Chapter 1

INTRODUCTION

1.1 Synthetic camera model

Suppose that a man is sitting in front of a computer calculating a function over its domain. In its simplest realization the program keeps printing out the samples of the domain with their respective function value in alphanumeric form. The user of the program who is mainly interested in the shape of the function has a very hard time reading all the data before they are scrolled off the screen, interpreting numbers like $1.2345e12$, and constructing the shape in his mind. He would prefer the computer to create the drawing of the function itself, and not to bother him with a list of mind-boggling floating point numbers. Assume that his dream comes true immediately, and the computer draws horizontal rectangles proportional to the function value, instead of printing them out in numeric form, making a histogram-like picture moving up as new values are generated. The user can now see the shape of a portion of the function. But is he satisfied? No. He also wants to have a look at the shape of larger parts of the function; he is not happy with the reduced accuracy caused by the limited resolution of the computer screen, meaning for example that two values that are very close to each other would share the same rectangle, and very large values generate rectangles that run off the screen. It irritates him that if he turns his head for just a second, he loses a great portion of the function, because it has already been scrolled off. The application of graphics instead of a numeric display has not solved many problems.

In order to satisfy our imaginary user, a different approach must be chosen; something more than the simple replacement of output commands by drawing primitives. The complete data should be generated and stored before being reviewed, thus making it possible to scale the rectangles adaptively in such a way that they would not run off the screen, and allowing for response to user control. Should the user desire to examine a very small change in the function for example, he should be able to zoom in on that region, and to move back and forth in the function reviewing that part as he wishes, etc.

This approach makes a clear distinction between the three main stages of the generation of the result, or image. These three stages can be identified as:

- Generation of the data
- Storage of the data
- Display of the data

The components of "data generation" and "data display" are not in any hierarchical relationship; "data display" routines are not called from the "data generation" module, but rather they respond to user commands and read out "data storage" actively if the output of the "data generation" is needed.

The concept which implements the above ideas is called the **synthetic camera model**, and is fundamental in most graphics systems today, especially in the case of three-dimensional graphics. The main components of the synthetic camera model are generalizations of the components in the previous example:

- **Modeling:**

  Modeling refers to a process whereby an internal representation of an imaginary world is built up in the memory of the computer. The modeler can either be an application program, or a user who develops the model by communicating with an appropriate software package. In both cases the model is defined by a finite number of applications of primitives selected from finite sets.

- **Virtual world representation:**

  This describes what the user intended to develop during the modeling phase. It can be modified by him, or can be analyzed by other programs capable of reading and interpreting it. In order to allow for easy modification and analysis by different methods, the model has to represent all relevant data stored in their natural dimensions. For example, in an architectural program, the height of a house has to be represented in meters, and not by the number of pixels, which would be the length of the image of the house on the screen. This use of natural metrics to represent data is known as the application of a **world coordinate system**. It does not necessarily mean that all objects are defined in the very same coordinate system. Sometimes it is more convenient to define an object in a separate, so-called **local coordinate system**, appropriate to its geometry. A transformation, associated with each object defining its relative position and orientation, is then used to arrange the objects in a common global **world coordinate system**.

- **Image synthesis:**

  Image synthesis is a special type of analysis of the internal model, when a photo is taken of the model by a "software camera". The position and direction of the camera are determined by the user, and the image thus generated is displayed on the computer screen. The user is in control of the camera parameters, lightsources and other studio objects. The ultimate objective of image synthesis is to provide the illusion of watching the real objects for the user of the computer system. Thus, the color sensation of an observer watching the artificial image generated by the graphics system about the internal model of a virtual world must be approximately equivalent to the color perception which would be obtained in the real world. The color perception of humans depends on the shape and the optical properties of the objects, on the illumination and on the properties and operation of the eye itself. In order to model this complex phenomenon both the physical-mathematical structure of the light-object interaction and the operation of the eye must be understood. Computer screens can produce controllable electromagnetic waves, or colored light for their

