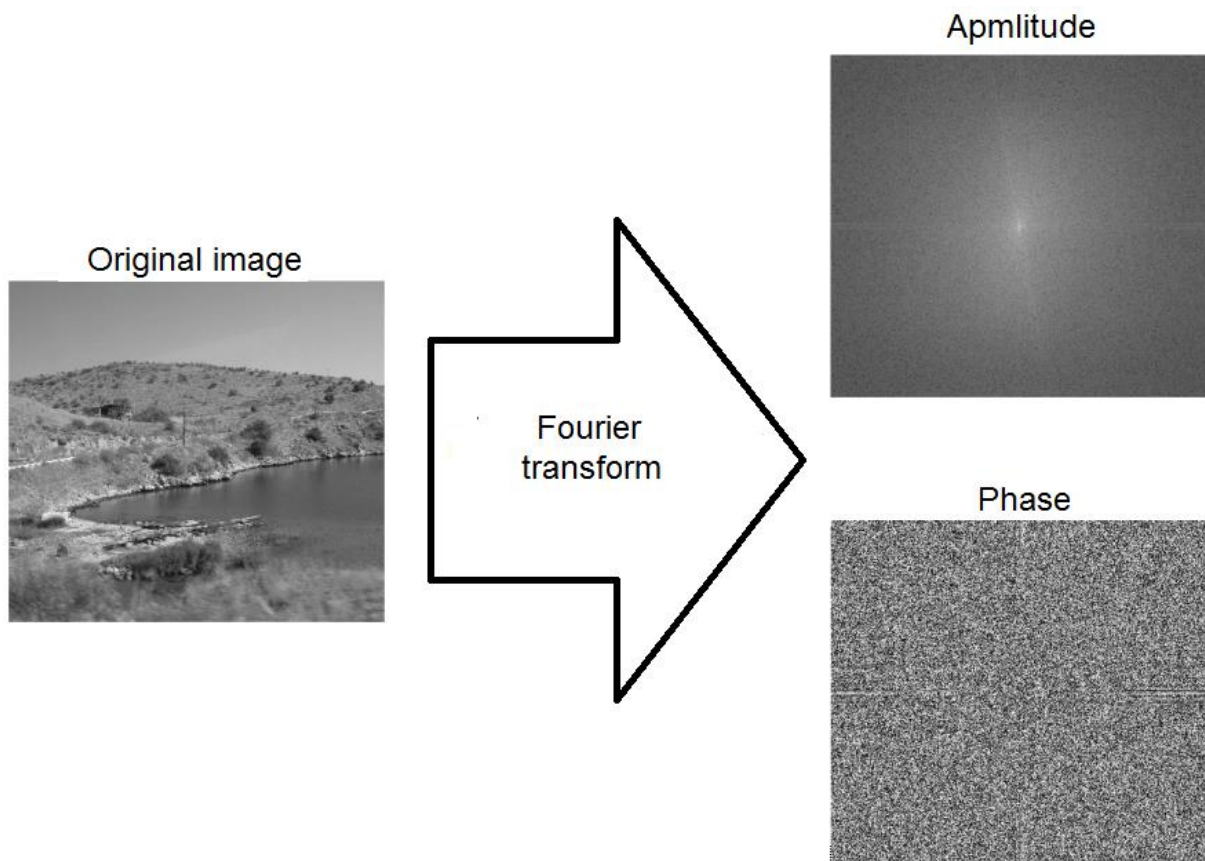
Table 2 Operations on images

Let the variables `imgA` and `imgB` represent two images with the same dimensions. Then we can create an `imgC` image that is the sum or multiplication of these two images:

```
imgC = imadd(imgA, imgB);
imgC = immultiply(imgA, imgB);
```

