Introduction

1

Computer graphics is the process of creating images using a computer. This process is often referred to as graphical data processing. In this book, an *image* is understood in an abstract sense. An image not only can represent a realistic scene from the everyday world, but it can also be graphics such as histograms or pie charts, or the graphical user interface of the software. In the following section, some application fields of computer graphics are presented as examples to give an impression of the range of tasks in this discipline. This is followed by explanations of the main steps in computer graphics and an overview of how a rendering pipeline works using the graphics pipeline of the *Open Graphics Library (OpenGL)*.

Computer graphics belongs to the field of *visual computing*. Visual computing, also known as image informatics, deals with both image analysis (acquisition, processing and analysis of image data) and image synthesis (production of images from data). Visual computing is an amalgamation of individual merging fields such as image processing, computer graphics, computer vision, human–machine interaction and machine learning. Computer graphics is an essential part of image synthesis, just as image processing is an essential part of image analysis. Therefore, in basic introductions to visual computing, the two disciplines of computer graphics and image processing are taught together. This book also integrates solutions to image processing problems, such as the reduction of aliasing in rasterisation or the representation of object surfaces using textures.

Further links to computer graphics exist with neighbouring disciplines such as computer-aided design/manufacturing (CAD/CAM), information visualisation, scientific visualisation and augmented reality (AR) and virtual reality (VR) (see Chap. 11).

K. Lehn et al., *Introduction to Computer Graphics*, Undergraduate Topics in Computer Science, https://doi.org/10.1007/978-3-031-28135-8_1

1.1 Application Fields

Although graphical user interfaces (GUIs) are fundamentally an application area of computer graphics, the basics of computer graphics now play a rather subordinate role in this field. On the one hand, there are standardised tools for the design and programming of graphical user interfaces that abstract the technical processes, and on the other hand, the focus is primarily on usability and user experience and thus in the area of human–computer interaction.

In advertising and certain art forms, many images are now produced entirely by computer. Artists rework or even alienate photographs using computer graphics techniques.

Besides the generation and display of these rather abstract graphics, the main application of computer graphics is in the field of displaying realistic—not necessarily real—images and image sequences. One such application is animated films, which are created entirely using computer graphics methods. The realistic rendering of water, hair or fur is a challenge that can be achieved already in a reasonable amount of time using sophisticated rendering systems.

For the very large amounts of data that are recorded in many areas of industry, business and science, not only suitable methods for automatic analysis and evaluation are required but also techniques for the visualisation of graphic presentations. The field of interactive analysis using visualisations is called *visual analytics*, which belongs to the field of *information visualisation*. These visualisations go far beyond simple representations of function graphs, histograms or pie charts, which are already accomplished by spreadsheet programs today. Such simple types of visualisations are assigned to *data visualisation*. This involves the pure visualisation of raw data without taking into account an additional relationship between the data. Data becomes information only through a relationship. Information visualisations include two- or three-dimensional visualisations of high-dimensional data, problem-adapted representations of the data [3,7,12,13] or special animations that show temporal progressions of, for example, currents or weather phenomena. The interactive visualisation of this information and the subsequent analysis by the user generates knowledge. The last step is assigned to visual analytics. Artificial intelligence techniques can be applied in this context as well.

Classic applications come from the fields of *computer-aided design (CAD)* and *computer-aided manufacturing (CAM)*, which involve the design and construction of objects such as vehicles or housings. The objects to be represented are designed on the computer, just as in computer games or architectural design programs for visualising real or planned buildings and landscapes. In these cases, the objects to be designed, for example an extension to a house, are combined with already existing objects in the computer. In the case of driving or flight simulators, real cities or landscapes have to be modelled with the computer.

Not only does the ability to model and visualise objects plays an important role in computer graphics, but also the generation of realistic representations from measured data. To generate such data, scanners are used that measure the depth information of the surfaces of objects. Several calibrated cameras can also be used for this purpose,

from which three-dimensional information can be reconstructed. A very important field of application for these technologies is medical informatics [6]. In this area, for example, three-dimensional visualisations of organs or bone structures are generated, which are based on data that was partly obtained by various imaging procedures, such as ultrasound, computer tomography and magnetic resonance imaging.

