BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF MATHEMATICS

ÚSTAV MATEMATIKY

VISUALIZATION OF SCALAR FIELDS BY BACK-TO-FRONT METHOD

VIZUALIZACE SKALÁRNÍCH POLÍ METODOU BACK-TO-FRONT

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR — Ing. Hana Gurecká

AUTOR PRÁCE

SUPERVISOR — doc. PaedDr. Dalibor Martišek, Ph.D.

VEDOUCÍ PRÁCE

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Specification Master's Thesis

Department:	Institute of Mathematics
Student:	**Ing. Hana Gurecká**
Study programme:	Applied Sciences in Engineering
Study branch:	Mathematical Engineering
Supervisor:	**doc. PaedDr. Dalibor Martišek, Ph.D.**
Academic year:	2019/20

Pursuant to Act no. 111/1998 concerning universities and the BUT study and examination rules, you have been assigned the following topic by the institute director Master's Thesis:

Visualization of scalar fields by back–to–front method

Concise characteristic of the task:

In this work, method back–to–front and possibilities of its use for three–dimensional scalar data in fixed data grid will be described.

Goals Master's Thesis:

In this work, methods suitable for visualization of scalar data in fixed data grid wil be described and software implemented.
These methods will be tested on the data from fluorescent confocal microscope.
Software solution will be a part of the work.

Recommended bibliography:

ŽÁRA, J. a kol. Moderní počítačová grafika, Computer Press Praha, 1998.

MARTIŠEK, D. Matematické principy grafických systémů, Littera Brno, 2002.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2019/20

In Brno,

L. S.

prof. RNDr. Josef Šlapal, CSc. Director of the Institute	doc. Ing. Jaroslav Katolický, Ph.D. FME dean

ABSTRAKT

Diplomová práce je zaměřena na metody zobrazování skalárních dat v pevné datové mřížce, konkrétně dat získaných užitím fluorescenčního konfokálního mikroskopu. Teoretická část textu začíná představením fungování konfokálních mikroskopů a zasazení problematiky zkoumaných grafických metod do matematického kontextu. Následující kapitola se věnuje odvození integrálu pro zobrazování objemů a z něj vyplývající back-to-front metodu. Teoretická část je zakončena představením metod vhodných pro zobrazování trojrozměrných skalárních dat při použití back-to-front algoritmu. V praktické části je pak popsán implementovaný algoritmus.

ABSTRACT

This master's thesis is focused on scalar data in rigid data mesh imaging. In particular the data are acquired from fluorescent confocal microscope. The theoretical part begins with introduction to confocal microscopy followed by putting the subject of examined graphical methods in mathematical context. Next chapter is devoted to volume rendering integral derivation and consequent back-to-front method. The theoretical part is finalized by introduction of methods suitable for rendering 3D scalar fields using back-to-front algorithm. In the practical part the implemented algorithm is described.

KLÍČOVÁ SLOVA

Konfokální mikroskopie, lineární prostory, projektivní prostory, integrál pro zobrazování objemů, metoda back-to-front, Bresenhamův algoritmus

KEYWORDS

Confocal microscopy, linear spaces, projective spaces, volume rendering integral, back-to-front method, Bresenham algorithm

GURECKÁ, Hana. *Visualization of scalar fields by back-to-front method.* Brno, 2020,(44s). Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/125349. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav matematiky. Vedoucí práce Dalibor Martišek.

AFFIRMATION

I declare that I have written the master's thesis Visualization of scalar fields by back-to-front method on my own according to advice of my master's thesis supervisor and using the sources listed in references.

In Brno, 26^{th} June,2020 .....

Hana Gurecká

I would like to express thanks to my master's thesis supervisor doc. PaedDr. Dalibor Martišek, Ph.D. for numerous comments and valuable suggestions on improving my thesis.

Hana Gurecká

FAKULTA STROJNÍHO INŽENÝRSTVÍ ústav matematiky

OBSAH

1 INTRODUCTION

At the beginning of this thesis we will describe the general design of confocal microscopes and mention their advantage over standard widefield microscope. We will explain how simulated fluorescence works followed by the explanation of two basic principles of confocal microscopy: point to point illumination and rejection of out of focus light. These principles ensure that we can scan one particular point in a focal plane at the time. In the last subchapter we will focus rather on confocal microscopy limitations, particularly light detection limitations and scanning speed limitations.

Chapter three will be devoted to covering the mathematical background of the graphical algorithms and methods described further in the text. We will start with a quick recapitulation of the basic spaces used in computer graphics in general such as linear spaces, normed linear spaces, unitary spaces, affine spaces and Euclidean spaces. At the end of the first subchapter we will put the spaces and related terms to context with analytic geometry. Second subchapter will introduce one of the foundations for future back-to-front algorithm implementation: the projective space with parallel and perspective projections.

Equipped with the mathematical foundations chapter four will focus on the direct volume rendering as a suitable approach to imaging of data with inner structures and back-to-front algorithm in particular. First we will shortly discuss possibilities of volumetric data representation particularly for 3D scalar field. Most methods for direct volume rendering are based on some version of approximation of the volume rendering integral which models propagation of light through participating media. Taking that in mind the thesis will continue with introduction of basic radiometric quantities defining light and its transmission. Following subchapter will make use of radiometric properties in volume rendering integral derivation. Last subchapter of the fourth chapter will introduce discretization of the volume rendering integral denoted as back-to-front visualization method, short notion on transfer functions and projection methods suitable for visualization by the back-to-front method.

The fifth chapter will describe the practical implementation of algorithms introduced in the theoretical part of the thesis. In particular we will describe loading the sets of bitmap images generated by scanning of a specimen by confocal microscope and their representation in the computer. We will follow by description of user defined projective and graphical inputs and their influence on the generated image. As we will have loaded data and defined all variables we will start the description of the computation part of the algorithm. In particular we will derive a method for output window generation and describe fast methods for axis aligned bounding box with ray intersection and Bresenham's line segment rasterization algorithm. The last subchapter will sum up the practical implementation of the back-to-front methods.

In the sixth chapter we will discuss achieved results.

2 CONFOCAL MICROSCOPY

This chapter is based on [1] [2] and [3]. Confocal microscope was introduced by Marvin Minsky in 1955 as a great improvement in observing thick 3D specimen. It is based on two essential principles: point to point illumination and rejection out of focus light. In this chapter we will describe one general design of such microscopes with laser light source. Confocal microscopes can image a specimen by reflected or transmitted light or by simulated fluorescence from fluorophores (fluorescent dyes). The simulated fluorescence is the most common one and even our dataset is acquired using such dyes. Therefore, we will describe the functioning of a confocal microscope with respect to this specimen illuminating method.

Fluorophores are chemical compounds which upon excitation by photon of a given wavelength can emit light of longer wavelength e.g. a fluorophore excited by blue light emits green light. This effect is known as fluorescence. Many different kinds of fluorophores are available with very specific properties. For example they are able to bond to macromolecules of specific parts of a cell (mitochondria, Golgi apparatus, etc.). Then, upon excitation, we are able to see the stained part of the cell glowing by emitted color (Fig.1).

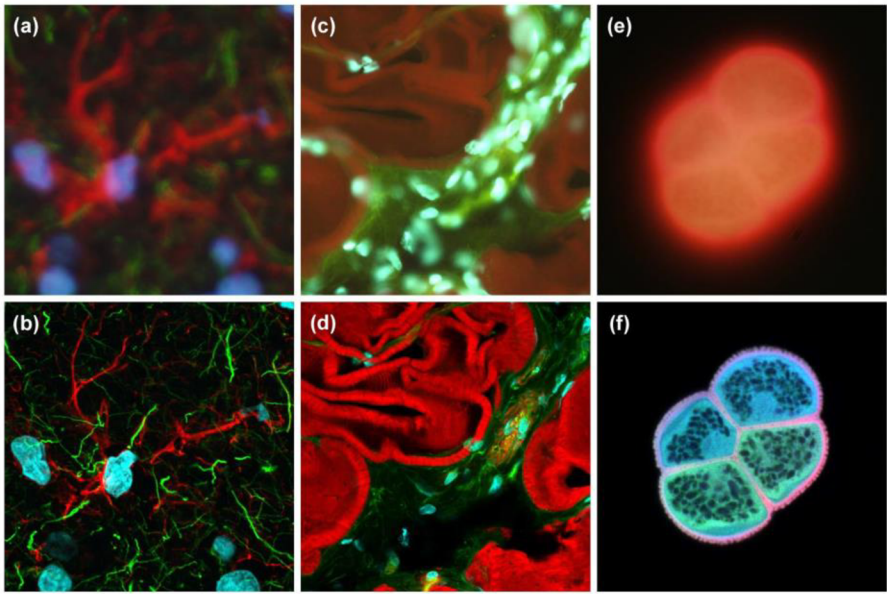


Fig. 1: Widefield (upper) vs. Confocal (lower) fluorescent microscopy images[1]

2.1 Fluorescence microscopy

Classical fluorescence (epifluorescence/widefield) microscope consists of a light source objective and dichroic mirror. Dichroic mirror can transmit a specific part of

[1] Pictures acquired from: https://www.olympus-lifescience.com/en/microscope-resource/primer/techniques/confocal/confocalintro

wavelength spectra and reflect the rest of it. In the (Fig.2) we can see that the light from source (blue) is reflected by a dichroic mirror through the objective onto the specimen. The dyed parts of specimen are excited and emit (green) light. This light with source light travels back through the objective and the dichroic mirror filters the original blue light away by reflecting it back to the source. On the other hand the light emitted by fluorophore with longer wavelength travels through the mirror onto the photodetector. The wide field microscopy has a significant disadvantage which is an illumination of the whole specimen at once. This causes that onto one point of the image plane (camera) falls light not only from the corresponding point of the focal plane but also light scattered from points in other layers of specimen. As a consequence the image from the widefield microscope is blurred (Fig.1).

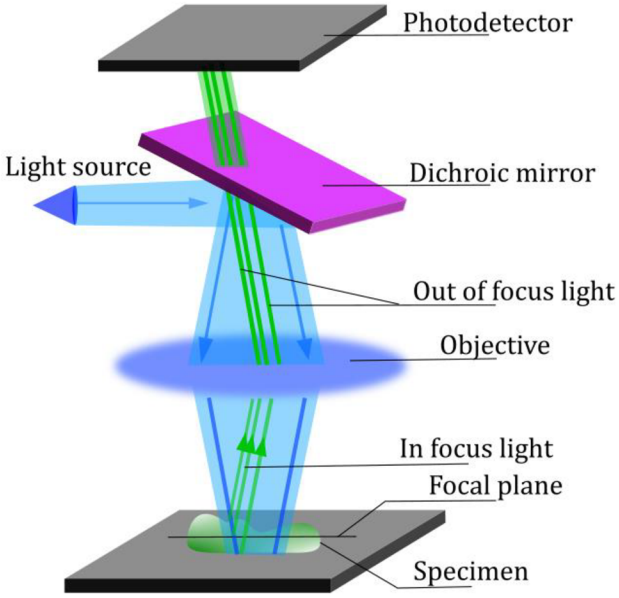


Fig. 2: Widefield microscope scheme

2.2 Confocal fluorescence microscopy

As we mentioned previously, confocal microscopy addresses the problem of secondary fluorescence emitted by not in focus parts of the specimen. Confocal microscope does not take the image at one instance but rather focuses on one specific point in the focal plane at the time. This is achieved by replacing standard objective with two lenses and a screen with a pinhole. The lenses are set up so they transmit light from the focus point of one to the focus point of the other (from here the name “confocal”). Obviously the lenses transmit also out of focus light as well so pinhole is put around the focal point of the second lens in order to let all of the light of point in focus through and significantly attenuate the out of focus light (Fig.3a).

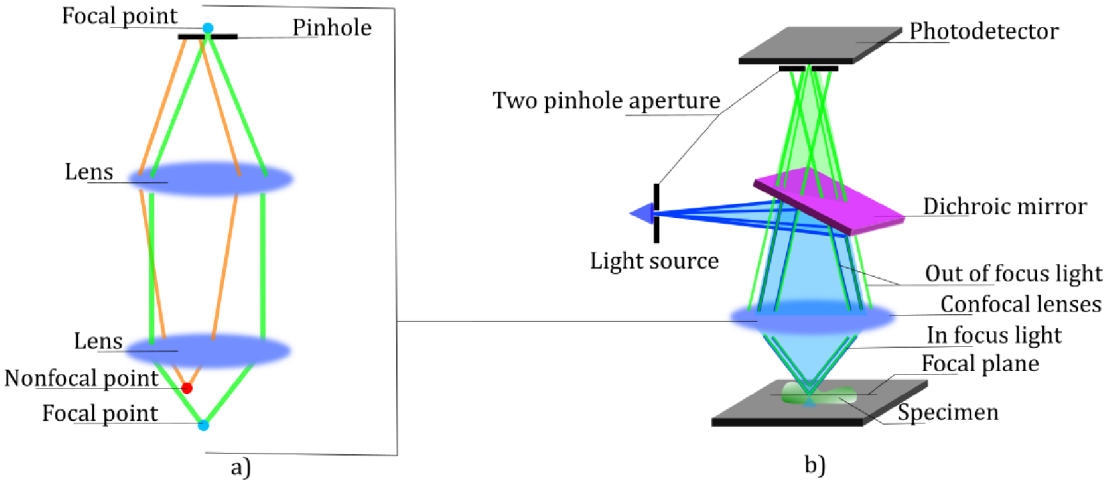


Fig. 3: a) Confocal lenses b) Confocal microscope scheme

Considering this two lenses setting, most of the intensity of the light source is naturally centred in the focus point of the first lens. However, there is still a significant amount of light reaching other parts of the specimen causing excitation and consequently the out of focus light. Another performance improvement can be done if we don't illuminate the whole specimen but instead we focus the excitation light to the point of interest in the focal plane. This is achieved by introducing another pinhole aperture in front of the light source (Fig.3b). In the figure we can observe that such illumination restricts the excitation area to the cone above and below the focus point. Most of the light emitted from this cone area is not confocal with the lens pinhole and therefore is filtered away.

In previous text we described how we scan one exact point in the focal plane. In order to scan the whole plane we need to add two extra motor driven mirrors to the construction. By rotating the mirrors we may reach any point on the focal plane.

Another part of the confocal microscope is the light source. Up to now we have spoken about a light source of given wavelength in general. By filtering away all the redundant light by the screens with pinholes, we are left with very few photons that make it through the whole system. Solution to this is either longer exposure time, which would significantly increase the scanning time of the specimen, or source of very high intensity. Given that the laser source is used in modern confocal microscopes.

The problem of too low light intensity coming to a light detector is addressed also by so called photomultiplier tube detectors (PMT detectors) which replaced standard CCD chips used in widefield microscopy. These detectors are transforming incident light to electric current. Such current is multiplied in the tube and is interpreted as high intensity light. The PMT is connected to a computer, which from incoming information builds up a raster image pixel by pixel. The whole setup of fluorescent confocal laser scanning microscope is shown in (Fig.4).

Thanks to the level of elimination of out of focus light it is possible to acquire sharp images of different depths through specimen. Making a series of cuts in short regular step size along z-axis (optical axis) is referred to as optical sectioning. From

these cuts using visualization algorithm we are able to reconstruct the specimen (this will be discussed in greater detail in chapter 4 of this thesis).

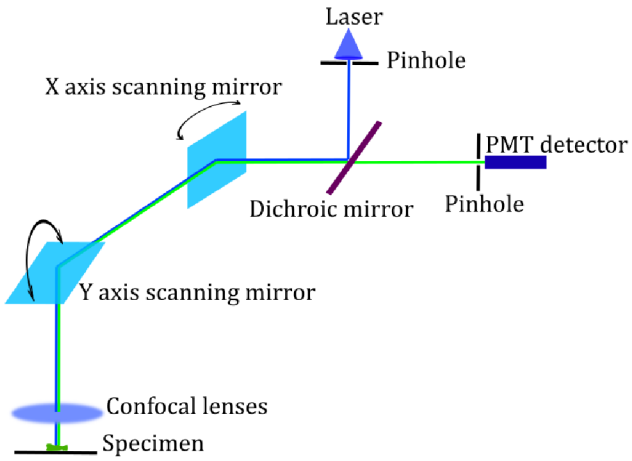


Fig. 4: Fluorescent confocal laser scanning microscope scheme

2.3 Confocal microscopy limitations

Although confocal microscopy offers many advantages, there are some considerations that have to be taken into account. First there are light concerning limitations that are related to how much light can reach the light detector. Another concern is scanning speed regarding the limit of mirror rotation speed.

Light detection limitations

First issue that should be mentioned is the resolution of the image which is a limitation for microscopes in general. In the theory we suppose that a point light source generates point illumination of the specimen. In praxis, due to diffraction of the light, that is not the case. Diffraction phenomena causes that ideal light point generated by circular aperture with a perfect lens is projected onto the image plane in the shape of Airy pattern. The Airy pattern is in two dimensions viewed as a bright central region (disc) with concentric rings around. The radius r_A of Airy disc is dependent on wavelength of the light and aperture of the microscope:

$$r_A = 0{,}61\frac{\lambda}{NA} \tag{1}$$

where λ is the wavelength and NA is numerical aperture of objective. Aperture is given by:

$$NA = n \cdot sin\theta \tag{2}$$

with n as index of refraction of the medium in which the lens is situated and θ is half angle from which the light can approach the lens. The Airy pattern limits maximal resolution of the image. According to Rayleigh criterion the two points are distinguishable if they are at least their radius of Airy disc away from each other. Maximal resolution for confocal microscopes is typically about 200nm. As we are

interested in optical cuts through the specimen we have to take into consideration the z-axis. Such generalization of Airy disc into 3rd dimension is called point-spread function and is ellipsoid shaped rather than a sphere as optical axis resolution tends to be poorer, typically around 500*nm*.