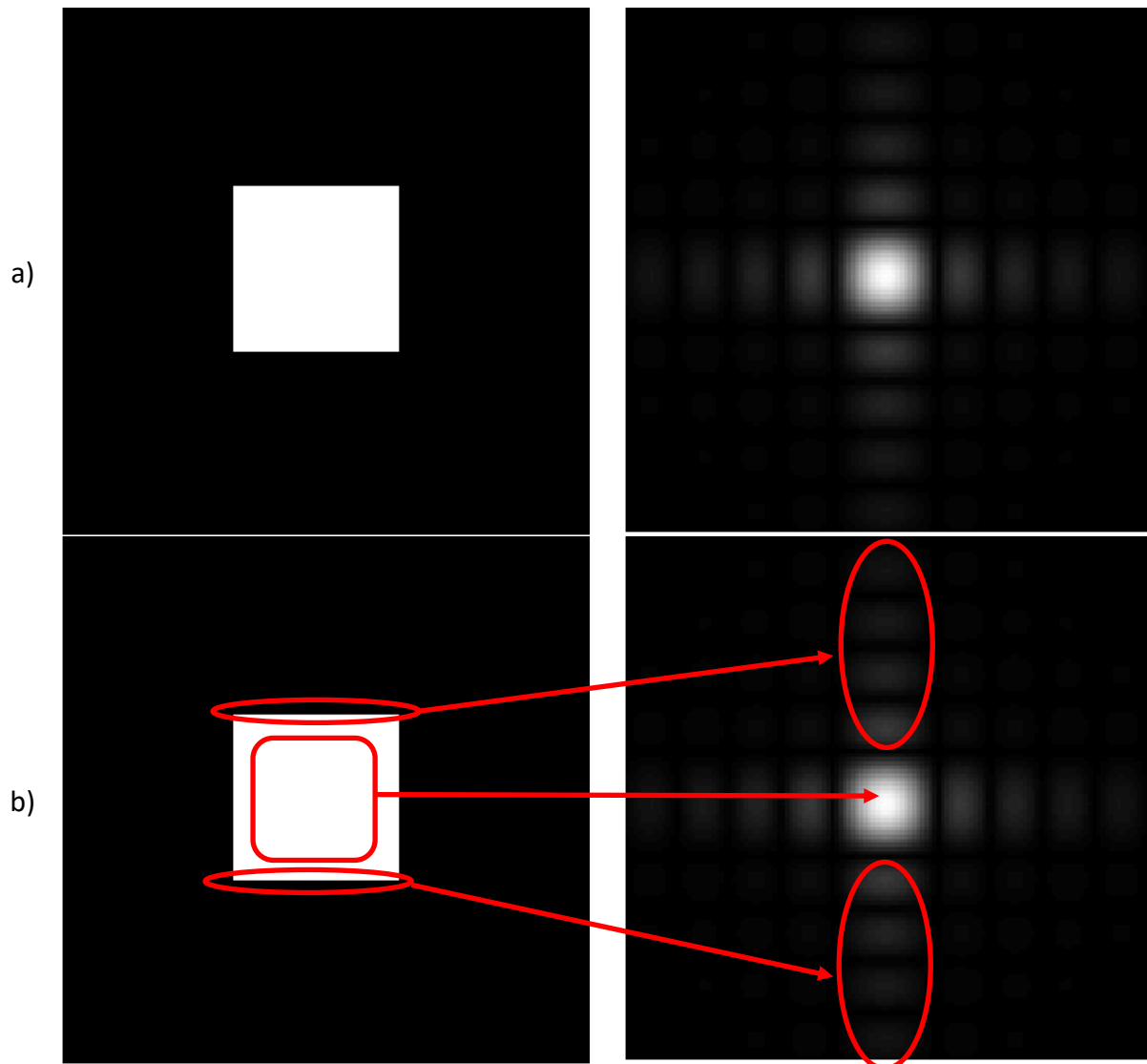
### 3.6.6. Fourier transform

Fourier's transform is a kind of transformation widely used in many fields such as electronics, optics or music. In the case of image processing, a Fourier transform can also be performed. Fourier transform of an image often looks like a collection of random pixels but it contains the same information as the original image (see Fig. 58).