The coupling of real data and images with computer graphics techniques is expected to increase in the future. Computer games allow navigating through scenes and viewing a scene from different angles. Furthermore, 360-degree videos allow for an all-around view of a scene. Such videos can be recorded using special camera arrangements, such as omnidirectional cameras. These are spherical objects equipped with several cameras that take pictures in different directions.

In most films in the entertainment industry, the viewer can only watch a scene from a fixed angle. The basic techniques for self-selection of the viewer position are already in place [4]. However, this requires scenes to be recorded simultaneously from multiple perspectives and intelligent playback devices. In this case, the viewer must not be bound to the positions of the cameras, but can move to any viewpoint. The corresponding representation of the scene is calculated from the information provided by the individual cameras. For this purpose, techniques of computer graphics, which serve the synthesis of images, have to be combined with procedures from image processing, which deal with the analysis of the recorded images [5].

Other important fields of application of computer graphics are *virtual reality (VR)* and *augmented reality (AR)*. In virtual reality, the aim is to create an immersion for the user through the appropriate stimulation of as many senses as possible, which leads to the perception of really being in the artificially generated world. This desired psychological state is called *presence*. This results in a wide range of possible applications, of which only a few are mentioned as examples here. Virtual realities can be used in architectural and interior simulations, for telepresence or learning applications in a wide variety of fields. Psychotherapeutic applications to control phobia (confrontation therapy) or applications to create empathy by a user putting himself in the situation of another person are also possible and promising applications.

Through the use of augmented reality technologies, real perception is enriched by additional information. Here, too, there are many possible applications, of which only a few are mentioned. Before a purchase, products can be selected that fit perfectly into their intended environment, for example, the new chest of drawers in the living room. Likewise, clothes can be adapted to individual styles in augmented reality. Learning applications, for example a virtual city or museum guide, are just as possible as assembly aids in factories or additional information for the maintenance of complex installations or devices, for example elevators or vehicles. Conferences can be supplemented by virtual avatars of participants in remote locations. In medicine, for the planning of an operation or during an operation, images can be projected precisely onto the body to be operated on as supplementary information, which was or is generated before or during the operation by imaging techniques.

As these examples show, computer graphics applications can be distinguished according to whether interaction with the application is required or not. For the production of advertising images or classic animated films, for example, no interaction

is intended, so elaborate computer graphics processes can be used for the most realistic representation possible. In the film industry, entire computer farms are used to calculate realistic computer graphics for a feature-length animated film, sometimes over a period of months.

However, many computer graphics applications are interactive, such as applications from the fields of virtual or augmented reality. This results in requirements for the response time to generate a computer graphic. For flight or vehicle simulators, which are used for the training of pilots or train drivers, these response times must be strictly adhered to in order to be able to represent scenarios that create a realistic behaviour of the vehicle. Therefore, in computer graphics it is important to have methods that generate realistic representations as efficiently as possible and do not necessarily calculate all the details of a scene. For professional flight simulators or virtual reality installations, high-performance computer clusters or entire computer farms are potentially available. However, this is not the case for computer games or augmented reality applications for portable devices, even though in these cases the available computing power is constantly growing. For this reason, simple, efficient models and methods of computer graphics, which have been known for a long time, are in use today to create interactive computer graphics applications.

1.2 From the Real Scene to the Computer Generated Image

From the example applications presented in Sect. 1.1, the different tasks to be solved in computer graphics can be derived. Figure 1.1 shows the rough steps needed to get from a real or virtual scene to a perspective image. First, the objects of the scene must be replicated as computer models. This is usually done with a modelling tool. This replication is generally only an approximation of the objects in the scene. Depending on the effort to be expended and the modelling types available, the objects can be approximated more or less well. This is illustrated in Fig. 1.1a, where a house was modelled by two simple geometric shapes. The deciduous trees consist of a cylinder and a sphere. The coniferous trees are composed of a cylinder and three intersecting cones.

One of the first steps in graphics processing is to determine the viewer location, also known as the camera position or eye point. Figure 1.1b shows the modelled scene as seen by a viewer standing at some distance in front of the house.