Another consideration, that has to be taken, is about the pinhole size. It is desirable to minimize the pinhole and consequently make the resolution along the optical axis smaller. Although, if the pinhole size is too small, then very few photons can make it to the detector and signal-to-noise ratio will decrease. As a fairly good approximation of pinhole size is Airy disc radius as a smaller pinhole doesn't bring much further improvement.

Further question is how intensive laser should be used. The obvious answer would be as intensive as possible as the signal-to-noise ratio would be increased. However, high intensity laser may damage the specimen and degrade the fluorophore as it gets saturated.

Finally the amount of photons that reach the PMT detector can be influenced by concentration of fluorophore coloring the specimen. This signal enhancement has a limit too as the molecules can quench each and limit the fluorescence from deeper parts of the specimen as the fluorophores in the shallower parts absorb most of the exciting photons.

Scanning speed

As we mentioned before, the point scanning in confocal microscopes is achieved by two rotating mirrors for x and y axis. Originally two galvanometers were used as motors for the rotation. By this method it was possible to acquire one image in 0,1-1s. However, this rate is not sufficient for observation of dynamical processes. Moreover the long relatively exposure to the intensive laser may cause damage to the specimen. As a consequence two main methods were developed for enhancing the scanning speed.

First method is based on replacing the galvanometer scanning horizontally (fast) by acousto-optic deflector (AOD). AOD deflector is a crystal which changes its refractive index depending on sound frequency input. By fast altering the input frequencies it is possible to steer the laser with great precision and make around 30 images per second. Disadvantage of using AOD deflectors is that they are deflecting different wavelengths differently. Meaning that we may point the source light with great precision but the fluorescence from the specimen of longer wavelength is not reflected to the source direction, as in the case of a mirror. So instead of the pinhole it is used as a slit for descanning fluorescence from the vertical scanning (slower) galvanometer driven mirror. As a consequence the resulting image is distorted but still high quality.

Second method replaces two mirrors and an excitation light pinhole by a spinning disc (Nipkow disc) with a mask of thousands of pinholes in order to scan multiple pixels at once. The pinholes are arranged so every pixel in the image is reached and scanning of one image can take about 1/15 of the spin. With 40 revolutions per second it is possible to acquire up to 600 images per second. The first drawback to using Nipkow disc is that a few hundreds of points are illuminated at once and therefore

background fluorescence is increased which is especially significant in the thick specimen. Also, as the disc rotates quickly, the light coming from the specimen through pinholes and to the detector is weakened and stronger fluorophores need to be applied.

3 MATHEMATICAL BACKGROUND

In this chapter we will state the basic mathematical foundations for computer graphics and back-to-front algorithm in particular. First we will review the linear algebra structures supporting the notion of Euclidean space and analytic geometry. Further we will continue with basic projection methods in the context of analytic geometry terms.

3.1 Spaces

This subchapter will recapitulate linear, unitary and affine spaces and basic concepts related to them in order to introduce Euclidean space and analytic geometry. The information and definitions in this chapter are taken from [4][5]. As the purpose of this chapter is not meant to be a full introduction to algebra but rather brief recapitulation, the proofs of the basic notions are omitted and can be found in the cited literature.

Linear space

Def.: 1: *Linear (vector) space* over a field $(\mathrm{F}; +; .)$ is a set V together with operations $\bigoplus: V \times V \to V$ and $\bigotimes: V \times \mathrm{F} \to V$ such that for each $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in V$ and scalars $a, b \in \mathrm{F}$ the following axioms hold:

i. $(\boldsymbol{u} \oplus \boldsymbol{v}) \oplus \boldsymbol{w} = \boldsymbol{u} \oplus (\boldsymbol{v} \oplus \boldsymbol{w})$
ii. $\exists \mathbf{0}: \mathbf{0} \oplus \boldsymbol{u} = \boldsymbol{u} \oplus \mathbf{0} = \boldsymbol{u}$
iii. $\exists \boldsymbol{u}^*: \boldsymbol{u}^* \oplus \boldsymbol{u} = \boldsymbol{u} \oplus \boldsymbol{u}^* = \mathbf{0}$
iv. $\boldsymbol{u} \oplus \boldsymbol{v} = \boldsymbol{v} \oplus \boldsymbol{u}$
v. $a \otimes (b \otimes \boldsymbol{u}) = (a \cdot b) \otimes \boldsymbol{u}$
vi. $1 \otimes \boldsymbol{u} = \boldsymbol{u}$
vii. $(a + b) \otimes \boldsymbol{u} = (a \otimes \boldsymbol{u}) \oplus (b \otimes \boldsymbol{u})$
viii. $a \otimes (\boldsymbol{u} \oplus \boldsymbol{v}) = (a \otimes \boldsymbol{u}) \oplus (a \otimes \boldsymbol{v})$.

Elements of the linear space are called *vectors* and operation $\bigoplus$ is vector addition, $\bigotimes$ is multiplication of a vector by scalar, $\cdot$ is scalar multiplication and + is scalar addition. The symbols $\mathbf{0}$ and $\boldsymbol{u}^*$denote neutral vector (*zero vector*) and opposite vector, respectively.

Def.: 2: *Normed linear space* is linear space equipped with mapping $\| \|: V \to \mathrm{F}$ such that for each $a \in \mathrm{F}$ and $\boldsymbol{u}, \boldsymbol{v} \in V$ it satisfies:

i. $\|\boldsymbol{u}\| \geq 0;\ \|\boldsymbol{u}\| = 0 \leftrightarrow \boldsymbol{u} = \mathbf{0}$
ii. $\|a \otimes \boldsymbol{u}\| = |a|\|\boldsymbol{u}\|$
iii. $\|\boldsymbol{u} \oplus \boldsymbol{v}\| \leq \|\boldsymbol{u}\| + \|\boldsymbol{v}\|$.

The mapping $\| \|$ is called *norm*.

Def.: 3: Linear space V equipped with mapping $*: V \times V \to \mathrm{F}$ such that for each $a \in \mathrm{F}$ and $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in V$ it satisfies:

i. $\boldsymbol{u} * \boldsymbol{u} \geq \mathbf{0};\ \boldsymbol{u} * \boldsymbol{u} = 0 \leftrightarrow \boldsymbol{u} = \mathbf{0}$
ii. $\boldsymbol{u} * \boldsymbol{v} = \boldsymbol{v} * \boldsymbol{u}$
iii. $(\boldsymbol{u} \oplus \boldsymbol{v}) * \boldsymbol{w} = (\boldsymbol{u} * \boldsymbol{w}) + (\boldsymbol{v} * \boldsymbol{w})$
iv. $(c \otimes \boldsymbol{u}) * \boldsymbol{v} = c \otimes (\boldsymbol{u} * \boldsymbol{v})$

is called *unitary space* and the mapping is called *scalar product.*

Further in the text we will denote the operations $\boldsymbol{u} \oplus \boldsymbol{v}$ as $\boldsymbol{u} + \boldsymbol{v}$, $a \otimes \boldsymbol{v}$ as $a \cdot \boldsymbol{v}$ or $a\boldsymbol{v}$ keeping in mind that those are vector operations in the sense of (Def.1). Also opposite vector $\boldsymbol{u}^*$ will be replaced by $-\boldsymbol{u}$.

Note that unitary space V equipped with mapping $\| \|: V \to \mathrm{F}$ satisfying for $\forall \boldsymbol{u} \in V$:

$$\|\boldsymbol{u}\| = \sqrt{\boldsymbol{u} * \boldsymbol{u}} \tag{3}$$

is also a normed linear space. Moreover both normed linear space and unitary space have metrics $\rho: V \times V \to \mathrm{F}$ induced by norm and scalar product, respectively, defined for each , $\boldsymbol{v} \in V$:

$$\rho(\boldsymbol{u}, \boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\| \tag{4}$$

$$\rho(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{(\boldsymbol{u} - \boldsymbol{v}) * (\boldsymbol{u} - \boldsymbol{v})}. \tag{5}$$

Def.: 4: A subset $\{\boldsymbol{u}_i\} \subset V; i = 1..n$ of a vector space is called *linearly dependent* if there exist $c_i \in \mathrm{F}; i = 1..n$ such that $\sum_{i=1}^{n} c_i \boldsymbol{u}_i = \mathbf{0}$ and at least one $c_i \neq 0$. Otherwise, the subset is called *linearly independent*.

Def.: 5: A vector $\boldsymbol{v} \in V$ is called *linear combination* of nonempty set $\{\boldsymbol{u}_i\} \subset V; i = 1..n$ if and only if there exist $c_i \in \mathrm{F}; i = 1..n$ such that $\sum_{i=1}^{n} c_i \boldsymbol{u}_i = \boldsymbol{v}$. A linear combination of empty set is the neutral vector $\mathbf{0}$.

Def.: 6: A linear space L formed as set of all linear combinations of a set $M \subset V$ is called (*linear*) *span* and denoted as $\langle \boldsymbol{M} \rangle$.

Def.: 7: A subset $B \subset V$ is called *finite basis of linear space* V if and only if it is linearly independent and its span is equal to V.

Remark: Although the basis of given linear space V is obviously not unique, it is possible to show, that for arbitrary choice of the base vectors for V, the cardinality $|B|$ remains the same.

Def.: 8: Let a linear space have a finite base B. Then the number n of vectors in B is called *dimension* of the linear space and the space is denoted as V_n.

Remark: Particularly interesting is the case, when we want to determine dimension of the trivial vector space $V = \{\mathbf{0}\}$. The only vector in the space is clearly linearly dependent by (Def.:4). In order to determine dimension, we have to find some basis of V. As the base vectors are expected to be linearly independent the base $B = \emptyset$. Using (Def.:5) we can say that the empty set really generates the zero vector space: $\langle \emptyset \rangle = \{\mathbf{0}\}$ and because $|\emptyset|$=0 the trivial vector space has zero dimension.

Def.: 9: Let V_n be a linear space with a base $\mathcal{B}$. Then for any $\boldsymbol{v} \in V_n$ we call the coefficients c_i satisfying the equation $\boldsymbol{v} = \sum_{i=1}^{n} c_i \boldsymbol{u}_i; i = 1..n$ coordinates of the vector **v** in basis $\mathcal{B}$ and we write $\boldsymbol{v} = (c_1, .., c_n)_{\mathcal{B}}$.

Def.: 10: Let V be a unitary space. Then the vectors $\boldsymbol{u}, \boldsymbol{v} \in V$ satisfying $\boldsymbol{u} * \boldsymbol{v} = \boldsymbol{0}$ are called *orthogonal*. If in addition the vectors satisfy $\|\boldsymbol{u}\| = \|\boldsymbol{v}\| = 1$ they are called *orthonormal*.

Def.: 11: A subset of unitary space $G = \{\boldsymbol{u}_i\} \subset V; i = 1..n$ is called *orthogonal system* if each pair of vectors where $i \neq j$ it satisfies $\boldsymbol{u}_i * \boldsymbol{u}_j = \boldsymbol{0}$. If in addition $\forall \boldsymbol{u}_i \in G$: $\|\boldsymbol{u}_i\| = 1$ we call the set *G orthonormal system.*

Affine space

Def.: 12: Let A_n be a nonempty set with associated linear space V_n over a field F and with a mapping $\varphi: A_n \times A_n \to V_n$ with following properties:

i. $\forall A \in A_n$ and $\forall \boldsymbol{u} \in V_n$: $\exists!\ B \in A_n$ such that $\varphi(A, B) = \boldsymbol{u}$
ii. $\forall\ A, B, C \in A_n : \varphi(A, C) = \varphi(A, B) + \varphi(B, C)$.

Such a set is called an *affine space* with dimension *n*. Elements of affine space are called *points*. An ordered pair [A,B] is called the *location of vector* **u**. We denote: $\boldsymbol{u} = \varphi(A, B) = \overline{AB} = B - A$ or analogically $B = \boldsymbol{u} + A$**.**

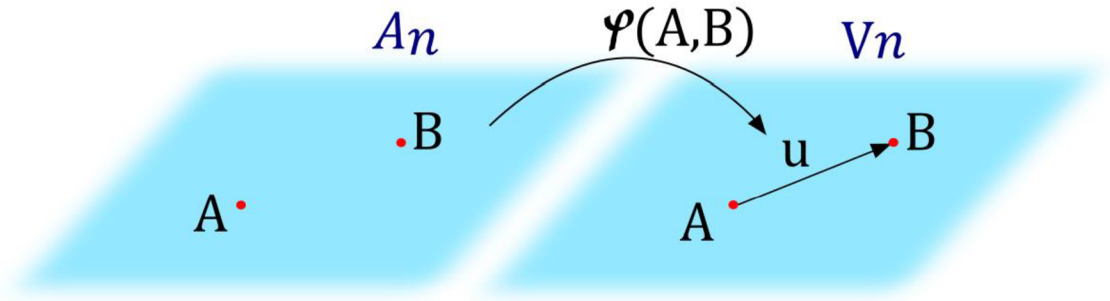


Fig. 5: Affine spaces

At this point it is convenient to highlight that without the associated linear space, the set of points has neither relation between two points nor neutral element (*origin*), therefore no way to uniquely specify points in the terms of coordinates.

Def.: 13: Let A_n be an affine space with associated linear space V_n and with point $O \in A_n$ called *origin*. Let $\mathcal{B} = \{e_1, .., e_n\}$ be a base of the V_n. Then the ordered (n+1)-tuple $\langle O, e_1, .., e_n \rangle$ is called a *linear coordinate system.* Moreover for $A \in A_n$ where $A - O = a_1 e_1 + .. + a_n e_n$ we denote $A = [a_1, .., a_n]$ where the ordered coefficients $a_1, .., a_n$ are called *affine coordinates* of point *A*.

Def.: 14: Let $\boldsymbol{u} \in V_n$ be a nonzero vector, $A \in A_n$ a point and a constant. Then the set $p = \{B \in A_n | \ B = \lambda \boldsymbol{u} + A; \lambda \in \mathbb{R}\}$ is called *(affine) line.* Moreover all the points in the set *p* are called *collinear.*

Def.: 15: Let $B, C, D \in A_n$ collinear and mutually different points, $\boldsymbol{u}, \boldsymbol{v} \in V_n$ be defined as $\boldsymbol{u} = B - D$, $\boldsymbol{v} = C - D$. Then the number λ such that $\boldsymbol{u} = \lambda \boldsymbol{v}$. is called affine ratio of the points *B,C,D* (in this order) .

With affine spaces arises the concept of affine transformations which preserves collinearity, parallelism, affine ratio and convexity. These transformations are particularly relevant in graphical systems as all operations on geometric objects are accomplished by composition of affine transformations.

Def.: 16: Let V_n, V'_n be linear spaces. A mapping $\beta: V_n \to V'_n$ is called *linear* if for each $\boldsymbol{u}, \boldsymbol{v} \in V_n$ and $a \in T$ satisfies:

i. $\beta(\boldsymbol{u} + \boldsymbol{v}) = \beta(\boldsymbol{u}) + \beta(\boldsymbol{v})$

ii. $\beta(a\boldsymbol{u}) = a\beta(\boldsymbol{u})$.

Def.: 17: Let A_n, A'_n be an affine spaces with associated linear spaces V_n, V'_n. The mapping $\alpha: A_n \to A'_n$ is called *affine mapping/affine transformation* if there exist an associated linear transformation $\beta: V_n \to V'_n$ such that for each $A \in A_n$ and $\boldsymbol{u} \in V_n$ equation:

$$\alpha(A + \boldsymbol{u}) = \alpha(A) + \beta(\boldsymbol{u}) \tag{6}$$

holds.
Note that the affine transformation is a combination of linear map and translation. Consequently the affine transformation can be expressed as:

$$B = \boldsymbol{M}\mathrm{A} + \mathbf{t} \tag{7}$$

where $\alpha(A) = \boldsymbol{M}A$ represents linear map, $\beta(\boldsymbol{u}) = \mathbf{t}$ associated map, $A \in A_n, B \in A'_n$, $\boldsymbol{M}$ is $n \times n$ matrix and $\boldsymbol{t} \in V'_n$. In the (Fig.6) are shown some examples of affine transformations and their matrices.

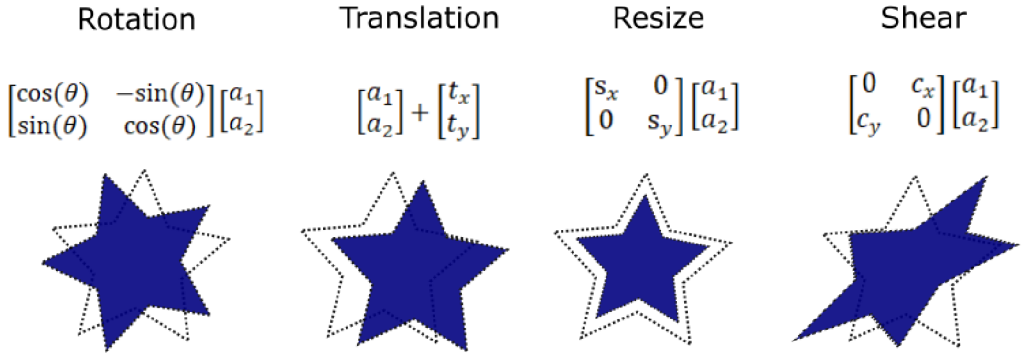


Fig. 6: Examples of affine transformations in 2D

Moreover it is also possible to represent affine transformations between *n*-dimensional spaces using single $(n + 1) \times (n + 1)$ *augmented matrix*:

$$\begin{bmatrix} B \\ 1 \end{bmatrix} = \begin{pmatrix} \boldsymbol{M} & \boldsymbol{t} \\ 0..0 & 1 \end{pmatrix} \begin{bmatrix} A \\ 1 \end{bmatrix}. \tag{8}$$

Such representation is admissible because of existence related projective space. Projective spaces will be discussed further in this chapter.