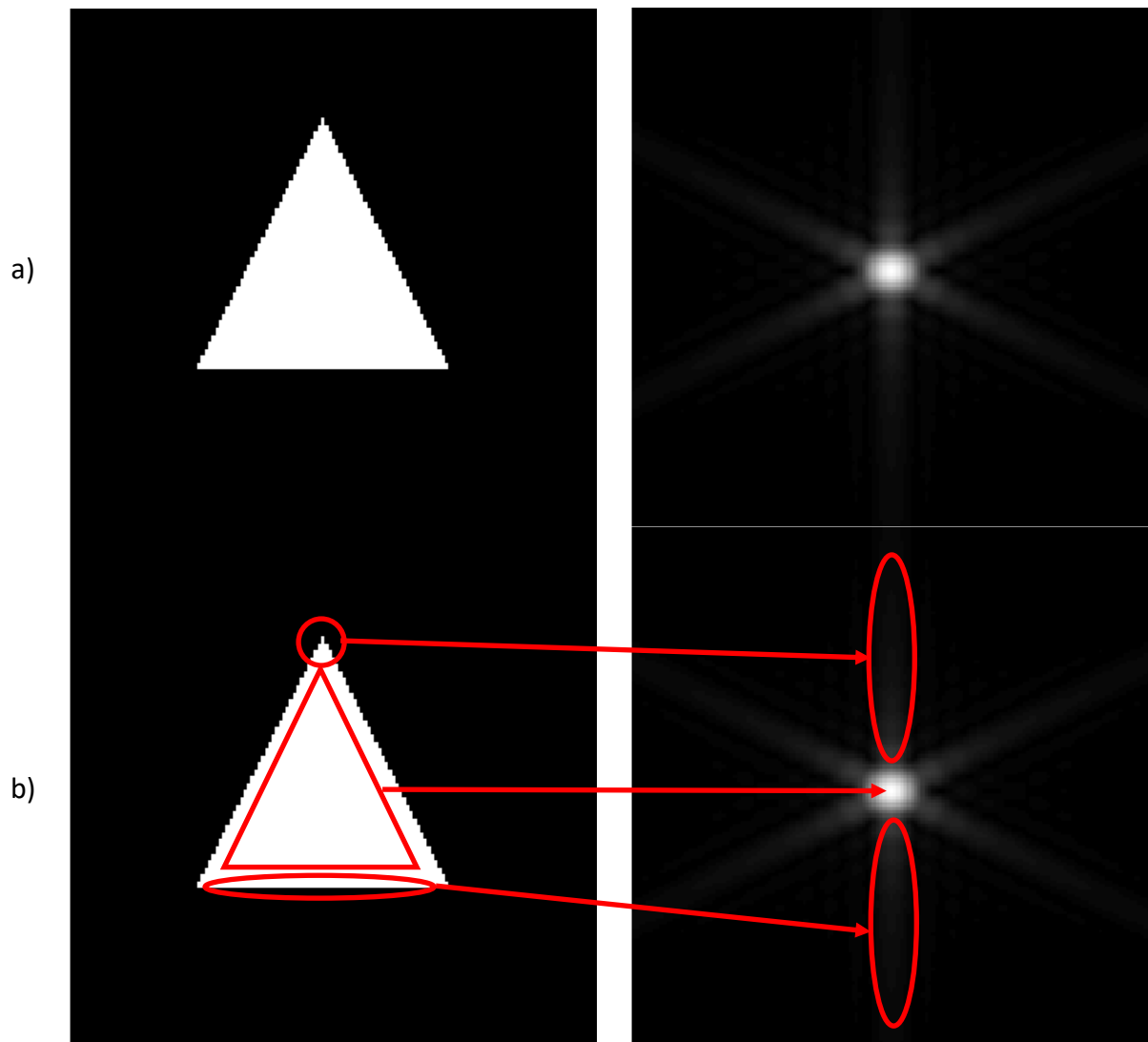
**Fig. 58 Image and its Fourier transform**

Each Fourier transform point has not only an amplitude but also a phase. A detailed discussion of this issue goes well beyond the scope of this guide. Let's just remember that the phase is necessary to reproduce the original image (through the inverse Fourier transform), whereas the amplitude is distributed in the following way: the points in the center of the transform correspond to the smooth areas of the image, while the transform points distant from the center correspond to the edges of the image (the edge is eg the border between smooth sky and mountains, between water and land, etc.). This can be illustrated in a simple example. Fig. 59a presents the white square and its Fourier transform. The middle spot of Fourier transform corresponds to the central part of the square (a smooth image) while each of the four "arms" corresponds to the corresponding edge of the square (Fig. 59b).