The modelled objects usually cover a much larger area than what is visible to the viewer. For example, several building complexes with surrounding gardens could be modelled, through which the viewer can move. If the viewer is standing in a specific room of the building, or is at a specific location in the scene looking in a specific direction, then most of the modelled objects are not in the viewer's field of view and can be disregarded in the rendering of the scene. The example scene in Fig. 1.1 consists of a house and four trees, which should not be visible at the same time. This is represented in Fig. 1.1c by the three-dimensional area marked by lines. If a perspective projection is assumed, then this area has the shape of a *frustum (truncated*

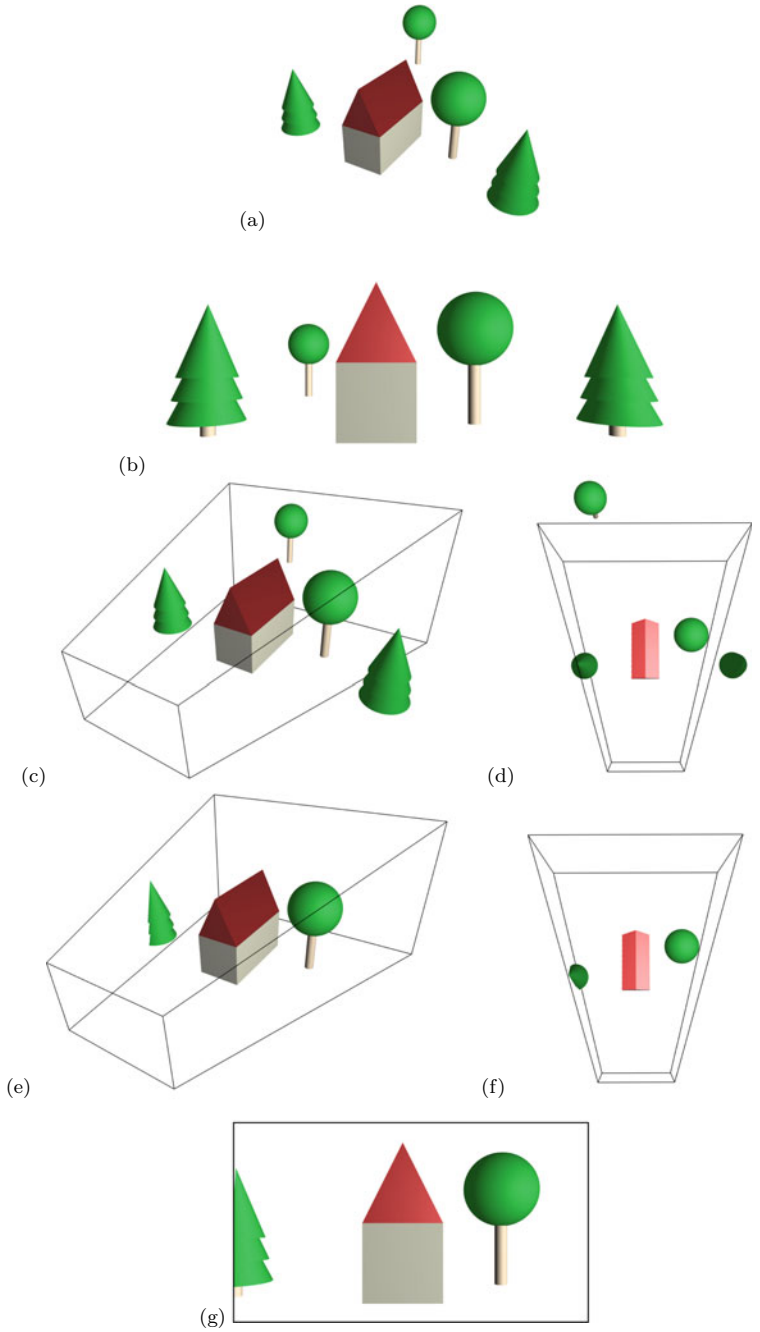

Fig. 1.1 Main steps of computer graphics to get from a real scene to a computer generated perspective image