observers. The calculation and control of this light distribution are the basic tasks of image synthesis which uses an internal model of the objects with their optical properties, and implements the laws of physics and mathematics to simulate real world optical phenomena to a given accuracy. The exact simulation of the light perceived by the eye is impossible, since it would require endless computational process on the one hand, and the possible distributions which can be produced by computer screens are limited in contrast to the infinite variety of real world light distributions on the other hand. However, color perception can be approximated instead of having a completely accurate simulation. The accuracy of this approximation is determined by the ability of the eye to make the distinction between two light distributions. There are optical phenomena to which the eye is extremely sensitive, while others are poorly measured by it. (In fact, the structure of the human eye is a result of a long evolutionary process which aimed to increase the chance of survival of our ancestors in the harsh environment of the pre-historic times. Thus the eye has become sensitive to those phenomena which were essential from that point of view. Computer monitors have had no significant effect on this process yet.) Thus image synthesis must model accurately those phenomena which are relevant but it can make significant simplifications in simulating those features for which the eye is not really sensitive.

This book discusses only the image synthesis step. However, the other two components are reviewed briefly, not only for the reader's general information, but also that model dependent aspects of image generation may be understood.

1.2 Signal processing approach to graphics

From the information or signal processing point of view, the modeling and image synthesis steps of the synthetic camera model can be regarded as transformations (figure 1.1). Modeling maps the continuous world which the user intends to represent onto the discrete internal model. This is definitely an analog-digital conversion. The objective of image synthesis is the generation of the data analogous to a photo of the model. This data

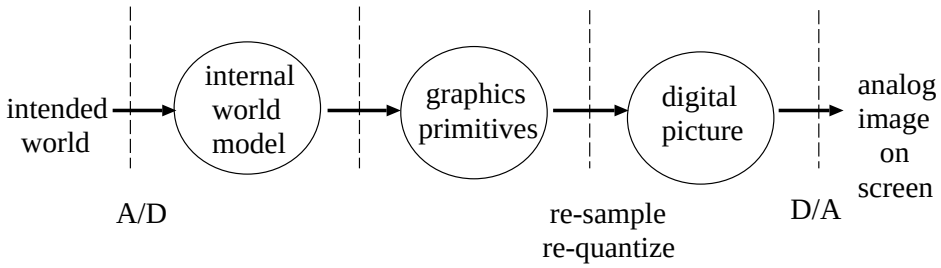

Figure 1.1: Dataflow model of computer graphics

is stored in the computer and is known as digital picture, which in turn is converted to analog signals and sent to the display screen by the computer's built in graphics hardware.

The digital picture represents the continuous two-dimensional image by finite, digital data; that is, it builds up the image from a finite number of building blocks. These building blocks can be either one-dimensional, such as line segments (called **vectors**), or two-dimensional, such as small rectangles of uniform color (called **pixels**). The word "*pixel*" is a composition of the words "*picture*" and "*element*".

The digital picture represented by either the set of line segments or the pixels must determine the image seen on the display screen. In **cathode ray tube (CRT)** display technology the color and the intensity of a display point are controlled by three electron beams (exciting red, green and blue phosphors respectively) scanning the surface of the display. Thus, the final stage of graphics systems must convert the digital image stored either in the form of vectors or pixels into analog voltage values used to control the electron beams of the display. This requires a digital-analog conversion.

1.3 Classification of graphics systems

The technique of implementing vectors as image building blocks is called **vector graphics**. By the application of a finite number of one-dimensional primitives only curves can be generated. Filled regions can only be approximated, and thus vector graphics is not suitable for realistic display of solid objects. One-dimensional objects, such as lines and characters defined as

a list of line segments and jumps between these segments, are represented by relative coordinates and stored in a so-called **display list** in a vector graphics system. The end coordinates of the line segments are interpreted as voltage values by a **vector generator** hardware which integrates these voltages for a given amount of time and controls the electron **beam** of the cathode ray tube by these integrated voltage values. The beam will draw the sequence of line segments in this way similarly to electronic oscilloscopes. Since if the surface of the display is excited by electrons then it can emit light only for a short amount of time, the electron beam must draw the image defined by the display list periodically about 30 times per second at least to produce **flicker-free** images.