**Fig. 59 Fourier transform of the square**

In the case of simple geometric figures, it is quite clearly visible. Fig. 60 presents the image of the triangle and its transformation. The middle spot of the transform corresponds to the central part of the triangle. The three sides of the triangle can be recognized in the transform as three "arms". An additional three "arms" are also visible. These arms represent the vertices of the triangle.



**Fig. 60 Fourier transform of the triangle**

In Octave, the Fourier transform function is called `fft2`. This function works in such a way that the Fourier transform is shifted from the center. It is therefore necessary to additionally use the `fftshift` function. For example, if you want to make a Fourier transform of `img1` image, write:

```
img2=fftshift(fft2(img1));
```

To display the amplitude of the transform one should write:

```
imshow(abs(img2),[]);
```

On the other hand, in order to write out the phase of the transform, one should write:

```
imshow(angle(img2),[]);
```

### 3.7. Algorithm for calculating Computer Generated Holograms (CGH)

To create an analog hologram, we need to have a real object, which will be recorded as a hologram. In the case of computer-generated holograms, the real object is replaced with a graphic file containing any shape (eg a graphic file containing a white text on a black background). The generation of a hologram involves the computer calculation of the interference pattern. The pattern can then be recorded on a holographic plate or film. After passing through such a hologram, the laser light will bend in such a way that it will create a previously designed pattern (eg text). The computer generation of the hologram is carried out using a certain algorithm called the Gerchberg-Saxton algorithm. This algorithm requires providing the input distribution of the designed intensity (i.e. the aforementioned graphic file with the text) and the intensity distribution of the light beam in which the hologram is to be reproduced (also a graphic file). As a result of the algorithm's operation, one can obtain a graphic file with a phase distribution being a designed hologram. The general idea is presented on Fig. 61.

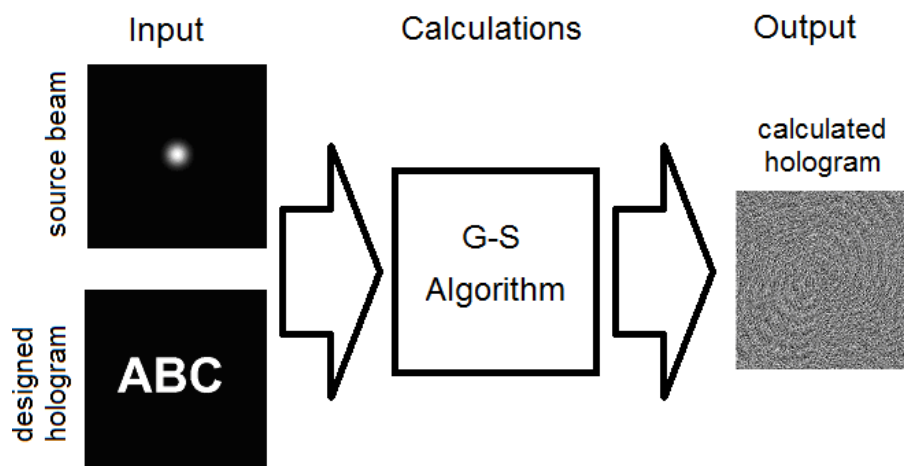


Fig. 61 The input data on the basis of which the hologram is calculated

The Gerchberg-Saxton algorithm is an iterative algorithm, which means that some operations (calculations) are performed many times. Each subsequent execution of the calculation causes that we obtain an increasingly better quality hologram. The scheme of the G-S algorithm is presented on Fig. 62.