pyramid) with a rectangular base. The (invisible) top of the pyramid is the viewer's location. The smaller boundary surface at the front (the clipped top of the pyramid) is called the *near clipping plane* (*near plane*). The area between the viewer and the near clipping plane is not visible to the viewer in a computer generated scene. The larger boundary surface at the rear (the base surface of the pyramid) is called *far clipping plane* (*far plane*). The area behind the far plane is also not visible to the viewer. The other four boundary surfaces limit the visible area upwards, downwards and to the sides. Figure 1.1d shows a view of the scene from above, revealing the positions of the objects inside and outside the frustum.

The process of determining which objects are within the frustum and which are not is called *(three-dimensional) clipping*. This is of great interest in computer graphics in order to avoid unnecessary computing time for parts of the scene that are not visible. However, an object that lies in this perceptual area is not necessarily visible to the viewer, as it may be obscured by other objects in the perceptual area. The process of determining which parts of objects are hidden is called *hidden face culling*. Clipping and culling are processes to determine the *visibility* of the objects in a scene.

It is relatively easy to remove objects from the scene that are completely outside of the frustum, such as the deciduous tree behind the far clipping plane and the conifer to the right of the frustum (see Fig. 1.1c–d). Clipping objects that are only partially in the clipping area, such as the conifer on the left in the illustration, is more difficult. This may require splitting geometrical objects or polygons and closing the resulting shape. Figure 1.1e–f shows the scene after clipping. Only the objects and object parts that are perceptible to the viewer and thus lie within the clipping area need to be processed further.

The objects that lie within this area must then be projected onto a two-dimensional surface, resulting in an image that can be displayed on the monitor or printer as shown in Fig. 1.1g. During or after this projection, visibility considerations must be made to determine which objects or which parts of objects are visible or obscured by other objects.

Furthermore, the illumination and light effects play an essential role in this process for the representation of visible objects. In addition, further processing steps, such as rasterisation, are required to produce an efficiently rendered image of pixels for displaying on an output device, such as a smartphone screen or a projector.

1.3 Rendering and Rendering Pipeline

The entire process of generating a two-dimensional image from a three-dimensional scene is called *rendering*. The connection in the series of the individual processing units roughly described in Fig. 1.1 from part (b) onwards is called *computer graphics pipeline*, *graphics pipeline* or *rendering pipeline*.

The processing of data through a pipeline takes place with the help of successively arranged stages in which individual (complex) commands are executed. This type of processing has the advantage that new data can already be processed in the first stage

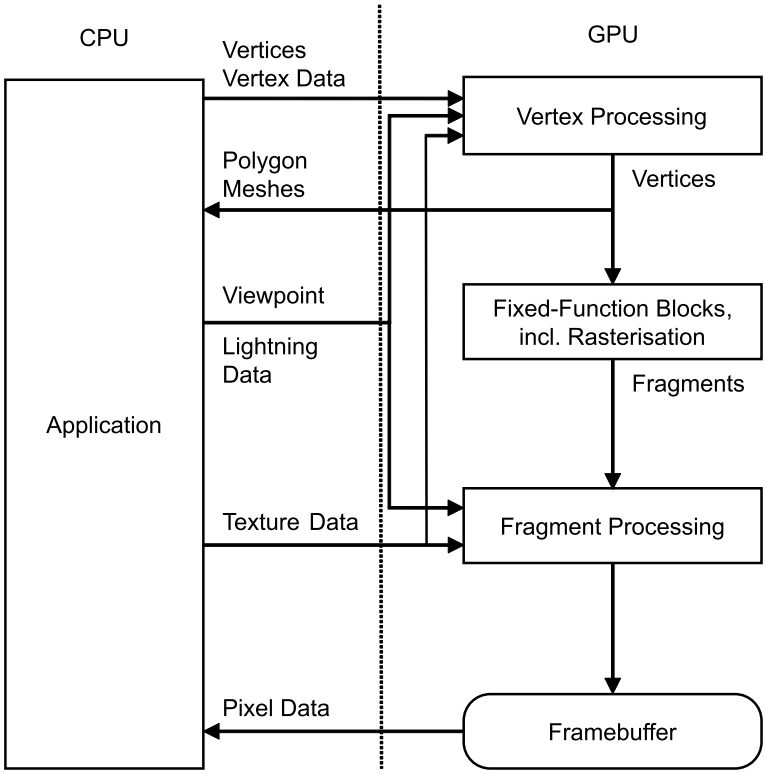

Fig. 1.2 Abstracted rendering pipeline of the Open Graphics Library (OpenGL). The basic structure is similar to the graphics pipeline of other graphics systems