Euclidean space

Def.: 18: Let E_n be an affine space with associated unitary space V_n of ordered n-tuples over the field of real numbers $\mathbb{R}$ with a base $\mathcal{B} = \{\boldsymbol{e_1}, .., \boldsymbol{e_n}\}$ and with point $O \in E_n$. Then we call E_n the *Euclidean space* and the ordered (n+1)-tuple $\langle O, \boldsymbol{e_1}, .., \boldsymbol{e_n} \rangle$ the *Cartesian coordinate system.* Moreover for $X \in E_n$ where $X - O = x_1 e_1 + .. + x_n e_n$ we denote $X = [x_1, .., x_n]$ where the ordered coefficients $x_1, .., x_n$ are called *Cartesian coordinates* of point *X*.

Remark: Euclidean space is obviously equipped with scalar product: $\boldsymbol{u} * \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i$. In this case, metric is given as $|X, Y| = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$ and norm $\|\boldsymbol{u}\| = \sqrt{\sum_{i=1}^{n} u_i^2}$ for each $\boldsymbol{u}, \boldsymbol{v} \in V_n$ and $X, Y \in E_n$. In the following text, Euclidean space will be understood in this sense.

Note that in preceding text we dealt with linear spaces over some unspecified field equipped with a norm and a scalar product. Euclidean space is a special case of affine space where the underlying field is a field of real numbers but more importantly the associated space is unitary space with a uniquely defined metric and scalar product. Such definition allows us to compute angle θ between two vectors $\boldsymbol{u}, \boldsymbol{v} \in V_n$ as:

$$\theta = arccos\left(\frac{\boldsymbol{u} * \boldsymbol{v}}{\|\boldsymbol{u}\| \|\boldsymbol{v}\|}\right) \tag{9}$$

which is essential for geometry.

Analytic geometry

Geometry in general studies figures in the *n*-dimensional spaces of different types. As we suggested earlier, in the field of computer graphics we are particularly interested in analytic geometry as it studies figures, their properties and manipulation with them in the terms of algebraic equations and therefore in the form suitable for computer algorithms. For such a description it is necessary to have an associated space with a coordinate system where each object can be described as set of points of dimension *n*. The most common such space is the above described Euclidean space of dimension two or three (plane and solid geometry respectively) with the Cartesian coordinate system. By means of analytic geometry we are able for example to determine relative positions, intersections or apply affine transformations of points, lines and other geometrical objects defined by equation (or system of equations).

Further in the thesis all the described theory, formulas and implementation of back to front algorithm will assume Euclidean space in accordance with all the definitions and notions stated above.

3.2 Projections

In computer graphics we often visualize three dimensional data on two dimensional output device, usually the computer screen. In order to do that, we use various types of projection methods. The projection methods and their capability to preserve geometric properties of objects (lengths, angles, ratios, parallelism etc.) are addressed by also called projective geometry. In this subchapter we will first introduce projection space and homogeneous coordinates which are widely used in computer graphics for projection computations. After that we will introduce two main types of projections. Chapter is supported by knowledge from [4][6][7].

Projective space

The projective space and homogeneous coordinates is an extension of the Euclidean space and Cartesian coordinates which makes computation of geometric transformations easier.

Def.: 19: Let V_{n+1} be a linear space over the field of real numbers $\mathbb{R}$. Then the set of one dimensional subspaces of V_{n+1} is called *projective space* and denoted as P_n.

An alternative (and more intuitive) definition of projective space is based on the fact, that any one dimensional subspace of a vector space is just a multiplication of a nonzero vector $\lambda\boldsymbol{u}$ where $\boldsymbol{u} \in V_{n+1}$ and $\lambda \in \mathbb{R}$.

Def.: 20: Let V_{n+1} be a linear space over the field of real numbers $\mathbb{R}$ and $\sim$ an equivalence defined as:

$$\boldsymbol{u} \sim \boldsymbol{v} \leftrightarrow \boldsymbol{v} = \lambda\boldsymbol{u}$$

for any $\boldsymbol{u}, \boldsymbol{v} \in V_{n+1} - \{\boldsymbol{0}\}$ and some $\lambda \in \mathbb{R} - \{0\}$. Then we define the associated *projective space* P_n as a set of equivalence classes on $V_{n+1} : V_{n+1} - \{\boldsymbol{0}\}/\sim$.

Although the projective space is in literature introduced as in (Def.: 19), we will continue with the more elegant version (Def.: 20)

Def.: 21: Let P_n be a projective space and $[\boldsymbol{u}]_\sim \in P_n$. Then the equivalence class $\bar{X} = [\boldsymbol{u}]_\sim$ is called *geometric point* of projective space and when $\boldsymbol{u} = (u_1, .., u_{n+1})$ then the $u_i, i \in 1, .. (n+1)$ are called *homogeneous coordinates* of $\bar{X}$ and we denote $\bar{X} = [u_1, .., u_{n+1}]$.

Def.: 22: Let V_{n+1} be a linear space, P_n be associated projective space and $\boldsymbol{u}, \boldsymbol{v} \in V_{n+1} - \{\boldsymbol{0}\}$. We say, that two points $\bar{X}_1, \bar{X}_2 \in P_n$ where $\bar{X}_1 = [\boldsymbol{u}]_\sim, \bar{X}_2 = [\boldsymbol{v}]_\sim$ are *linearly independent* if the vectors $\boldsymbol{u}, \boldsymbol{v}$ are linearly independent. Moreover a pair of independent points defines *projective line* $\bar{p} = \{\bar{X}_1, \bar{X}_2\}$ and a triplet of independent points is called *projective plane* $\pi = \{\bar{X}_1, \bar{X}_2, \bar{X}_3\}$.

Note that the projections in general are transformations from *n* dimensional spaces to spaces with dimension lower than *n* and of various shapes. In the following we will

consider only the special case where each point of a 3D geometric object in Euclidean space is mapped to its image in the Euclidean projection plane.

For points in Euclidean space $X = [x_1, x_2, x_3] \in E_3$ there is isomorphism to the projective subspace EP_3 given as $\bar{X} = [\lambda x_1, \lambda x_2, \lambda x_3, \lambda]$ where $\lambda \in \mathbb{R} - \{0\}$. Moreover we can *normalize* homogeneous coordinates of points in EP_3 to the form $[x_1, x_2, x_3, 1]$ as: $[\lambda x_1, \lambda x_2, \lambda x_3, \lambda] \sim [x_1, x_2, x_3, 1]$.

Although we have excluded the possibility of $\lambda = 0$ the result of computing, for example intersection of two parallel lines, in the projective space can result in the fourth coordinate of the point being equal to zero. These points indicate the ideal points i.e. the points at infinity. See that such representation brings up a great advantage of projective spaces for computer graphics as infinity is represented by finite real coordinates.

Another noteworthy remark is that any transformation (including projections) in projective space can be achieved solely by matrix multiplication. At this point we can highlight that affine transformations represented by augmented matrix (as introduced in the previous subchapter) are achieved by affine space extension to projective space.

Perspective and parallel projections

Now that we have introduced the projective space, we may move forward to the projections. Having the 3D object (set of points) in Euclidean space and (Euclidean) projective plane, the projections are defined as intersection of lines passing through the points of the object with the projection plane. Such lines are called *projectors* and by the arrangement of the projectors we distinguish two major types of projections: *parallel* and *perspective*.

Def.: 23: Let us have a projection plane $\pi \in EP_3$ defined by its normal vector $\boldsymbol{n} = (n_1, n_2, n_3, n_4)$ and a *viewpoint* $V \in EP_3$ but $V \notin \pi$. Let $P, P' \in EP_3$ and $V \neq P$. Then the transformation $\rho: P \to P'$ which maps P onto P'such that $P' \in \pi$ and $P' \in p_{VP}$, where p_{VP} is projector line defined by P and V, is called *projection.* Moreover if P is ideal point we say ρ is *parallel projection*. Otherwise ρ is called *perspective projection.*

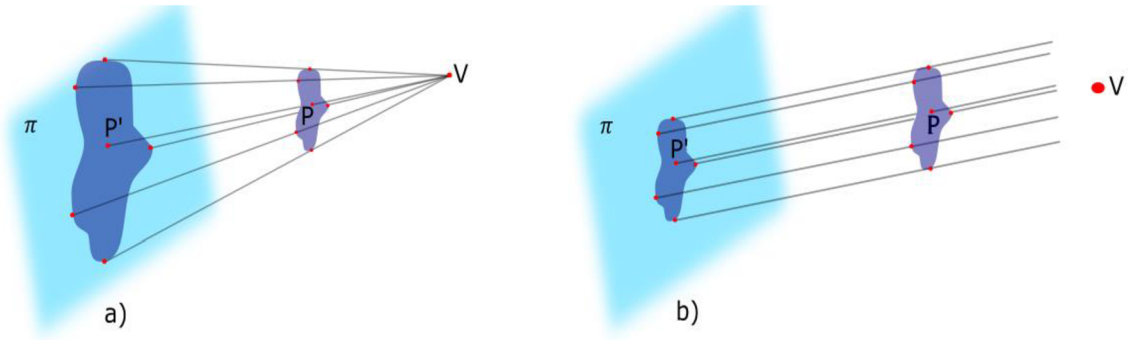


Fig. 7: a) Perspective projection b) Parallel projection

The general projection with viewpoint V and projection plane π as defined above can be expressed by $4x4$ *projection matrix*:

$$M = \boldsymbol{n}^T V - (\boldsymbol{n}V)\boldsymbol{I} \tag{10}$$

where $\boldsymbol{I}$ is $4x4$ identity matrix and $\boldsymbol{n}, V$ are row vectors.
In literature we can often find further classification of the above projections (Fig.8).

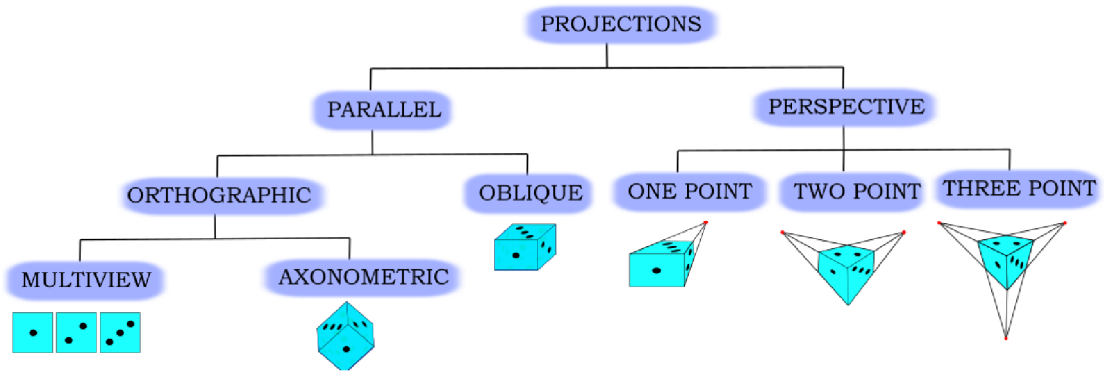


Fig. 8: Classification of Perspective and parallel projections

Considering the perspective projection, each line not parallel to the projective plane is projected converging to some *vanishing point* which represents projection of an ideal point. In addition if a line is parallel to some of the coordinate axes it converges to *principal vanishing point.* On the projective plane there can be up to three principal vanishing points depending on the position of the projective plane with respect to coordinate axes. In particular the parallel is called one/two/three point if the projective plane intersects one, two or three coordinate axes respectively.

In the case of parallel projection we distinguish *orthographic* and *oblique* projections. Orthographic projection is typical by the projectors being orthogonal to the projective plane. Moreover if the projective plane is parallel to two of coordinate axes then we speak about *multiview projection*. Otherwise the orthographic projection is *axonometric*. On the other hand oblique projection has no right angle between projectors and projective plane.

Now that we have introduced both major types of projections it is appropriate to discuss their suitability. Because in the perspective projection the size of the projected objects varies with distance from the viewpoint, it is being used widely in computer graphics as it reflects human visual perception and therefore creates more a realistic 3D impression. On the other hand in general it does not preserve parallelism nor angles. On the contrary the parallel projection does not form a realistic view of objects but preserves parallelism, relative proportions and in some cases even angles and distances. Therefore parallel projection is more suitable for scientific purposes when we are interested in objective measurement of data.

4 BACK-TO-FRONT ALGORITHM

This chapter is devoted to the introduction of a back-to front algorithm. In the first short subchapter we will introduce direct volume rendering and how are the volumetric data represented in the computer. As the back-to-front algorithm is based on the physical model of light propagation we will continue with definition of basic radiometric quantities needed for derivation of the volumetric integral which exactly describes the propagation of the light through volume. In the last subchapter we will discretize the volume rendering integral and obtain the back-to-front algorithm.

4.1 Direct volume rendering and volumetric data representation

This subchapter is based on [6][8][9]. By volume rendering we denote a set of methods invented to visualize 3D data on 2D images by simulating light transport across the volume. These procedures are used for imaging data from computed tomography (CT), magnetic resonance imaging (MRI) or as in our case confocal microscopy. Volume rendering methods are generally used when the visualized data contain inner structures. The cost of precision and detail in volume rendering are large datasets that need to be processed. On the other hand, by using less space demanding surface rendering methods like triangular surface meshes or boundary representation of objects, on such data, we would have lost a significant amount of information.

In volume rendering it is usually worked with a set D of physical data (velocity, density, temperature…) assigned to discrete points in space $[x, y, z, v] = V \epsilon D$ where v is the value of the observed quantity. The value v may be binary (observed phenomenon is or is not present) or single valued (measured value of observed phenomenon). Note that v may be of higher dimension if there are more types of information observed at the point. In some applications it is advantageous to include also time component so that $V = [x, y, z, t, v]$ and observe the dynamic change of the data over a time period.

The assignment of the observed data to the points in space can be either completely random or more often arranged to regular or irregular grid structures. The basic structures for volumetric data are shown in (Fig.9).

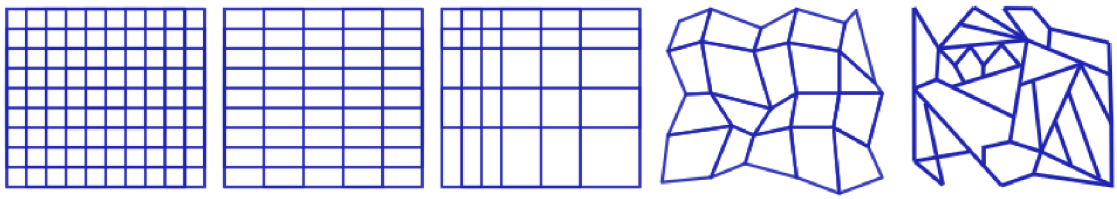


Fig. 9: From left: Cartesian, rectangular, rectilinear, structured, unstructured grid

The first three data arrangements in the figure are based on axis aligned rectangles, more precisely: Cartesian (isotropic) grid is made of cubical cells, rectangular grid is

composed of homogenous rectangles and rectilinear grid is axis-aligned but the spacing along axis is arbitrary. The structured grid is achieved by application of non-linear transformation to a rectangular grid. In the case of unstructured the grid cells don't have to be implicitly connected to their neighbours. Note that depending on the type of represented data, the grids have arbitrary dimension.

Since our data samples are raster images with square pixels with constant distance between each slice, it is convenient to organize them into 3D rectangular grid, where v is triplet $R, G, B \in \langle 0{,}255 \rangle$ of colour components of each pixel. The basic element of a regular volumetric grid is called *voxel* (volume element) which is a 3D analogy for 2D pixel. In literature there are two possibilities how to estimate the value of observed phenomena using voxels. Either the assigned value is constant in the whole volume of voxel or is considered valid in the voxel centre. In the latter case any value at a point that is not the centre of the voxel is interpolated from two closest voxel centres. Further in the text we will use the first voxel interpretation.

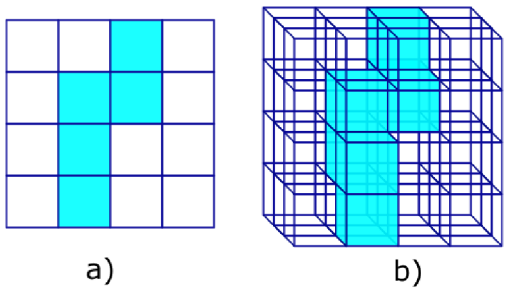


Fig. 10: a) Binary pixel grid, b) binary voxel grid.

4.2 Basics of lighting theory

This section we will deal with the theory of light transmission which is crucial to realistic rendering of objects. Information in this part is based on [6][10][11]. Even in real life we are able to see objects because of the light falling to our retina. The discipline in physics dealing with light and its properties is called radiometry and discipline dealing with human perception of light is photometry. These two topics are quite close. Actually the radiometric quantities are convertible to photometric and vice versa. In this chapter we will introduce basic terms of these disciplines in order to be able to derive volume rendering integral further in the text.