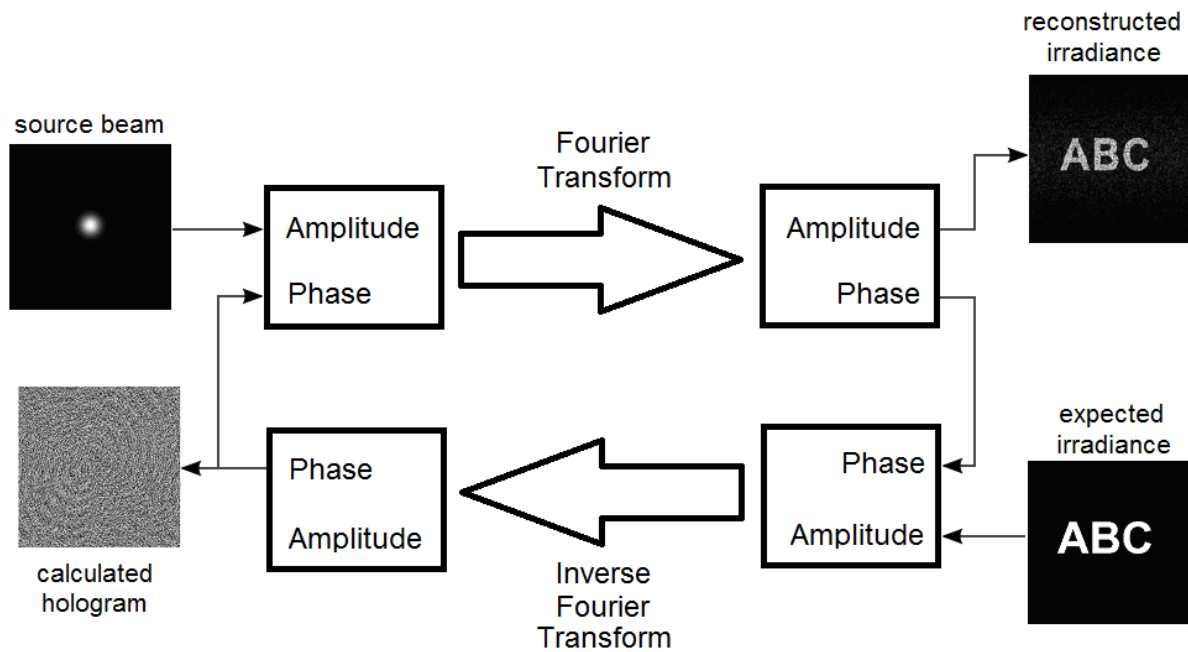


Fig. 62 The Gerchberg-Saxton algorithm

Below is a script that we will use to generate holograms. The calculated hologram will be saved to the CGH.bmp file. Sample input files can be downloaded from the holomakers.eu website.

```
pkg install image
pkg load image

GaussianBeam = imread('GaussianBeam.png');
GaussianBeam = rgb2gray(GaussianBeam);

StartPhase=zeros(1024,1024);

Source=fft2(ifft2(GaussianBeam));
StartPhase=fft2(ifft2(StartPhase));
Source = abs(Source).*exp(1i*angle(StartPhase));

TargetImg = imread('target.bmp');
Target=fft2(ifft2(TargetImg));

A = fftshift(ifft2(fftshift(Target)));
for i=1:20
    B = abs(Source).* exp(1i*angle(A));
    C = fftshift(fft2(fftshift(B)));
    D = abs(Target).* exp(1i*angle(C));
    A = fftshift(ifft2(fftshift(D)));
end
```

```
figure(1);imshow(GaussianBeam);title('Source Intensity');  
figure(2);imshow(Target);title('Expected Intensity');  
figure(3);imshow(angle(A),[]);title('Calculated Hologram');  
figure(4);imshow(abs(C),[]);title('Reconstructed Intensity');  
  
A=angle(A);  
A=A-min(A(:));  
A=A/max(A(:));  
  
imwrite(A, 'CGH.bmp');
```