of the pipeline as soon as the old data has been passed on to the second stage. There is no need to wait for the complete computation of the data before new data can be processed, resulting in a strong parallelisation of data processing. Data in graphical applications is particularly well suited for pipeline processing due to its nature and the processing steps to be applied to it. In this case, parallelisation and thus a high processing speed can easily be achieved. In addition, special graphics hardware can accelerate the execution of certain processing steps.

Figure 1.2 shows an abstract representation of the graphics pipeline of the *Open Graphics Library (OpenGL)* on the right. Since the stages shown in the figure are also used in this or similar form in other graphical systems, this pipeline serves to explain the basic structure of such systems. More details on OpenGL and the OpenGL graphics pipeline can be found in Chap. 2.

On the left side in the figure is the graphics application, which is usually controlled by the main processor, the *central processing unit (CPU)*, of the computer. This application sends the three-dimensional scene (3D scene) consisting of modelled objects to the *graphics processing unit (GPU)*. Graphics processors are integrated

into modern computer systems in different ways. A graphics processor can be located in the same housing as the CPU, in a separate microchip on the motherboard or on a separate graphics card plugged into the computer. Graphics cards or computers can also be equipped with several graphics processors. In this book, the word *graphics processor* or the abbreviation *GPU* is used to represent all these variants.

In most cases, the surfaces of the individual (3D) objects in a 3D scene are modelled using polygons (for example, triangles) represented by *vertices*. A *vertex* (plural: *vertices*) is a corner of a polygon. In addition, in computer graphics it is a data structure that stores, for example, the position coordinates in 3D-space and colour values (or other data) at the respective corner of the polygon (see Sect. 2.5.1). A set of vertices (vertex data) is used to transfer the 3D scene from the application to the GPU. Furthermore, additional information must be passed on how the surfaces of the objects are to be drawn from this data. In Chap. 3, the basic geometric objects available in OpenGL and their application for drawing objects or object parts are explained. Chapter 4 contains basics for modelling three-dimensional objects.

Furthermore, the graphics application transmits the position of the viewer, also called camera position or viewer location, (mostly) in the form of matrices for geometric transformations to the GPU. With this information, vertex processing takes place on the GPU, which consists of the essential step of geometry processing. This stage calculates the changed position coordinates of the vertices due to geometric transformations, for example, due to the change of the position of objects in the scene or a changed camera position. As part of these transformations, a *central projection* (also called perspective projection) or a *parallel projection* can be applied to obtain the spatial impression of the 3D scene. This makes the scene mapping onto a two-dimensional image. Details on geometry processing can be found in Chap. 5.

Furthermore, in vertex processing, an appropriate *tessellation* (see Sect. 4.2) can be used to refine or coarsen the geometry into suitable polygons (mostly triangles). After vertex processing (more precisely in vertex post-processing; see Sect. 2.5.2), the scene is reduced to the area visible to the viewer, which is called *clipping* (see Sect. 8.1).

Until after vertex processing, the 3D scene exists as a vector graphic due to the representation of the objects by vertices (see Sect. 7.1). *Rasterisation* (also called *scan conversion*) converts this representation into a raster graphic.[1] Rasterisation algorithms are presented in Chap. 7. The chapter also contains explanations of the most undesirable *aliasing effect* and ways to reduce it. The conversion into raster graphics usually takes place for all polygons in the visible frustum of the scene. Since at this point in the graphics pipeline it has not yet been taken into account which objects are transparent or hidden by other objects, the individual image points of the rasterised polygons are called *fragments*. The rasterisation stage thus provides a set of fragments.