The light has dual nature, it behaves as waves as well as particles. In computer graphics we mostly neglect the wave nature, polarization and dispersion and we consider the light as rectilinear rays of infinite velocity which are not affected by gravity or electromagnetic fields. Basic element of light is called photon which represents the smallest emittable energy of light of wavelength $\lambda[nm]$. The energy of a photon q_λ [Joule] is determined by equation:

$$q_\lambda = \frac{hc}{\lambda} \tag{11}$$

where $h \approx 6.63 \cdot 10^{-34}$ is Planck constant and $c = 299\,792\,458 ms^{-1}$ is the velocity of the light in vacuum.

Radiation universally indicates propagation of energy through space. *Radiant energy Q[Joule]* refers to an amount of energy of all photons in specific area:

$$Q = \int_0^\infty n_\lambda q_\lambda d\lambda \tag{12}$$

where n_λ represents number of photons of wavelength λ. In general physics an integral for the whole wavelength spectrum or for the visible spectrum is used. In computer graphics we are typically using RGB model (or in that matter any colour model with finite number of wavelengths) so we are only interested in the three corresponding scalar wavelengths. For such case we introduce *spectral radiant energy* $Q_\lambda[Joule \cdot nm^{-1}]$ representing radiant energy per unit wavelength

$$Q_\lambda = \frac{dQ}{d\lambda}. \tag{13}$$

In literature regarding computer graphics the subscript λ is often omitted as the interest in just three basic colours is a matter of course. Further in the text we will omit this subscript as well keeping in mind that any mentioned quantities are meant for specific wavelength and for the general case we only need to integrate a given equation over the desired spectrum.

The time rate of spectral radiant energy (received or emitted) is defined as *spectral radiant flux (power)* Φ $[J \cdot s^{-1} \cdot nm^{-1} = Watt \cdot nm^{-1}]$

$$\Phi = \frac{dQ}{dt}. \tag{14}$$

The spectral radiant flux per unit of area in a surface that is incident on (density of radiant flux) is called s*pectral irradiance E* $[Watt \cdot nm^{-1}m^{-2}]$:

$$E = \frac{d\Phi_i}{dA}. \tag{15}$$

where A represents illuminated surface and Φ_i is incident radiant flux. The irradiance leaving considered surface (either emitted or reflected) is called *radiant excitance (radiosity) M*

$$M = \frac{d\Phi_o}{dA} \tag{16}$$

where subscript *o* in Φ_o means outgoing. In most computer graphic related literature the irradiance and radiosity are not distinguished as there is no difference in their equations from the mathematical point of view, and both terms are called irradiance. Further in the thesis, we will as well, merge these two terms as distinction is not important for us. Note that irradiance is a function of position on a surface therefore in general case, when we can't assume that the radiant flux is constant over a considered portion of surface, we have to consider radiant flux from/to any direction in hemispherical solid angle (Fig.11a)).

Sometimes it may be more suitable to consider the density of radiant flux in solid angle rather than on the unit of area. Then we are speaking about *radiant intensity* $I\,[Watt \cdot nm^{-1}sr^{-1}]$ defined as:

$$I = \frac{d\Phi}{d\omega} \tag{17}$$

where ω is solid angle in specific direction defined by φ, ψ (Fig.11b)).This function of direction and point is useful, when we deal with point sources of radiation.

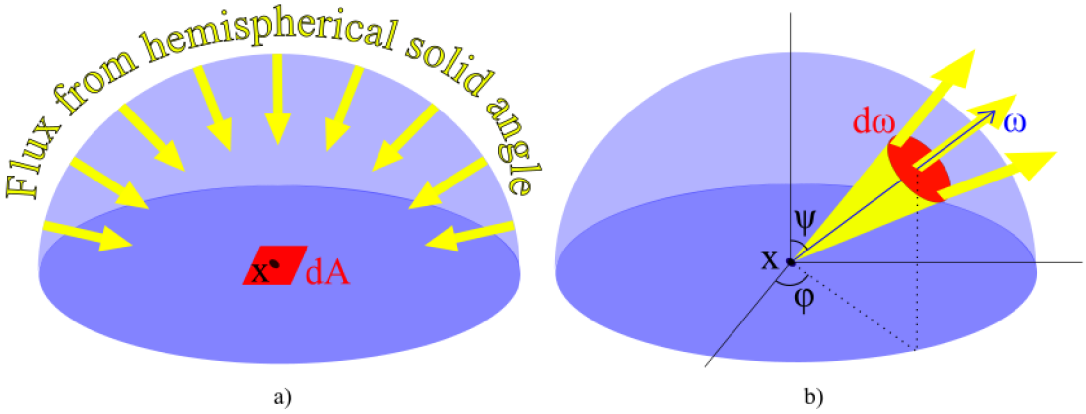


Fig. 11: a) Irradiance- radiant flux per unit area *dA* incident to a point *x* at object surface. b) Radiant intensity- radiant flux per element of solid angle *dω* defined by point *x* and direction ω.

Finally we are getting to the most important radiometric quantity called *radiance* $L[Watt \cdot nm^{-1}sr^{-1}m^{-2}]$. Radiance is a function of position (point), area and direction. More precisely it states absorbed or emitted power on unitary solid angle per unitary area projected perpendicularly to the given direction. The defining equation is:

$$L = \frac{\partial^2 \Phi}{cos\psi \partial A \partial \omega} \tag{18}$$

where $cos\psi\partial A$ represents projected area of unitary area ∂A to the surface containing the point of interest (this follows from Lambert law), $\partial\omega$ is element of solid angle in specified direction and ψ is the angle between the given direction and normal to the surface.

The importance of radiance lies in the fact that what we described in terms of physic is actually what human eye, cameras etc. detect as color and what we have stored in pixels of our raster image dataset. Moreover from radiance equation (18) we may derive all the radiometric quantities described above.

In most algorithms regarding computer graphics the radiance is considered constant on the whole trajectory (ray). This is not our case though. We want to render our data so they appear semi-transparent. We may achieve this by considering the 3D data matrix as participating media which will be elaborated in the next subchapter.

4.3 Volume rendering integral

Most of the image order methods are based on approximation of *volume rendering integral* (VRI) which considers volume as a participating medium. Participating medium is a cloud consisting of small particles where each particle can absorb, emit or scatter light (for example water, fog or smoke are rendered as participating media) (Fig. 12). In our case we will consider low-albedo particles meaning that reflectivity of the particles is negligible and the light passing through volume is considered to be a single ray (hence the name "ray casting methods"). The idea of volume rendering integral was first described by Blinn [12] and formally derived by Max in [13]. This section is cited from [6][8][13].

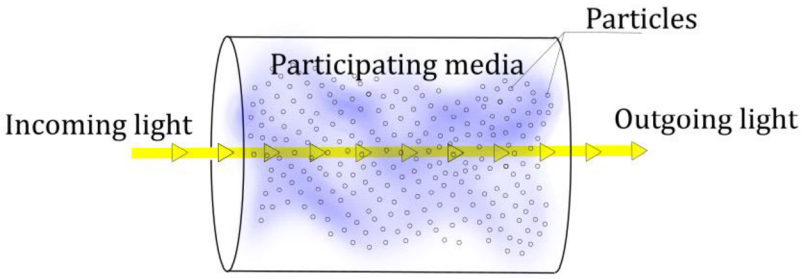


Fig. 12: Participating medium

VRI is in general integrated over visible spectrum of wavelengths. As we mentioned in the previous subchapter while dealing with raster data we are interested only in wavelengths of red, green and blue colour. In the following we will derive the VRI for one scalar wavelength with validity for each particular colour. In the next subsections we will consider the absorption and emission components of the equation separately and later we will combine them to form VRI.

4.3.1 Absorption of the light

Absorption only participating medium consists of perfectly black particles which do not emit any light. We will consider a very thin cylindrical layer of volume with area of base B and thickness Δs (Fig.13). In the volume we consider medium with spherical particles of identical radius r and with density of particles per unit volume ρ. Let the light flow through the layer perpendicularly to the base. Then each particle casts a shadow (projection) of area $A = \pi r^2$ to the base B. Consequently the volume contains

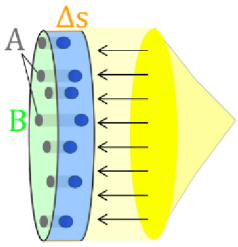


Fig. 13: Absorption of light

$N = \rho \Delta s B$ particles. If we take Δs small enough so we can suppose that the particles are not overlapping each other, then the area of B overshadowed by the particles is approximately $NA = \rho AB\Delta s$. Relatively speaking the occluded fraction of the light flow is $NA/B = \rho A \Delta s$. Taking $\Delta s \to 0$ the probability of overlapping particles tends to zero and we get differential equation:

$$\frac{dI}{ds} = -\rho(s)AI(s) = -\tau(s)I(s) \tag{19}$$

where *s* is a length parameter, *I(s)* is light intensity at distance *s*. Coefficient $\tau(s) = \rho(s)A$ is called *extinction coefficient* and expresses the rate of decline of light intensity along a ray. Solution to (19) is

$$I(s) = I_o e^{-\int_0^s \tau(t)dt} \tag{20}$$

where I_0 is intensity of light at distance *s=0* (at point where the ray intersects the volume) (Fig.14) and

$$T(s) = e^{-\int_0^s \tau(t)dt} \tag{21}$$

is called *accumulated transparency* or simply *transparency*. Complementary element to transparency on voxel of length *l* is *opacity*

$$\alpha = 1 - T(l) = 1 - e^{-\int_0^l \tau(t)dt}. \tag{22}$$

Moreover if the extinction coefficient is constant in voxel $\tau(l) = \tau$ then we can also express opacity by Taylor expansion:

$$\alpha = 1 - e^{-\tau l} = \tau l - \frac{(\tau l)^2}{2} + \cdots \tag{23}$$

For small voxels we may approximate (23) as $min(1, \tau l)$.

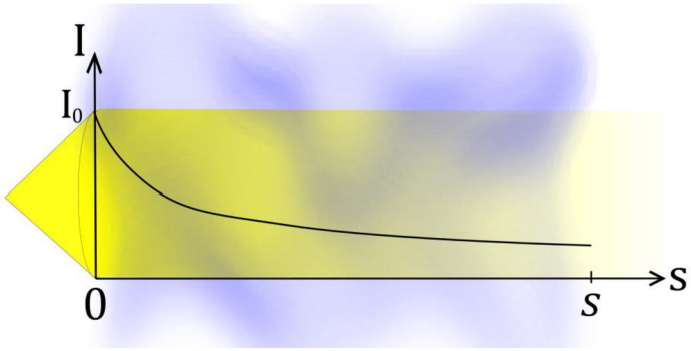


Fig. 14: Intensity function $I(s)$

A mapping that assigns value to scalar function *f(X)* is called *transfer function*. Depending on the chosen transfer function we distinguish different optical models. For example the simplest optical model is *thresholding,* given by

$$\tau = \begin{cases} \infty & if\ f(X) > K \\ 0 & if\ f(X) \leq K \end{cases}$$

where K is a constant threshold.

4.3.2 Emission of the light

Now we consider a situation when each particle of the participating medium adds some portion of intensity to the light flow. We will use the same cylindrical layer model as in the derivation of the absorption component of VRI. For elimination of the absorption we will now consider the particles to be completely transparent so no light intensity is lost traversing through volume.

Let C be an intensity emitted by each particle per unit of area. Then similarly to the absorption approach we can deduce that intensity added to particle projection to the base is approximately $C\rho AB\Delta s$ and for unit area we get intensity $C\rho A\Delta s$. By sending $\Delta s \to 0$ we get equation

$$\frac{dI}{ds} = C(s)\rho(s)A = C(s)\tau(s) = g(s) \tag{24}$$

where $g(s)$ is called the *source term.* Solution to the equation (24) is

$$I(s) = I_0 + \int_0^s g(t)\,dt \tag{25}$$

4.3.3 Emission and absorption combined

For a more realistic view we have to consider particles as intensity emitters as well as attenuation source. Corresponding relation is obtained by combination of equations (19) and (24) :

$$\frac{dI}{ds} = g(s) - \tau(s)I(s) \tag{26}$$

And the solution to the equation (26) for $I(s)$ is:

$$I(s) = I_o e^{-\int_0^s \tau(t)dt} + \int_0^s g(s')e^{-\int_{s'}^s \tau(t)dt}ds' \tag{27}$$

Or alternatively for $I(0)$

$$I(0) = I_s e^{-\int_s^0 \tau(t)dt} + \int_s^0 g(s')e^{-\int_s^{s'} \tau(t)dt}ds' \tag{28}$$

where 0 represents edge of the volume and s a point in the eye (or image frame). As we can see the first term in (27) is the light intensity of background multiplied by transparency T(s) (21) and the second term is intensity contribution of each particle on the ray multiplied by transparency between 0 and s:

$$T'(s) = e^{-\int_{s'}^s \tau(t)dt} \tag{29}$$

And we can rewrite equation (27) as

$$I(s) = I_o T(s) + \int_0^s g(s')T'(s')ds'. \tag{30}$$

The equation (30) is known as *volume rendering integral*. Note that for the simplicity we have set the length interval as $\langle 0, s\rangle$ but it can be written more generally as $\langle s_0, s_1\rangle$.

4.4 Back-to-front algorithm

The volume rendering integral cannot be computed analytically. Therefore we have to at least approximate it. We employ discrete Riemann sum over the casted ray with discrete samples at distances spaced apart by Δs. By such discretization we get

$$I(s) = I_o e^{-\int_0^s \tau(t)dt} + \int_0^s g(s') e^{-\int_0^{s'} \tau(t)dt} ds'$$

$$\approx I_o e^{-\sum_{i=1}^{n} \tau(i\Delta s)\Delta s} + \sum_{i=1}^{n} g(i\Delta s) e^{-\sum_{j=i+1}^{n} \tau(j\Delta s)\Delta s}$$

$$= I_o \prod_{i=1}^{n} e^{-\tau(i\Delta s)\Delta s} + \sum_{i=1}^{n} g(i\Delta s) \prod_{j=i+1}^{n} e^{-\tau(j\Delta s)\Delta s}$$

where $n = |s_1 - s_0|/\Delta s$. Using equation (21) we have transparency $T(s) \approx t(i\Delta s) = t_i$ and $g(i\Delta s) = g_i$ hence

$$I(s_1) = I_o \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j. \tag{31}$$

Equation (31) is currently known as back-to-front visualization method. However, the phrase "back-to-front" does not describe data processing itself but direction of data processing only. Data in this direction may be processed in different ways. In the following we will introduce basic options for these different solutions and output image generation.

Transfer functions

Note that having a scalar field, if we combine all the information stored, some of the properties of imaged data can get attenuated. For example if we scan a cell and apply the back to front algorithm the nucleus will be partially overshadowed by the cell membrane. Although this approach produces fairly realistic output, sometimes we may prefer to highlight particular parts of the imaged object and inhibit or even hide others. For a such purpose it is possible to apply transfer functions.

Transfer functions in general are mappings from a 3D scalar field to optical properties. For example in case of magnetic resonance imaging it is of the form $f: I \to R, G, B, \alpha$ where I is intensity, R, G, B represent colour and α is opacity (Fig.15). As we will render data from confocal microscope we have the colours assigned so the mapping is rather $f: R, G, B \to \alpha$.

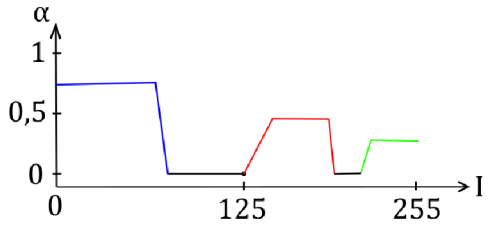


Fig. 15: An example of a transfer function for a grey scale set of images where the color of the graph determines color for pixels with given intensity.

In the implementation of the back to front algorithm we will consider:

- *Constant transfer function* with $\alpha = constant$ for all values of RGB intensities
- *Piecewise constant transfer function* with $f(U_i) = \alpha_i$ where U_i is subset of the $RxGxB$ color space.

In our implementation the transfer function is user defined piecewise constant. Note that it is possible to define the function differently. There are even some semi-automated methods for their determination based for example on dataset histograms, frequency distributions, gradient methods etc. The topic of transfer functions is extensive and far beyond the limits of this thesis, but a brief summary in case of interest can be found for example in [19].

Back-to-front methods

Along with transfer functions, which assign opacity to the individual voxels and the basic back to front compositing equation (31) we have alternative projective methods for generation final image out of scalar field:

- *Maximal intensity projection* where the voxel with highest intensity on the ray is taken as an output.
- *Average intensity projection* computes output pixel as average of all voxels intersected by the ray.
- *Constant transparency method* iterates over voxels on the ray and each step *i* computes:

$$I_{new} = TI_{previous} + g_i(1 - T) \tag{32}$$

  where the T is constant transparency, g_i is intensity of processed voxel and (1-T) represents constant opacity. Term $g_i(1 - T)$ models emission of processed voxel.
- *Volume Rendering integral method* iterates over voxels on the ray and each step *i* computes an approximation to volume rendering integral as:

$$I_{new} = t_i I_{previous} + g_i \tag{33}$$

  t_i is assigned transparency given by transfer function and g_i represents intensity of processed voxel which is considered to be its emission.