[1] The primitive assembly step executed in the OpenGL graphics pipeline before rasterisation is described in more detail in Sect. 2.5.4.

Within the fragment processing, the illumination calculation usually takes place. Taking into account the lighting data, such as the type and position of the light sources and material properties of the objects, a colour value is determined for each of the fragments by special algorithms (see Chap. 9). To speed up the graphics processing or to achieve certain effects, *textures* can be applied to the objects. Often, the objects are covered by two-dimensional images (two-dimensional textures) to achieve a realistic representation of the scene. However, three-dimensional textures and geometry-changing textures are also possible (see Chap. 10).

The calculation of illumination during fragment processing for each fragment is a typical procedure in modern graphics pipelines. In principle, it is possible to perform the illumination calculation before rasterisation for the vertices instead of for the fragments. This approach can be found, for example, in the OpenGL fixed-function pipeline (see also Sect. 2.5).

Within the fragment processing, the combination of the fragments into pixels also takes place through visibility considerations (see Sect. 8.1). This calculation determines the final colour values of the pixels to be displayed on the screen, especially taking into account the mutual occlusion of the objects and transparent objects. The pixels with final colour values are written into the frame buffer.

In the OpenGL graphics pipeline, the polygon meshes after vertex processing or completely rendered images (as pixel data) can be returned to the graphics application. This enables further (iterative) calculations and thus the realisation of complex graphics applications.

The abstract OpenGL graphics pipeline shown in Fig. 1.2 is representative of the structure of a typical graphics pipeline, which can deviate from a concrete system. In addition, it is common in modern systems to execute as many parts as possible in a flexible and programmable way. In a programmable pipeline, programs—so-called *shaders*—are loaded and executed onto the GPU. Explanations on shaders are in Sects. 2.6, 2.9 and 2.10.

1.4 Objectives of This Book and Recommended Reading Order for the Sections

This book aims to teach the basics of computer graphics, supported by practical examples from a modern graphics programming environment. The programming of the rendering pipeline with the help of shaders will play a decisive role. To achieve easy access to graphics programming, the graphics programming interface *Open Graphics Library (OpenGL)* with the Java binding *Java OpenGL (JOGL)* to the Java programming language was selected (see Sect. 2.1).

OpenGL is a graphics programming interface that evolved greatly over the past decades due to its platform independence, the availability of increasingly powerful and cost-effective graphics hardware and its widespread use. In order to adapt this programming interface to the needs of the users, ever larger parts of the rendering pipeline became flexibly programmable directly on the GPU. On the other hand, there

was and still is the need to ensure that old graphics applications remain compatible with new versions when extending OpenGL. Reconciling these two opposing design goals is not always easy and usually leads to compromises. In addition, there are a large number of extensions to the OpenGL command and feature set, some of which are specific to certain manufacturers of graphics hardware. Against this background, a very powerful and extensive graphics programming interface has emerged with OpenGL, which is widely used and supported by drivers of all common graphics processors for the major operating systems.

Due to this complexity, familiarisation with OpenGL is not always easy for a novice in graphics programming and it requires a certain amount of time. The quick achievement of seemingly simple objectives may fail to materialise as the solution is far more complex than anticipated. This is quite normal and should be met with a healthy amount of perseverance.

Different readers have different prior knowledge, different interests and are different types of learners. This book is written for a reader[2] with no prior knowledge of computer graphics or OpenGL, who first wants to understand the minimal theoretical basics of OpenGL before starting practical graphics programming and deepening the knowledge of computer graphics. This type of reader is advised to read this book in the order of the chapters and sections, skipping over sections from Chap. 2 that seem too theoretical if necessary. Later, when more practical computer graphics experience is available, this chapter can serve as a reference section.

The practically oriented reader who wants to learn by programming examples is advised to read this chapter (Chapter 1). For the first steps with the OpenGL fixed-function pipeline, Sect. 2.7 can be read and the example in Sect. 2.8 can be used. The corresponding source code is available in the supplementary material to the online version of Chap. 2. Building on the understanding of the fixed-function pipeline, it is useful to read Sect. 2.9 and the explanations of the example in Sect. 2.10 for an introduction to the programmable pipeline. The source code of this example is available in the supplementary material to the online version of Chap. 2. Afterwards, selected sections of the subsequent chapters can be read and the referenced examples used according to interest. The OpenGL basics are available in Chap. 2 if required.

If a reader is only interested in the more theoretical basics of computer graphics and less in actual graphics programming, he should read the book in the order of the chapters and sections. In this case, the detailed explanations of the OpenGL examples can be skipped.

A reader with prior knowledge of computer graphics will read this book in the order of the chapters and sections, skipping the sections that cover known prior knowledge. If OpenGL is already known, Chap. 2 can be skipped entirely and used as a reference if needed.

Chapter 2 was deliberately designed to be a minimal introduction to OpenGL. However, care has been taken to cover all the necessary elements of the graphics

[2] It always refers equally to persons of all genders. To improve readability, the masculine form is used in this book.

pipeline in order to give the reader a comprehensive understanding without overwhelming him. For comprehensive and more extensive descriptions of this programming interface, please refer to the OpenGL specifications [9,10], the GLSL specification [1] and relevant books, such as the OpenGL SuperBible [11], the OpenGL Programming Guide [2] and the OpenGL Shading Language book [8].

1.5 Structure of This Book

The structure of this book is based on the abstract OpenGL graphics pipeline shown in Fig. 1.2. Therefore, the contents of almost all chapters of this book are already referred to in Sect. 1.3. This section contains a complete overview of the contents in the order of the following chapters. Section 1.4 gives recommendations for a reading order of the chapters and sections depending on the reading type.

Chapter 2 contains detailed explanations of the OpenGL programming interface and the individual processing steps of the two OpenGL graphics pipelines, which are necessary to develop your own OpenGL applications with the Java programming language. Simple examples are used to introduce modern graphics programming with *shaders* and the shader programming language *OpenGL Shading Language (GLSL)*.

In Chap. 3, the basic geometric objects used in OpenGL for modelling surfaces of three-dimensional objects are explained. In addition, this chapter contains a presentation of the different OpenGL methods for drawing these primitive basic shapes. The underlying concepts used in OpenGL are also used in other graphics systems.

Chapter 4 contains explanations of various modelling approaches for three-dimensional objects. The focus here is on modelling the surfaces of three-dimensional bodies.

In Chap. 5, the processing of the geometric data is explained, which mainly takes place during vertex processing. This includes in particular the geometric transformations in the viewing pipeline. In this step, among other operations, the position coordinates and normal vectors stored in the vertices are transformed.

Models for the representation of colours are used at various steps in the graphics pipeline. Chapter 6 contains the basics of the most important colour models as well as the basics of greyscale representation.

An important step in the graphics pipeline is the conversion of the vertex data, which is in a vector graphics representation, into a raster graphics representation (images of pixels). Procedures for this rasterisation are explained in Chap. 7. This chapter also covers the mostly undesirable aliasing effects caused by rasterisation and measures to reduce these effects.

A description of procedures for efficiently determining which parts of the scene are visible (visibility considerations) can be found in Chap. 8. This includes methods for clipping and culling.

During fragment processing, lighting models are usually used to create realistic lighting situations in a scene. The standard lighting model of computer graphics

is explained together with models for the so-called global illumination in Chap. 9. Effects such as shading, shadows and reflections are also discussed there.

Chapter 10 contains descriptions of methods for texturing the surfaces of three-dimensional objects. This allows objects to be covered with images (textures) or the application of special effects that change their appearance.

In the concluding Chap. 11, selected advanced topics are covered that go beyond the basics of computer graphics and lead to the popular fields of virtual reality (VR) and augmented reality (AR). Important for the realisation of interactive computer graphics applications are methods for object selection and the detection and handling of object collisions. Furthermore, methods for the realisation of fog effects or particle systems are presented, whereby realistic effects can be created. The immersion for virtual reality applications can be increased by auralising three-dimensional acoustic scenes. Therefore, a rather detailed section from this area has been included. Stereoscopic viewing of three-dimensional scenes, colloquially called "seeing in 3D", is an important factor in achieving immersion in a virtual world and thus facilitating the grasp of complex relationships.