5 SOFTWARE SOLUTION

In the following chapter the software implementation of the scalar field imaging methods will be described. The algorithm is implemented in c++ language within Visual Studio 2019 integrated development environment. Consequently the following text will be written with respect to c++17 standard. In the attachment of this thesis the implementation can be found in two versions. First version is an algorithm written as a console application giving faster results. Second version is the algorithm running in intuitive graphical user interface for a price of noticeably increased computing time. Note that in the two attached versions there is no other difference, than graphical appearance and way of entering the variables. In this chapter we will introduce the algorithm step by step as it is implemented in attached files.

The idea of software application is founded on a projection method combined with a back to front algorithm described in the preceding chapters. More precisely we are projecting the data matrix onto a projective plane (output window) where the final projected value is a function of voxels hit by the projection line (Fig.16) .

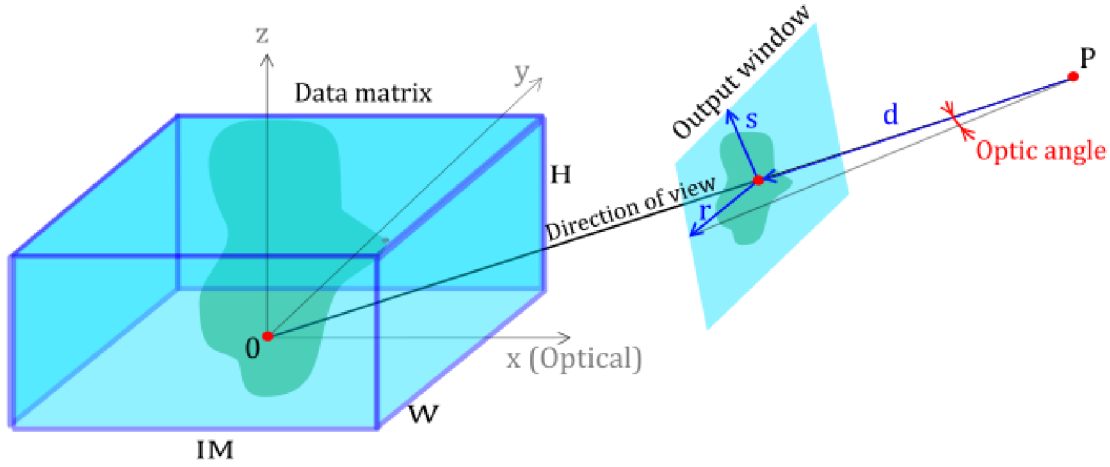


Fig. 16: The main idea of the implementation

5.1 Loading data from file

The first step is to load data that are going to be visualized. Recall that we have data acquired by confocal microscope i.e. set of images. In the application the set of images is supposed to be in the BMP file format and with equal resolution. Moreover each image name needs to end with a decimal number starting from zero and indicating its position in the scanned specimen. From such a set the algorithm reads each image incrementally by its number, pixel by pixel. The read scalar values (R, G, B) are then represented by a 3D matrix (Fig. 17). For further computations we also create integer array variable representing dimensions of the created matrix

dim=[number of loaded images, width, height].

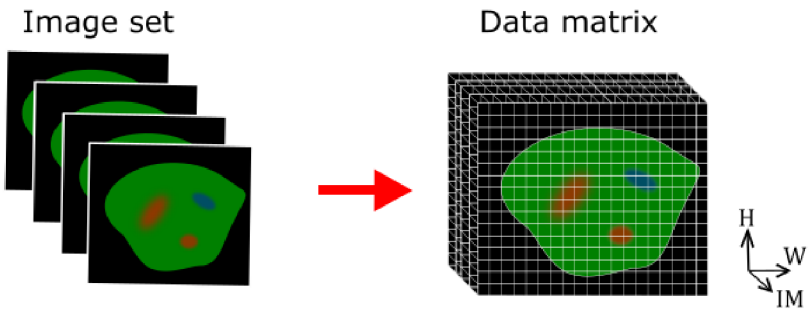


Fig. 17: Data representation

In the following text we will denote the dimensions as *[IM, W, H]*. In the console implementation the path to the images has to be defined manually in the code with the slash direction as backslash in the strings is reserved for special pattern characters.

5.2 Input variables

In this part we will introduce the user defined variables. Thoughtful alteration of them will more or less significantly influence the output and can produce relevant computation results. We can split the variables into two categories: *projective variables* for definition of geometrical properties of the view (most of them can be seen on Fig.16) and *graphical variables* representing coefficients for computation resulting colour for output pixels. In the console version of the algorithm the input variables can be set on the lines *1071-1088*.

a) Projective variables

- *Camera position* $P = [P_x, P_y, P_z]$ is an array of doubles representing a point from which we are looking at a data matrix. Due to further computations $P_x, P_y, P_z \neq 0{,}0$. Convenient choice of values might be e.g. consistent with:
$$\|P\| > \max\{\mathrm{IM} + (\mathrm{IM} - 1) * optDist;\ \mathrm{W}; \mathrm{H}\} + dist$$
- Distance *dist* representing the distance between the camera position and projective plane. Due to further computations $dist > 0{,}0$.
- In the implementation we can decide about the projection method using boolean variable *perspective*. If the variable is set to *true* the output image is computed as a result of perspective projection. *False* value sets projection method to parallel. Depending on choice of projection we need one of the following variables in order to determine size of output window:
    - If we are imaging by perspective projection, we need a double variable *angle* that determines the angle under which we want to see the data matrix (optic angle) in degrees. Due to further computations $angle \in (0; 180) - \{90\}$

- In the means of parallel projection double variable *mag* is coefficient by which we magnify the output window or alternatively it is the inverse of magnification of data matrix. Clearly $mag \neq 0{,}0$.

- As we have discussed in the chapter about confocal microscopy the resolution in the optical axis is typically worse than the resolution in focal plane. If we neglected the fact we would get the resulting image disproportionately flatter in the optical axis. In order to get realistic result we define *optDist* integral variable which determines the optical distance between the individual images. In the terms of application we virtually expand the data by inserting spaces of size *optDist* in between the actual images (Fig.18). The optical distance is typically given by the microscope. The dimension of expanded matrix in optical axis is given by: $\mathrm{IM} + (\mathrm{IM} - 1) * optDist$. Note that we haven't excluded the case of $optDist = 0$ as for some cases it can provide the same output at less computational cost as any other value.

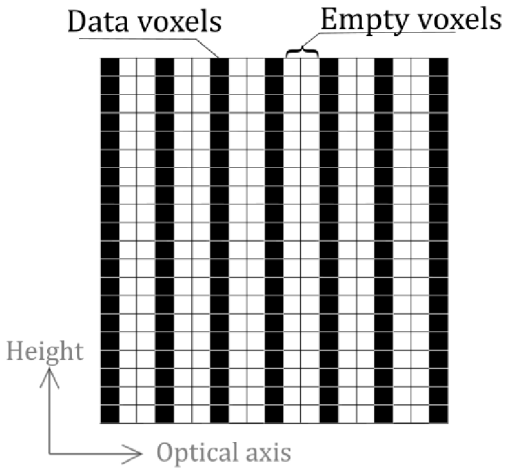


Fig. 18: Optical distance representation in data matrix

- Closely related to the optical distance is a boolean variable called *spaces*. It determines how to cope with empty space generated by $optDist \neq 0$. If the value is set to *false* it approximates the empty space with weighted average of the right and left nearest data voxels. It is an especially convenient setting for perspective projection as it creates a visually more realistic impression of the data without intensity jumps due to the gaps. On the other hand the variable set to *true* leaves the space empty, which is a more suitable setting for performing measurements. Moreover for the average and maximal intensity method the spaces are automatically assumed empty as computing approximations would be redundant or even distorting.

b) Graphical variables

- *BTFmethod* is an integral variable representing the method applied on the data.

  0: Maximal intensity projection method,
  1: average projection method,
  2: constant transparency method,

3: volume rendering interval method with constant transfer function,
4: volume rendering interval method with user defined piecewise constant transfer function.

- *Transparency* is integral value in the interval <0;100>, which in the case of the method with constant transparency defines the coefficient *T* in the equation (32), in the case of volume rendering integral method coefficient t_i in equation (33).
- For volume rendering integral method with user defined transfer function there is:
  - Vector called *TF* of seven integer arrays representing the upper and lower bound for each of *R,G,B* intensities and assigned transparency. These intervals should not overlap and obviously the lower bounds need to be lesser or equal to upper bounds.
  - Opacity constant called *op* which is used for multiplication of each voxel intensity value in the given interval.
  - Default opacity *dop* which is used as opacity constant outside of the defined interval.
- *Threshold* is integral variable from interval <0;254> determining whether to include a given voxel in the computation of output pixel intensity. More specifically if the voxel doesn't exceed a given threshold by any value of intensity (R,G,B) it is skipped. Thoughtful choice of *threshold* can significantly accelerate computation or even filter away some noise.
- *Brightness* is integral coefficient from <-255,255> that is added to each intensity (R,G,B) of output pixels up to intensity = 255 or down to intensity=0 in order to get brighter result.
- *Contrast* is double variable multiplying each intensity (R,G,B) of output pixels up to intensity = 255 in order to get a more contrasting result.

5.3 Generation of output window

As we have loaded the data and acquired the user defined variables it is possible to continue within the algorithm. We will proceed by definition of the output data window. Although the definition differs by the projection method used, the variation in fact manifests only in one parameter and therefore can be implemented as a single function.

The output window position and rotation in the coordinate system will be defined by vectors $\boldsymbol{r} = (r_0, r_1, r_2)$ and $\boldsymbol{s} = (s_0, s_1, s_2)$. The vectors can be determined by posing six requirements:

1. The distance of the output window from the camera position is equal to the *dist* variable: Taking the direction of view $\boldsymbol{d} = \boldsymbol{O} - P$ and making its length equal to *dist:* $\boldsymbol{d} = \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|} dist$, then we acquire the centre of the output window $O' = P + \boldsymbol{d}$.
2. Vector $\boldsymbol{r}$ is parallel to the *xy* plane. Therefore $r_2 = 0$.
3. Vector $\boldsymbol{r}$ is perpendicular to the direction of view $\boldsymbol{d}$*:*

$$\boldsymbol{r} * \boldsymbol{d} = r_0 d_0 + r_1 d_1 + r_2 d_2 = 0$$

and with $r_2 = 0.0$ we acquire:

$$r_1 = \frac{-r_0 d_0}{d_1}$$

4. In the case of perspective projection the last restriction is given by the user defined optic angle transformed to radians and for parallel projection it is given by the *mag* variable. In fact both given variables determine the length of the vector ***r***:

$$\|\boldsymbol{r}\| = \begin{cases} c = \|\boldsymbol{d}\| \tan\left(\frac{angle}{2}\right) & perspective\ projection \\ c = mag * width = mag * \mathrm{W} & parallel\ projection \end{cases}.$$

Then:

$$\|\boldsymbol{r}\| = \sqrt{r_0^2 + r_1^2 + r_2^2} = \sqrt{r_0^2 + \left(\frac{-r_0 d_0}{d_1}\right)^2} = c$$

from which we can compute:

$$r_0 = \frac{c}{2\sqrt{1 + \frac{d_0^2}{d_1^2}}}$$

and we acquired all coordinates of vector ***r***.

5. Vector ***s*** is perpendicular to both ***d*** and ***r*** thus we can compute it as cross product of ***d*** and ***r*** and acquire:

$$\boldsymbol{s}' = (-r_1 d_2, r_0 d_2, r_1 d_0 - r_0 d_1).$$

6. The output window size proportionally corresponds to the input/output image size: $\frac{\|r\|}{\|s\|} = \frac{W}{H}$. This fulfils the ***s*** computed as:

$$\boldsymbol{s} = \frac{\frac{W}{H}\|r\|\boldsymbol{s}'}{\|\boldsymbol{s}'\|}.$$

Note that our task will be to go through the output window pixel by pixel and shoot a ray through it in order to compute the projection of the data matrix. Previously in the theoretical part of the thesis we have chosen the notation, where the voxel (or now in the case of output window pixel) has the constant value in its whole volume (area). Bearing that in mind we can represent each pixel by a point. Setting:

$$Q_{00} = P + \boldsymbol{d} + \boldsymbol{s} + \boldsymbol{r}$$

as starting point and

$$\Delta\boldsymbol{i} = \frac{2\boldsymbol{r}}{\mathrm{W}}$$

$$\Delta\boldsymbol{j} = \frac{2\boldsymbol{s}}{\mathrm{H}}$$

vectors representing distance between the nearest two pixels in the directions of ***r*** and ***s***. Then value of each output pixel can be estimated as:

$$Q_{ij} = Q_{00} - \Delta\boldsymbol{i} * i - \Delta\boldsymbol{j} * j$$

where $i \in < 0; \mathrm{W} - 1 >$ and $j \in < 0; \mathrm{H} - 1 >$ are integers.

This part of the algorithm is implemented as function outputWindowGen() which returns (saves to referenced variable) point Q_{00} and vectors $\Delta\boldsymbol{i}, \Delta\boldsymbol{j}, \boldsymbol{d}$.

5.4 AABB ray intersection

In this subchapter we will look into the algorithm for finding the intersection of the projective line with the data matrix (box). Naive approach would be to find intersections of the line with all six planes forming the block separately. Then we would check whether the intersection point lies within the rectangle boundaries in the plane. Although functional, this approach has a lot of special cases that have to be handled which make the algorithm relatively slow. Since ray-box intersection is very widely used in computer graphics, several much more effective and robust algorithms had been developed. In the software solution there is implemented the axis aligned bounding box (AABB) algorithm introduced in [14]. This subchapter will be based on the information form [6][14].

The algorithm is based on comparison of parameters $t \in (-\infty, \infty)$ of the parametric line equations:

$$\begin{aligned} x &= P_x + projLin_x t \\ y &= P_y + projLin_y t \\ z &= P_z + projLin_z t \end{aligned} \tag{34}$$

where the point P is the position of the camera and vector ***projLin*** direction of view. The box is defined by two points on its space diagonal. These points are called *lower and upper bound* where the lower bound has the minimal and the upper bound maximal coordinates on the axes (Fig.19). In our case we also have consider the distance

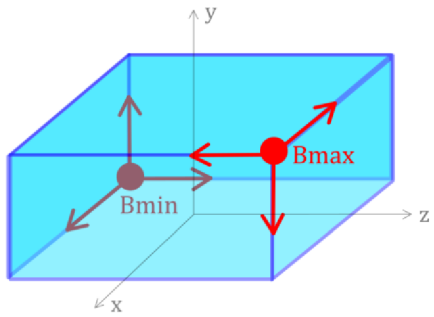


Fig. 19: Upper and lower bounds representing a box

in-between the samples in the optical axis *optDist* so our bounds are determined as:

$$\begin{aligned} B_{min} &= \left[\frac{-(\mathrm{IM} + optDist(\mathrm{IM}-1))}{2}, \frac{-\mathrm{W}}{2}, \frac{-\mathrm{H}}{2}\right] \\ B_{max} &= \left[\frac{\mathrm{IM} + optDist(\mathrm{IM}-1)}{2}, \frac{\mathrm{W}}{2}, \frac{\mathrm{H}}{2}\right]. \end{aligned} \tag{35}$$

Since the box is axis aligned, each coordinate of the bounds represents one of the six boundary planes of the box and we can acquire the intersections of them with the viewing ray as:

$$B_{ij} = P_j + projLin_j t_{ij} \tag{36}$$

where the meaning of subscripts is: $i \in \{min, max\}$ and $j \in \{x, y, z\}$. From the above equation we express t_{ij}:

$$t_{ij} = \left(\frac{B_{ij} - P_j}{projLin_j}\right). \tag{37}$$

Now that we have the intersections with all six planes we want to determine which two of them (if any) are placed on the box. First we will explain the logic behind the parameters comparison on the 2D space with a viewing ray increasing in both axes. Next we will generalise the algorithm to all directions of the ray and finally we will make a simple extension onto 3D space.

So the first question to be asked is if the viewing ray intersects the box at all. From the (Fig. 20a)) we can see, that if either $t_{min,x} > t_{max,y}$ or $t_{min,y} > t_{max,x}$ the ray doesn't intersect the box. In that case we terminate the algorithm and we assign (0,0,0) value to the related pixel on the output screen. Otherwise, the ray intersects the box and we need to figure out which two of the four points lie on it. Looking at (Fig. 19b) we can see that any ray intersecting the box intersects first the lines defined by B_{min} and then the lines of B_{max}, whilst the points lying on the box are those defined by $t_{max} = min(t_{max,x}, t_{max,y})$ and $t_{min} = max(t_{min,x}, t_{min,y})$.

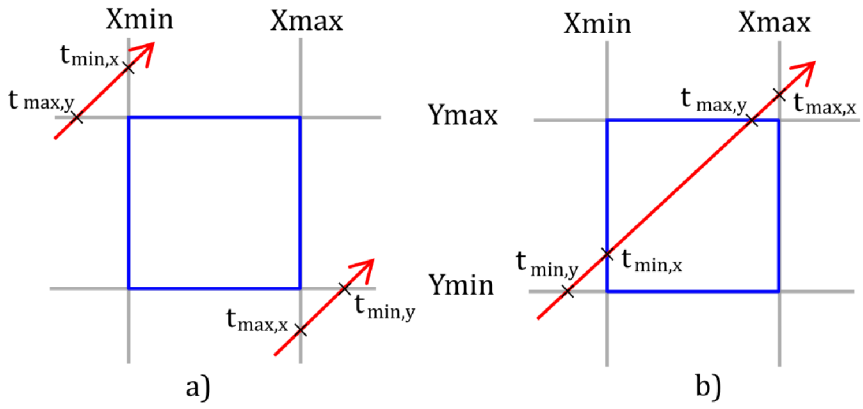


Fig. 20: a) Case of the viewing ray missing the box; b) the viewing ray intersecting the box.

As we mentioned this parameter comparison is only valid for the viewing ray increasing in both axes. We can generalize the approach by switching $B_{max,j}$ and $B_{min,j}$ coordinates for all j-axes in which the viewing ray is decreasing. For extension to the 3rd dimension we only need to compare acquired t_{max}, t_{min} to the values $t_{min,z}$, $t_{max,z}$ the same way as we did in x,y comparison.

In this algorithm there is one special case that needs to be handled. That is if a coordinate of the ray direction ***projLin*** is 0. In C++ division by zero is possible for float/double data type and by IEEE standard [15] it yields $\pm\infty$ for positive and negative numerator respectively so the algorithm works correctly. However, if the numerator is equal to 0.0 the result would be NaN (not a number) so we have to set the result to

±infinity. Closely related to this problem is that if we determined the increase/decrease of the ray in an axis based on whether the corresponding coordinate is ≥ 0, by the same standard it is defined that 0.0=-0.0 which would be lethal for the intersection determination. Therefore it is convenient to compute rather with the ray inversion $invq = \left\{\frac{1}{projLin_x}, \frac{1}{projLin_y}, \frac{1}{projLin_z}\right\}$.

Pseudocode for the AABB vs. ray intersection algorithm:

1. Set $B_{min} = \left[\frac{-(\mathrm{IM}+optDist(\mathrm{IM}-1)}{2}, \frac{-W}{2}, \frac{-H}{2}\right]$
   $B_{max} = \left[\frac{\mathrm{IM}+optDist(\mathrm{IM}-1}{2}, \frac{\mathrm{W}}{2}, \frac{\mathrm{H}}{2}\right]$
   $invq = \left\{\frac{1}{projLin_x}, \frac{1}{projLin_y}, \frac{1}{projLin_z}\right\}$
2. If $(invq_i < 0)$ swap $(B_{min,i}, B_{max,i})$ $i \in \{x, y, z\}$
3. Initialize:
   If $((B_{min,i} - P_i) = 0.0 \; and \; invq_i = \pm\infty)$: $t_{min,i} = invq_i$
   else: $t_{min,i} = (B_{min,i} - P_i) * invq_i$

   If $((B_{max,i} - P_i) \neq 0.0 \; and \; invq_i = \pm\infty)$: $t_{max,i} = invq_i$
   else: $t_{max,i} = (B_{max,i} - P_i) * invq_i$
4. If $\left((t_{min,x} > t_{max,y}) \; or \; (t_{min,y} > t_{max,x})\right)$: return *no intersection*
   else set: $t_{max} = min(t_{max,x}, t_{max,y})$
   $t_{min} = max(t_{min,x}, t_{min,y})$
5. If $\left((t_{min} > t_{max,z}) \; or \; (t_{min,z} > t_{max})\right)$: return *no intersection*
   else set: $t_{max} = min(t_{max}, t_{max,z})$
   $t_{min} = max(t_{min}, t_{min,z})$
6. Set: $A = P + \boldsymbol{projLin} * t_{min}$
   $B = P + \boldsymbol{projLin} * t_{max}$
7. Return *A*,*B*

In the software solution the intersection of the ray with box is implemented as a function which returns boolean data type valued true for the ray intersecting and false for missing the box. Moreover if the ray intersects the box the function saves the intersection points to referenced Point structures *A*,*B*.

5.5 Bresenham algorithm

As follows from the previous description of the back to front algorithm we will need to compute thousands of rays going through our 3D data grid and hence the rays determining algorithm should be very fast. Suitable one is a line segment rasterization

algorithm which takes two boundary points of a line segment and generates a sequence of integer voxel coordinates representing a line segment in a raster image. In this subchapter we will introduce the Bresenham line segment rasterization algorithm and its extension to the 3rd dimension based on [6][16][17]. The Bresenham algorithm is an incremental algorithm based on integer arithmetic. Using only integer computations results in significant speed acceleration.

5.5.1 Basic Bresenham algorithm in 2D

As said above for two integer input points A= [x_1, y_1] and B = [x_2, y_2] we are looking for integer line segment representation. The algorithm is derived from the line equation:

$$y = mx + b \tag{38}$$

where $m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$ is the slope of the line segment and b is translation. Knowing the slope m we can determine *main axis* to which given line segment leans to:

$$\begin{cases} m \in \langle -1,1 \rangle & \text{main axis } x \\ otherwise & \text{main axis } y. \end{cases} \tag{39}$$

In the rasterization algorithm we are iteratively moving along the main axis with the unit step and for each such step we decide whether to make a step along the secondary axis as well (Fig.21a)).

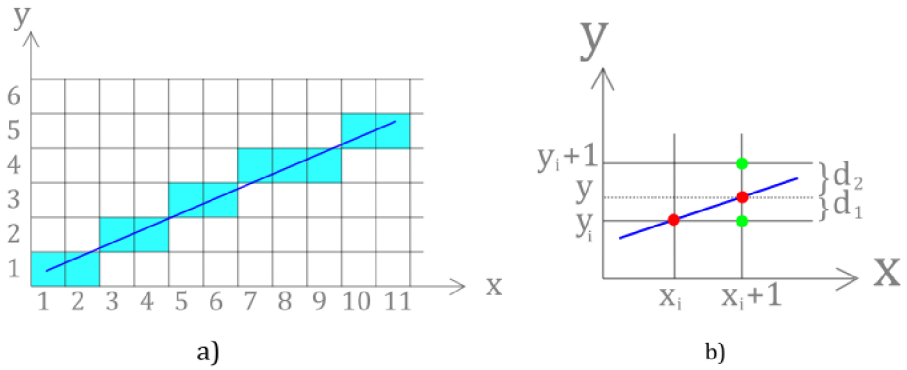


Fig. 21: a) Rasterization of line segment with main axis x. b) Closer pixel determination.

For simplicity the algorithm will be explained assuming:

i. Input points lie in the first quadrant of the plane.
ii. Input points are ordered from left to right. ($x_1 \leq x_2$).
iii. The slope $m \in \langle 0,1 \rangle \rightarrow$ the main axis is x.

Other cases will be discussed in the next subchapter.

Let pixel [x_i,y_i] be processed. Now we have to make a decision whether to move the next pixel up in the y axis [x_i+1, y_i+1] or stay on the same level [x_i+1, y_i]. The decision is based on distances d_1, d_2 of the two possible pixels from the real line segment (Fig.21b). The pixel closer to the real line segment will be picked and processed next. The real line segment coordinate in the y-axis is:

$$y = m(x_i + 1) + b \tag{40}$$

and the distances from each pixel:

$$d_1 = y - y_i = m(x_i + 1) + b - y_i \tag{41}$$

$$d_2 = y_i + 1 - y = y_i + 1 - m(x_i + 1) - b. \tag{42}$$

If we subtract the distances:

$$\Delta d = d_1 - d_2 = 2m(x_i + 1) + 2b - 2y_i - 1 \tag{43}$$

we can determine which pixel to choose only by the sign of the result. If the $\Delta d < 0$ the closer pixel is $[x_i+1, y_i]$ and will be processed next, in the other case we process pixel $[x_i+1, y_i+1]$.

In order to make the calculation faster it is desirable to transform equation (43) to integer form. First we substitute $b=y_1-mx_1$ and get:

$$\begin{aligned}\Delta d &= 2m(x_i + 1) + 2(y_1 - mx_1) - 2y_i - 1 \\ &= 2\frac{\Delta y}{\Delta x}(x_i + 1) + 2(y_1 - \frac{\Delta y}{\Delta x}x_1) - 2y_i - 1.\end{aligned}$$

Next we multiply the equation by positive member Δx so we eliminate the fraction and we get *decision parameter* p_i for i-th step of the algorithm in the form:

$$\begin{aligned}p_i = \Delta d\Delta x &= 2\Delta y(x_i + 1) + 2(y_1\Delta x - \Delta y x_1) - 2y_i\Delta x - \Delta x \\ &= 2\Delta y x_i - 2y_i\Delta x + 2(y_1\Delta x - \Delta y x_1) + 2\Delta y - \Delta x \\ &= 2\Delta y x_i - 2y_i\Delta x + c \end{aligned} \tag{44}$$

where $c = 2(y_1\Delta x - \Delta y x_1) + 2\Delta y - \Delta x$ is a constant. By expressing of the subsequent decision parameter:

$$p_{i+1} = 2\Delta y x_{i+1} - 2y_{i+1}\Delta x + c = 2\Delta y(x_i + 1) - 2y_{i+1}\Delta x + c \tag{45}$$

and subtracting equations (44) and (45) we can express the subsequent decision parameter from p_i:

$$\begin{aligned}p_{i+1} - p_i &= 2\Delta y - 2y_{i+1}\Delta x + 2y_i\Delta x \\ p_{i+1} &= p_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i)\end{aligned} \tag{46}$$

where

$$y_{i+1} - y_i = \begin{cases} 0 & for\ p_i \le 0 \\ 1 & for\ p_i > 0 \end{cases}$$

hence

$$p_{i+1} = \begin{cases} p_i + 2\Delta y & for\ p_i \le 0 \\ p_i + 2\Delta y - 2\Delta x & for\ p_i > 0. \end{cases} \tag{47}$$

The first decision parameter in the sequence is stated by inserting initial point $[x_1, y_1]$ to the equation (47):

$$p_1 = 2\Delta y - \Delta x. \tag{48}$$

Pseudocode for the basic version of the Bresenham algorithm:

1. Set: $\Delta x = x_2 - x_1,\ \Delta y = y_2 - y_1$
2. Initialize $p = 2\Delta y - \Delta x$
3. Set [x, y]=[x_1, y_1]
4. Process point [x, y]
5. While $x \leq x_2$
   a. $x = x + 1$
   b. If $p > 0$ then
      $y = y + 1$
      $p = p + 2(\Delta y - \Delta x)$
      else
      $p = p + 2\Delta y$
   c. Process point [x, y]
6. End.

5.5.2 Bresenham algorithm for all directions in 3D

In the previous subchapter we concentrated on the principle of Bresenham algorithm and we considered one specific case of algorithm in two dimensions. In this subchapter we will extend the algorithm to a general form in the three dimensional space along with an introduction to how the algorithm is implemented in the software solution.

At first we considered *x* as the main axis. In order to generalize the algorithm we first have to decide, which of the three axes the main one is. We determine it easily by comparing the absolute values of input points A[x_1,y_1,z_1], B[x_2,y_2,z_2] differences $|x_2 - x_1|$, $|y_2 - y_1|$, $|z_2 - z_1|$ where the greatest difference determines the main axis along which there will be a constant unit step.

Moreover in the subchapter 5.5.1 we have considered one specific direction given by the input points A, B (increasing in both x and y axis). But there are four possible directions of line rasterization in the plane and eight directions in the 3D space. Some of the cases can even result in sign alterations in (47). Although, if we realize that the line segment increases/decreases in the secondary axes symmetrically with respect to the centre of the line segment, we may omit the alterations using absolute values of $\Delta x, \Delta y, \Delta z$.

Even when the computation of decision parameters p_i is now generalized, we still need to acknowledge the direction of rasterization. This may be expressed by three coefficients $c_x, c_y, c_z \in \{-1,1\}$ where value *1* represents ascending and *-1* descending of the line segment in the considered axis. As a result we move along the main axis with step ± 1 and for each secondary axis we keep its own decision coefficient given by:

$$p_1 = 2\Delta_s - \Delta_m \tag{49}$$

$$p_{i+1} = \begin{cases} p_i + 2\Delta_s & for\ p_i \leq 0 \\ p_i + 2\Delta_s - 2\Delta_m & for\ p_i > 0. \end{cases} \quad (50)$$

where $\Delta_{m,s} = |\Delta_{x,y,z}|$ and subscripts m and s indicate main and secondary axis, respectively.

Pseudocode for the 3D general Bresenham algorithm:

1. Set $\Delta x = |x_2 - x_1|,\ \Delta y = |y_2 - y_1|,\ \Delta z = |z_2 - z_1|$
2. If $x_1 \leq x_2$: $c_x = 1$ else: $c_x = -1$
   If $y_1 \leq y_2$: $c_y = 1$ else: $c_y = -1$
   If $z_1 \leq z_2$: $c_z = 1$ else: $c_z = -1$
3. If $\Delta x \geq \Delta y$ and $\Delta x \geq \Delta z$: main axis is x
   else if $\Delta y \geq \Delta x$ and $\Delta y \geq \Delta z$: main axis is y
   else: main axis is z
4. Link main axis as m, secondary axes as s_1, s_2
5. Initialize $p_1 = 2\Delta_{s1} - \Delta_m$ $p_2 = 2\Delta_{s2} - \Delta_m$
6. Set [x, y, z]=[x_1, y_1, z_1]
7. Process point [x, y, z]
8. For i=1:(dm+1)
   a. $m = m + c_m$
   b. If $p_1 > 0$ then
      $s_1 = s_1 + c_{s1}$
      $p_1 = p_1 + 2\Delta_{s1} - 2\Delta_m$
      else
      $p_1 = p_1 + 2\Delta_{s1}$
   c. If $p_2 > 0$ then
      $s_2 = s_2 + c_{s2}$
      $p_2 = p_2 + 2\Delta_{s2} - 2\Delta_m$
      else
      $p_2 = p_2 + 2\Delta_{s2}$
   d. Process point [m, s_1, s_2]
9. End.

In the software solution the Bresenham algorithm is implemented as two nested functions. The first function takes the two border integer points as an input, determines the main axis and coefficient for each secondary axis (1.-4. in the pseudocode). The second function is then called with differences, coefficients and related positions in the point structure (linking m,s_1, s_2 to the corresponding axes) ordered so that the main axis coefficients are on first positions. Then the line 5.- 9. in the pseudocode is executed and the processed points are saved in the Line structure for later use in back to front algorithm.

5.6 Back to front method implementation

At this point we are getting to the core of the algorithm. As it was outlined at the beginning of this subchapter for each pixel of the output window we will:

1. Cast a ray through it.
2. Compute intersection of the ray with data matrix. (AABB-ray intersection)
3. Determine the set of voxels lying on the intersection. (Bresenham)
4. Perform one of the methods suitable for back to front application.

As for the first step the casted ray (projection line) is represented as a point and direction vector. In the case of perspective projection the point is camera position *P* and the vector is computed as:

$$\boldsymbol{projLin} = P - Q_{ij}$$

where Q_{ij} is a point representing the processed output pixel. In the case of parallel projection the point is Q_{ij} and the direction $\boldsymbol{projLin}$ is simply equal to the direction of view $\boldsymbol{d}$ as it is perpendicular to the output window.

The algorithm proceeds with a function *intersection* which takes the point and direction defining ray, dimensions *dim* and constant *optDist* as input. If there exist an intersection with a data matrix it returns bool value *true,* the intersections are saved to referenced points *A,B* and the algorithm proceeds. On the other hand if there is no intersection the current output pixel value is set to zero and the program proceeds with processing the next output pixel.

In the next step (if the intersections exist) the points *A,B* are rounded to integral values and taken as input to the *Bresenham* function. This function generates the array *AB* of coordinates of the voxels lying on the intersection of the ray with the data matrix from the furthest to nearest. In the implementation there are two versions of the *Bresenham* function. One is for the choice of weighted average for the spaces between the images in the data matrix and returns coordinates of all the voxels on the intersection. Second returns only intersected voxels with data as it is unnecessary to involve void voxels in the computation.

The fourth step applies one of the imaging methods, considered in the theoretical part, on the voxels intersected by the ray. As the methods are straightforward we will describe their implementation in terms of short pseudocodes.

- Maximal intensity method

1. Initialize voxel $Output_{R,G,B} = 0$, intensitySum = 0, maxIntensitySum = 0,
2. For each voxel *P* in *AB:*

   If ($P_R > threshold$ OR $P_G > threshold$ OR $P_B > threshold$)
   intensitySum=$P_R + P_G + P_B$
   If (intensitySum> maxIntensitySum)
   maxIntensitySum= intensitySum
   $Output_{R,G,B} = P_{R,G,B}$
   else continue with next voxel;
3. End.

- Averages

1. Initialize voxel $Output_{R,G,B} = 0$
2. For each voxel P in AB:

   If ($P_R > threshold$ OR $P_G > threshold$ OR $P_B > threshold$)
   $Output_{R,G,B} = P_{R,G,B} + Output_{R,G,B}$
   else continue with next voxel;
3. $Output_{R,G,B} = \frac{Output_{R,G,B}}{sizeOf(AB)}$
4. End.

- Constant transparency method:

1. Initialize voxel $Output_{R,G,B} = 0$
2. For each voxel P in AB:

   If ($P_R > threshold$ OR $P_G > threshold$ OR $P_B > threshold$):
   $Output_{R,G,B} = P_{R,G,B} * (1 - transp) + Output_{R,G,B} * transp$
   else continue with next voxel;
3. End.

- Volume rendering integral with constant transfer function

1. Initialize voxel $Output_{R,G,B} = 0$
2. For each voxel P in AB:

   If (($P_R > threshold$ OR $P_G > threshold$ OR $P_B > threshold$):
   $Output_{R,G,B} = P_{R,G,B} + Output_{R,G,B} * transp$
   else continue with next voxel;
3. End.