1.6 Exercises

Exercise 1.1 Background of computer graphics: Please research the answers to the following questions:

(a) What is the purpose of computer graphics?
(b) On which computer was the first computer graphic created?
(c) Which film is considered the first fully computer-animated film?
(d) What is meant by the "uncanny valley" in the context of computer graphics? Explain this term.

Exercise 1.2 Main steps in computer graphics

(a) Name the main (abstract) steps in creating a computer graphic.
(b) What are the main processing steps in a graphics pipeline and which computations take place there?

Exercise 1.3 Requirements in computer graphics: Please research the answers to the following questions:

(a) Explain the difference between non-interactive and interactive computer graphics.
(b) From a computer science point of view, which requirements for computer graphics software systems are particularly important in order to be able to create computer graphics that are as realistic as possible? Differentiate your answers

according to systems for generating non-interactive and interactive computer graphics.

(c) Explain the difference between a real-time and non-real-time computer graphics.

(d) In computer graphics, what is meant by “hard real time” and “soft real time”?

References

1. J. Kessenich, D. Baldwin and R. Rost. *The OpenGL Shading Language, Version 4.60.6. 12 Dec 2018*. Specification. Abgerufen 2.5.2019. The Khronos Group Inc, 2018. URL: https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.pdf.
2. J. Kessenich, G. Sellers and D. Shreiner. *OpenGL Programming Guide*. 9th edition. Boston [u. a.]: Addison-Wesley, 2017.
3. F. Klawonn, V. Chekhtman and E. Janz. “Visual Inspection of Fuzzy Clustering Results”. In: *Advances in Soft Computing*. Ed. by Benítez J.M., Cordóon O., Hoffmann F. and Roy R. London: Springer, 2003, pp. 65–76.
4. M. Magnor. “3D-TV: Computergraphik zwischen virtueller und realer Welt”. In: *Informatik Spektrum* 27 (2004), pp. 497–503.
5. A. Nischwitz, M. Fischer, P. Haberäcker and G. Socher. *Computergrafik*. 4. Auflage. Computergrafik und Bildverarbeitung. Wiesbaden: Springer Vieweg, 2019.
6. D. P. Pretschner. “Medizinische Informatik - Virtuelle Medizin auf dem Vormarsch”. In: *Carolo-Wilhelmina Forschungsmagazin der Technis-chen Universitäat Braunschweig* 1 (2001). Jahrgang XXXVI, pp. 14–22.
7. F. Rehm, F. Klawonn and R. Kruse. “POLARMAP - Efficent Visualizationof High Dimensional Data”. In: *Information Visualization*. Ed. by E. Banissi, R.A. Burkhard, A. Ursyn, J.J. Zhang, M. Bannatyne, C. Maple, A.J. Cowell, G.Y. Tian and M. Hou. London: IEEE, 2006, pp. 731–740.
8. R. J. Rost and B. Licea-Kane. *OpenGL Shading Language*. 3rd edition. Upper Saddle River, NJ [u. a.]: Addison-Wesley, 2010.
9. M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 4.6 (Compatibility Profile) - October 22 2019*. Abgerufen 8.2.2021. The Khronos Group Inc, 2019. URL: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.compatibility.pdf.
10. M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 4.6 (Core Profile) - October 22, 2019)*. Abgerufen 8.2.2021. The Khronos Group Inc, 2019. URL: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf.
11. G. Sellers, S. Wright and N. Haemel. *OpenGL SuperBible*. 7th edition. New York: Addison-Wesley, 2016.
12. T. Soukup and I. Davidson. *Visual Data Mining*. New York: Wiley, 2002.
13. K. Tschumitschew, F. Klawonn, F. Höoppner and V. Kolodyazhniy. “Landscape Multidimensional Scaling”. In: *Advances in Intelligent Data Analysis VII*. Ed. by R. Berthold, J. Shawe-Taylor and N. Lavrač. Berlin: Springer, 2007, pp. 263–273.