- Volume rendering integral with user defined transfer function

1. Initialize voxel $Output_{R,G,B} = 0$
2. For each voxel P in AB:

   If (($P_R > threshold$ OR $P_G > threshold$ OR $P_B > threshold$) AND ($P_{R,G,B}$ $is\ within\ defined\ boundaries$):
   $Output_{R,G,B} = P_{R,G,B} * tf(P_{R,G,B}) + transp * Output_{R,G,B}$
   else continue with next voxel;
3. End.

When the computation of either method is done the resulting pixel values are modified by the user setting of *brightness* and *contrast*:

$$pixOut_{R,G,B} = pixOut_{R,G,B} * contrast + brightness$$

and the algorithm continues with processing the next pixel. When all the output pixels are processed the result is saved in the folder with project as *Output.bmp*.

6 RESULTS AND DISCUSSION

The implementation of methods for scalar data visualization described in the chapter 5 was tested on the following three datasets acquired by confocal microscope as series of BMP images with dynamic range 8 bits per colour component:

- *Dataset1*: two speciments of protozoan Paramecium caudatum, real size cca 300 μm
  No of Images: 57
  Width: 1600
  Height: 1600
  (relatively high resolution, relatively small noise)
- *Dataset2*: tobacco cell, real size cca 200 μm
  No of Images: 100
  Width: 800
  Height: 800
  (suitable small resolution, heavily degradated by noise)
- *Dataset3*: tobacco cell, real size cca 350 μm
  No of Images: 96
  Width: 954
  Height: 638
  (suitable resolution, slightly degradated by noise)

Note that in the ZIP attachment to the thesis, there is only Dataset1 provided with half of the original resolution due to size restrictions. On the attached CD there are all three datasets in full resolution. In the attached files there are also two versions of application. One is pure algorithm where the manipulation with input data has to be done in the actual code. The second is implemented as a graphic user interface where the inputs can be entered and changed intuitively.

In the appendix of the thesis you can find the examples of images generated by the algorithm. Particularly there are three examples from each set and each set is shown from different points of view generated by back to front method with constant transparency in order to give an idea how the original specimen looked. Moreover there are results of average projection, maximal intensity projection and volume rendering integral with constant transfer function and user defined transfer function method with two versions transfer function for each dataset. Further there is shown an impact of optical distance setting and difference between parallel and perspective view.

From chapter 5, where the algorithm functionality is described is obvious, that in general the parallel projection is faster than projective, as we don't have to compute the viewing ray direction. With the same clarity we can say that treating optical distance in-between the voxels as an empty space rather than estimating weighted average of incident voxels is exceptionally faster. In fact, depending on the defined optical distance variable, this artificial stuffing of the void voxels quickly becomes the heaviest process in the algorithm. Looking at the implemented methods, we can observe that the maximal intensity and average projections are very similar and relatively fast in terms

of processing time. On the other hand these methods are quite rigid when it comes to output information. On the contrary, back to front method with constant transparency and volume rendering integral provide more flexibility in the image output definition and consequently in the amount of information obtainable at the cost of longer computation time. Naturally the constant transfer function for volume rendering integral is faster than the method with user defined transfer function as in the latter we need to check whether the value of a voxel lies in the interval for each voxel in the data matrix and each interval defined.

Note that we may only need to reconstruct the frontal 3D view of the scanned specimen (in the direction of optical axis x). Then it is not needed to fill the void between the voxels, as those empty spaces won't be visible. Moreover in this case when we apply a parallel projection method the optical distance becomes redundant, as it won't result in change in the output. Knowing this, we can significantly reduce the runtime of the algorithm.

7 CONCLUSION

The first theoretical chapter in this thesis is devoted to confocal microscopy. In particular we have described how simulated fluorescence from fluorophores serves in widefield and confocal microscopy. We have continued with explanation how confocal microscopes focus on one particular point in a focal plane and they filter out of focus light using two pinhole aperture and two confocal lenses. Further we discussed the whole microscope setup. The chapter continues with introduction of limitations regarding confocal microscopy. We have mentioned light detection limitations – the resolution limitation caused by diffraction phenomena, pinhole size, laser intensity and amount of fluorophore and scanning speed limitations followed by description of acousto-optic deflectors and Nipkow discs addressing the speed limitation issues.

Chapter three sets the mathematical background for computer geometry and back-to-front algorithm in particular. In this chapter we have quickly recapitulated the basic definitions regarding linear, normed, unitary, affine and Euclidean spaces and put them in context with analytic geometry. The second subchapter brings up projective space definitions followed by introduction of parallel and perspective projections and ends with a short discussion on their suitability.

The fourth chapter leads toward the back-to-front algorithm. First we have introduced a direct volume rendering concept which is advantageous for data with inner structures imaging and possibilities of representation of such volumetric data in the computer with accent to 3D scalar fields. In order to create realistic rendering of objects the direct volume rendering methods are mostly based on volume rendering integral, which describes the light propagation through the participating media. In the second subchapter we have introduced the basic radiometric quantities used for describing the transmission of light: photon energy, radiant energy, radiant flux, irradiance, radiant intensity and most importantly radiance. Based on the theory we have described the derivation of the volume rendering integral by combination of absorption and emission of light by particles in participating media. The theoretical part of the thesis culminates in the fourth subchapter, where the back-to-front visualization method is obtained by discretization of volume rendering integral followed by brief introduction of transfer functions and methods suitable for imaging scalar data fields by back-to-front method: maximal intensity projection, average intensity projection, constant transparency method and volume rendering integral method.

The fifth chapter is dedicated to step by step description of the back-to-front algorithm implementation. We have mentioned the loading of the images from the given dataset and continued with introduction of the user defined projective and graphical variables, their possible ranges meaning in the algorithm. Having the input variables defined we approached the output window generation where we determined its exact position in the coordinate system based on the camera position, distance of the window from camera position, optic angle or magnitude (depending on the projection) and

resolution of the input/output images. As subsequent parts of the algorithm are repeated up to millions times (depending on images resolution) we have implemented fast methods of their computation. The first algorithm is used for finding intersection of a ray with axis aligned bounding box using comparison of parameters from parametric line equation. Another fast implemented method is Bresenham algorithm for line segment rasterization. The algorithm is based on integer arithmetic where a simple decision parameter is compared to zero. The last section of the practical part describes implementation of the back-to-front methods introduced in chapter four.

In the sixth chapter we have discussed the results of implemented methods in the terms of computational complexity and their suitability for different result purposes. For the end we can summarize, that the maximal intensity projection, average projection, constant transparency method and volume rendering integral method with constant transfer function can be used for getting an overall idea about given data whereas volume rendering integral method with user defined transfer function is more suitable for specific information extraction. Furthermore the parallel projection with void in-between the data is better for measurement purposes where the precision is required, while perspective projection and filling the optical distance with weighted average provides a more realistic impression.

8 LITERATURE

[1] Semwogerere, D, Weeks, ER. Confocal microscopy. In: Wnek, G, Bowlin, G (eds) Encyclopedia of biomaterials and biomedical engineering. New York: Taylor & Francis, 2005, pp. 705–714.

[2] PRASAD, V, D SEMWOGERERE a Eric R WEEKS. Confocal microscopy of colloids. *Journal of Physics: Condensed Matter* [online]. 2007, **19**(11) [cit. 2020-06-24]. DOI: 10.1088/0953-8984/19/11/113102. ISSN 0953-8984.

[3] VERGARA-IRIGARAY, Nuria, Michèle RIESEN, Gianluca PIAZZA, et al. Laser Scanning Confocal Microscopy. *Encyclopedia of Nanotechnology*. Dordrecht: Springer Netherlands, 2012, 2 1192-1192. DOI: 10.1007/978-90-481-9751-4_100341

[4] MARTIŠEK, Dalibor. *Matematické principy grafických systémů*. Brno: Littera, 2002. ISBN 80-857-6319-2.

[5] SOJKA, Eduard. *Počítačová grafika II: průvodce studiem*. Ostrava: VŠB - Technická univerzita, Regionální centrum celoživotního vzdělávání, 2003. ISBN 80-248-0293-7.

[6] ŽÁRA, Jiří. *Moderní počítačová grafika*. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[7] DUNCAN, Marsh. *Applied geometry for computer graphics and CAD*. Springer: London, 2000. ISBN 1-85233-080-5.

[8] HANSEN, Charles D. a Chris R. JOHNSON. *The visualization handbook*. Burlington: Elsevier Butterworth-Heinemann, 2005. ISBN 0-12-387582-X.

[9] YAGEL, R., D.M. REED, A. LAW, PO-WEN SHIH a N. SHAREEF. Hardware assisted volume rendering of unstructured grids by incremental slicing. In: *Proceedings of 1996 Symposium on Volume Visualization* [online]. ACM, 1996, 55-62 [cit. 2020-06-24]. DOI: 10.1109/SVV.1996.558043.

[10] SHIRLEY, Peter a Stephen Robert MARSCHNER. *Fundamentals of computer graphics*. 3rd ed. Natick, Mass.: A K Peters, c2009. ISBN 978-1-56881-469-8.

[11] MCCLUNEY, William. *Introduction to Radiometry and Photometry* (1st ed 1994). Boston, London: Artech House Publ, ISBN 0-89006-678-7.

[12] BLINN, James F. Light reflection functions for simulation of clouds and dusty surfaces. *ACM SIGGRAPH Computer Graphics* [online]. 1982, **16**(3), 21-29. DOI: 10.1145/965145.801255 Accessible: http://portal.acm.org/citation.cfm?doid=965145.801255

[13] MAX, N. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* [online]. **1**(2), 99-108 DOI: 10.1109/2945.468400.

[14] WILLIAMS, Amy, Steve BARRUS, R. Keith MORLEY a Peter SHIRLEY. An efficient and robust ray-box intersection algorithm. In: *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*. New York, New York, USA: ACM Press, 2005, 2005, s. 9 DOI: 10.1145/1198555.1198748.

[15] IEEE Standards Association. IEEE standard for binary floating-point arithmetic. IEEE Report (New York), 1985. ANSI/IEEE Std 754-1985.

[16] BRESENHAM, J. E. Algorithm for computer control of a digital plotter. *IBM Systems Journal* [online]. 1965, **4**(1), 25-30. DOI: 10.1147/sj.41.0025. ISSN 0018-8670

[17] MARTIŠEK, Dalibor a MARTIŠEK Karel. Direct Volume Rendering Methods for Cell Structures. *Scanning* [online]. 2012, **34**(6), 367-377. DOI: 10.1002/sca.21019. ISSN 01610457

[18] ENGEL, Klaus. *Real-time volume graphics*. Wellesley: A.K. Peters, c2006. ISBN 1-56881-266-3.

[19] LJUNG, Patric, Jens KRÜGER, Eduard GROLLER, Markus HADWIGER, Charles D. HANSEN a Anders YNNERMAN. State of the Art in Transfer Functions for Direct Volume Rendering. *Computer Graphics Forum* [online]. 2016, **35**(3), 669-691. DOI: 10.1111/cgf.12934

9 LIST OF ATTACHEMENTS

CD.
Compressed ZIP file with implementations, datasets and example outputs.

APPENDIX

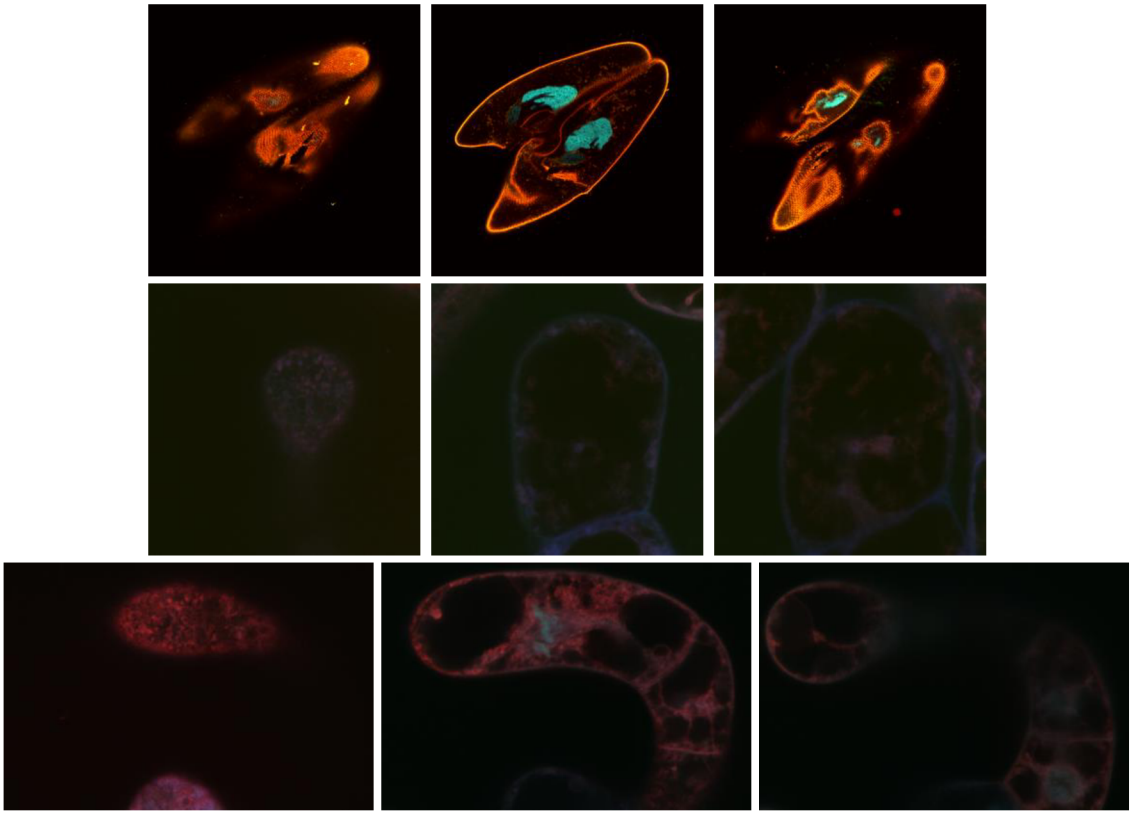


Fig. 22: Examples of images from dataset 1,2 and 3

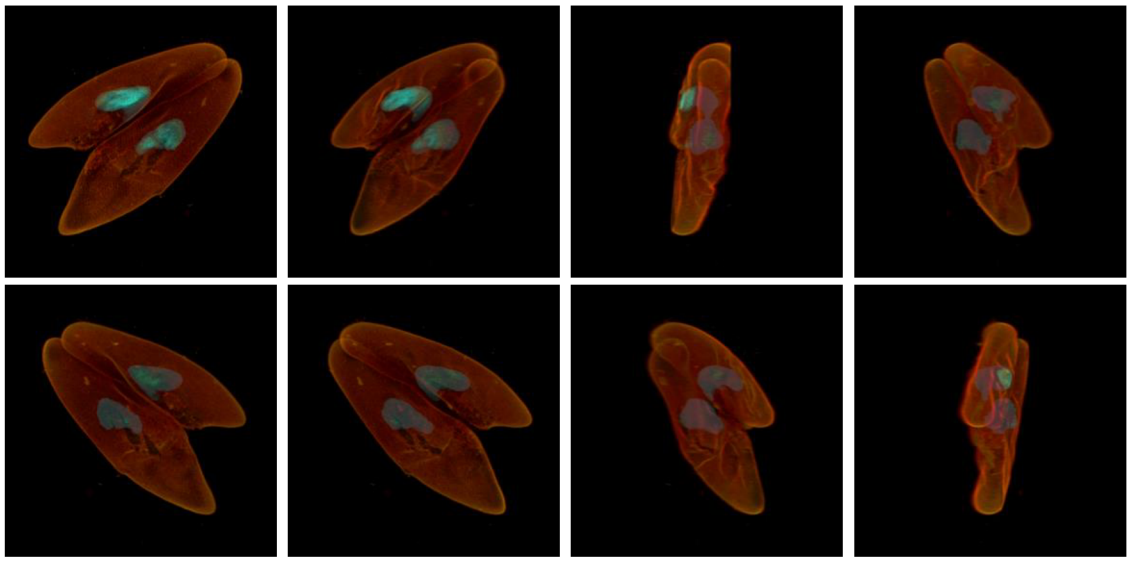


Fig. 23: DATASET1: constant transparency method, parallel projection, mag=1.0, distance=300, optical distance=4, empty spaces approximation on, transparency=99%, threshold=30, brightness=0, contrast=1.0.

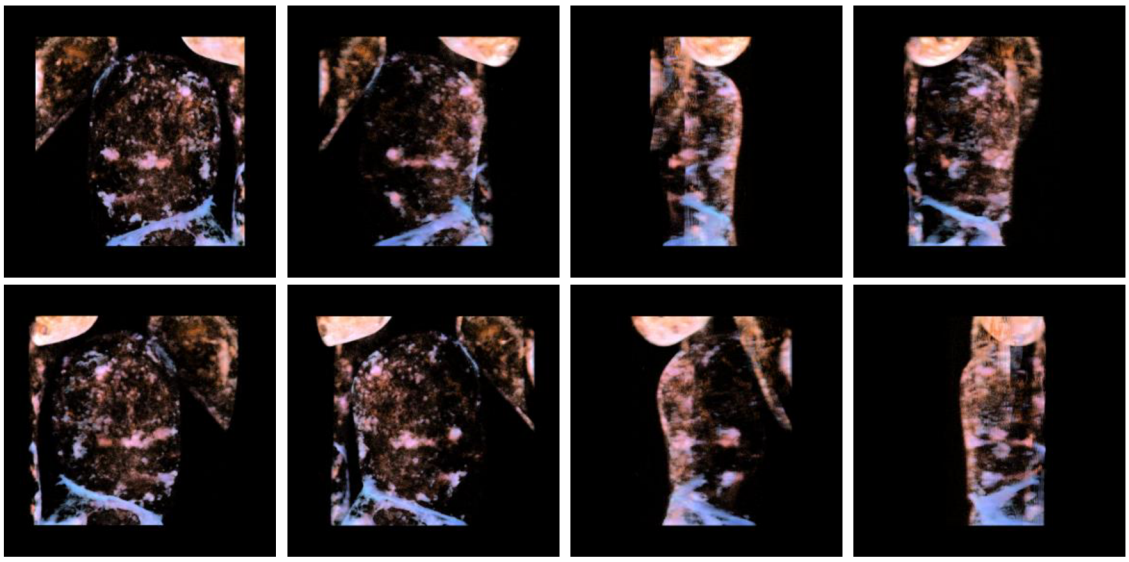


Fig. 24: DATASET2: constant transparency method, parallel projection, mag=1.3, distance=300, optical distance=2, empty spaces approximation on, transparency=85%, threshold=40, brightness=0, contrast=5.

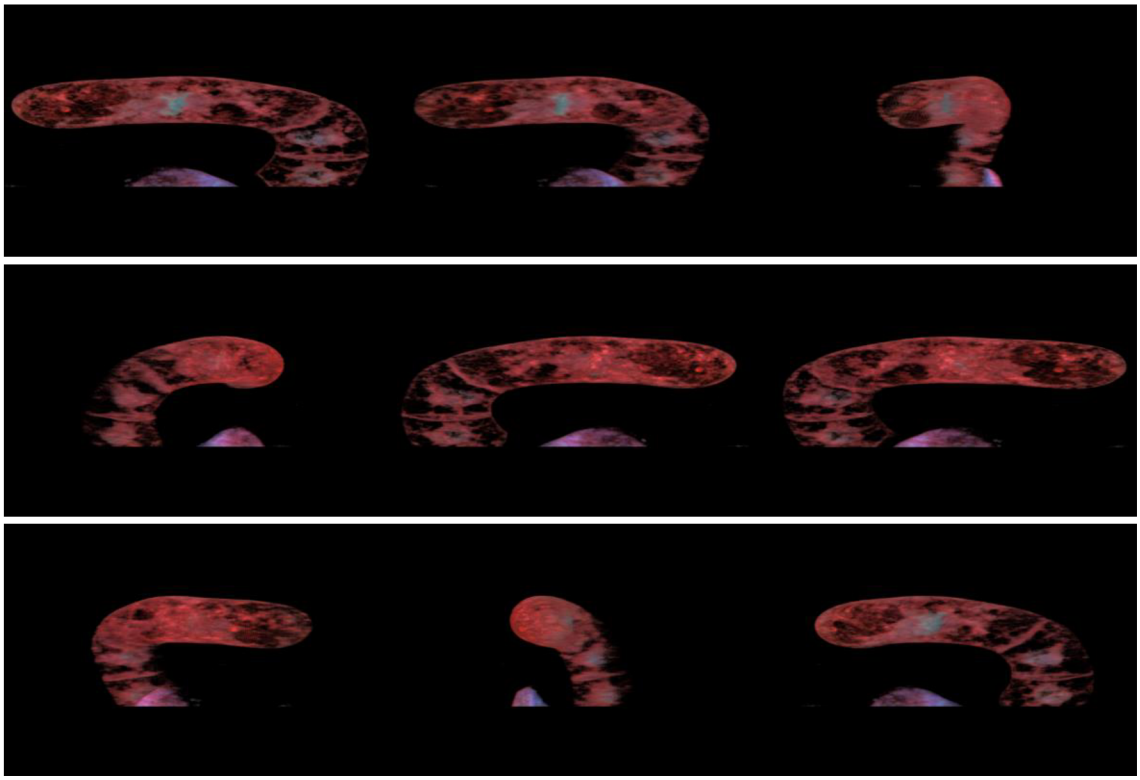


Fig. 25: DATASET3: constant transparency method, parallel projection, mag=1.0, distance=300, optical distance=2, empty spaces approximation on, transparency=85%, threshold=50, brightness=0, contrast=2.0.

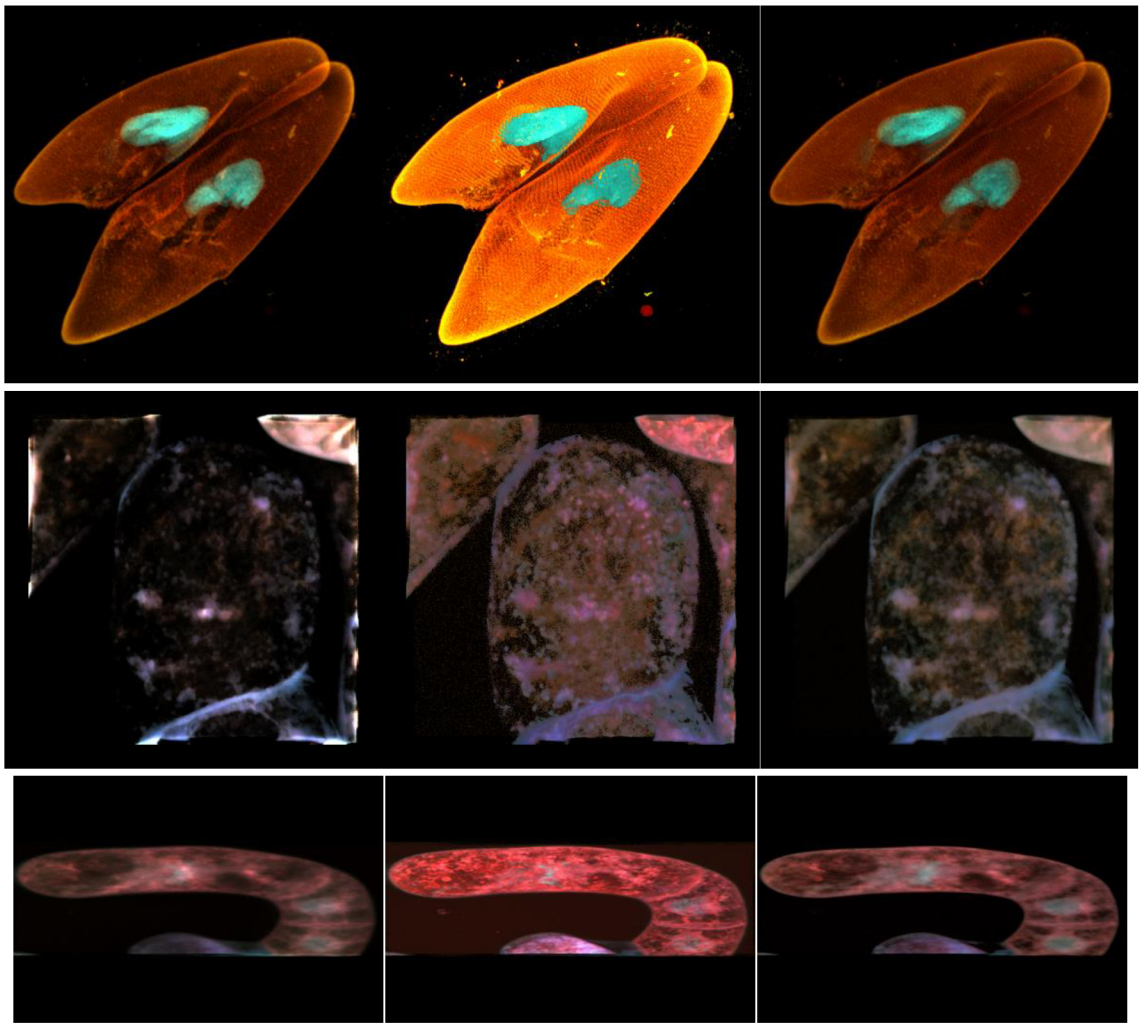
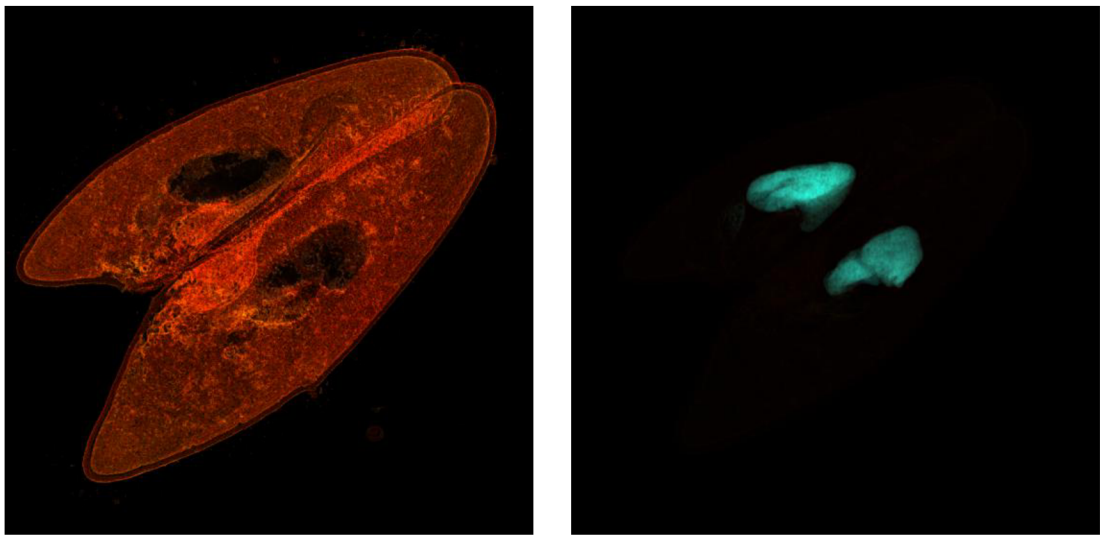


Fig. 26: a) Average ,b) maximal intensity ,c) volume rendering integral method constant transfer function



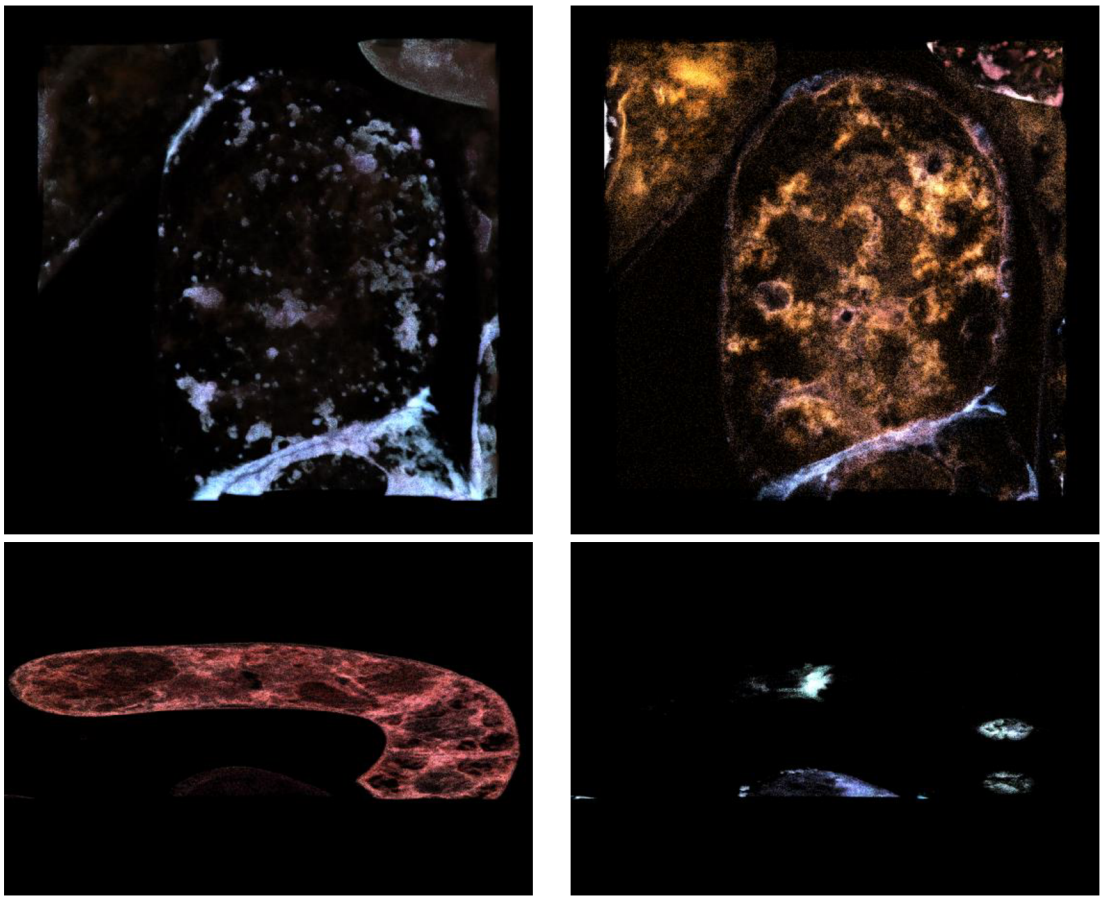


Fig. 27: Volume rendering integral method with user defined transfer function

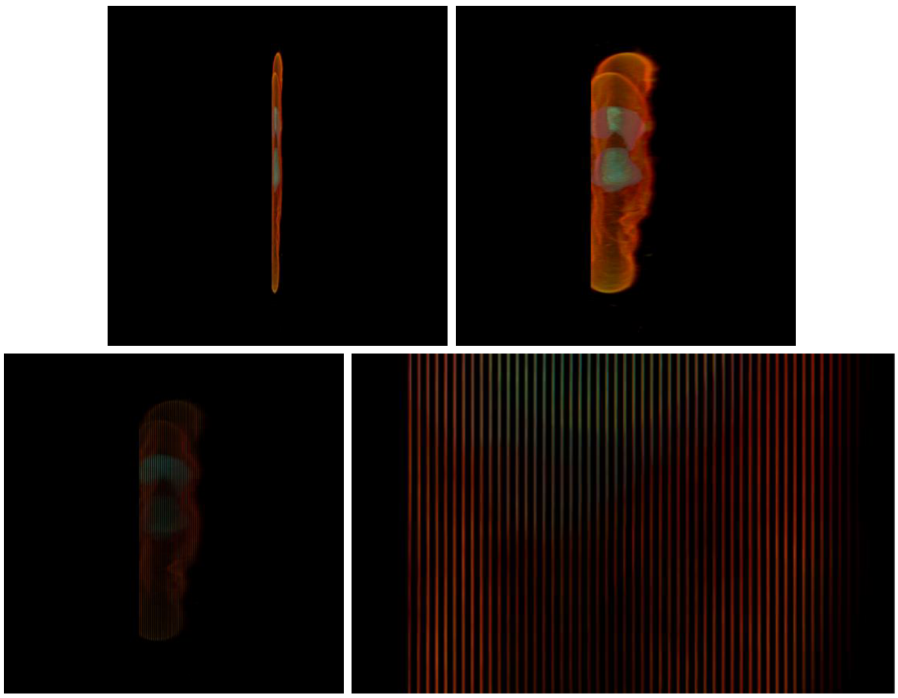


Fig. 28: a) Optical distance= 0 b) optical distance=5 with weighted average approximation c) optical distance=5 without weighted average approximation with parallel projection d) close-up view of c).

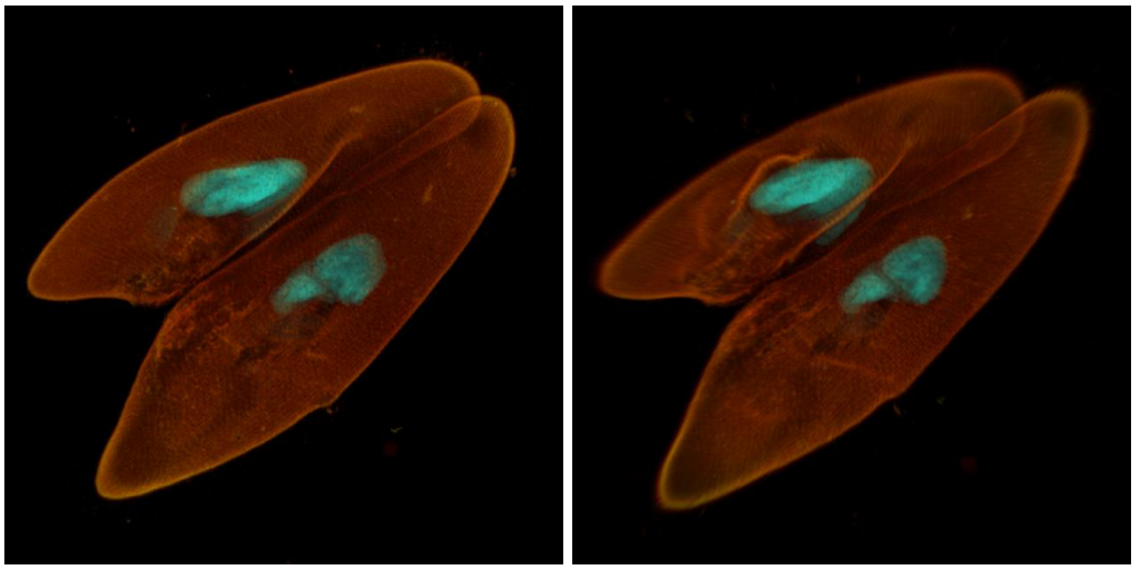


Fig. 29: a) Parallel projection b) perspective projection with optical distance= 20.