Raster graphics, on the other hand, implements pixels, that is two-dimensional objects, as building blocks. The image of a raster display is formed by a **raster mesh** or **frame** which is composed of horizontal *raster* or **scan-lines** which in turn consist of rectangular pixels. The matrix of pixel data representing the entire screen area is stored in a memory called the **frame buffer**. These pixel values are used to modulate the intensities of the three electron beams which scan the display from left to right then from top to bottom. In contrast to vector systems where the display list controls the direction of the electron beams, in raster graphics systems the direction of the movement of the beams is fixed, the pixel data are responsible only for the modulation of their intensity. Since pixels cover a finite 2D area of the display, filled regions and surfaces pose no problem to raster based systems. The number of pixels is constant, thus the cycle time needed to avoid flickering does not depend on the complexity of the image unlike vector systems. Considering these advantages the superiority of raster systems is nowadays generally recognized, and it is these systems only that we shall be considering in this book.

When comparing vector and raster graphics systems, we have to mention two important disadvantages of raster systems. Raster graphics systems store the image in the form of a pixel array, thus normal image elements, such as polygons, lines, 3D surfaces, characters etc., must be transformed to this pixel form. This step is generally called the **scan conversion**, and it can easily be the bottleneck in high performance graphics systems. In addition to this, due to the limitations of the resolution and storage capability of the graphics hardware, the digital model has to be drastically re-sampled and re-quantized during image generation. Since the real or

intended world is continuous and has infinite bandwidth, the Shannon–Nyquist criterion of correct digital sampling cannot be guaranteed, causing artificial effects in the picture, which is called **aliasing**.

Note that scan-conversion of raster graphics systems transforms the geometric information represented by the display list to pixels that are stored in the frame buffer. Thus, in contrast to vector graphics, the display list is not needed for the periodic screen refresh.

1.4 Basic architecture of raster graphics systems

A simple raster graphics system architecture is shown in figure 1.2. The **display processor** unit is responsible for interfacing the frame buffer memory with the general part of the computer and taking and executing the drawing commands. In personal computers the functions of this display processor are realized by software components implemented in the form of a graphics library which calculates the pixel colors for higher level primitives. The programs of this graphics library are executed by the main CPU of the computer which accesses the frame buffer as a part of its operational memory.

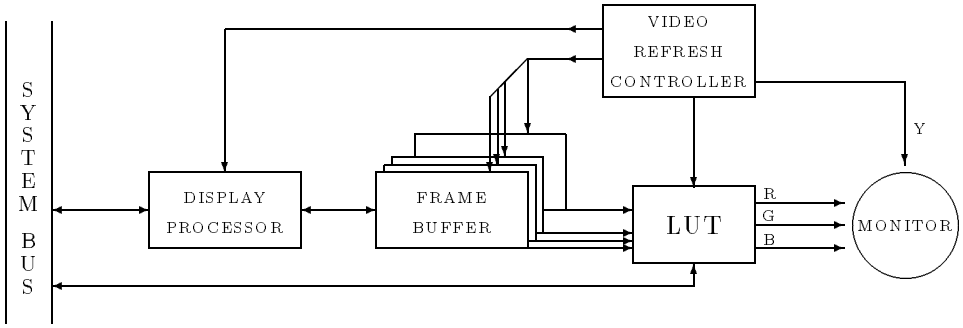

Figure 1.2: Raster system architecture with display processor

In advanced systems, however, a special purpose CPU is allocated to deal with pixel colors and to interface the frame buffer with the central CPU. This architecture increases the general performance because it relieves the central CPU of executing time consuming scan-conversion tasks on the one hand, and makes it possible to optimize this display processor for the graphics tasks on the other hand. The central CPU and the display processor communicate using drawing commands referring to higher level graphics primitives. The level of these primitives and the coordinate system where their geometry is defined is a design decision. These primitives are then transformed into pixel colors by the display processor having executed the image synthesis tasks including transformations, clipping, scan-conversion etc., and finally the generated pixel colors are written into the frame buffer memory. Display processors optimized for these graphics operations are called **graphics (co)processors**. Many current graphics processor chips combine the functions of the display processor with some of the functions of the video refresh controller, as for example the TMS 34010/20 [Tex88] from Texas Instruments and the HD63484 [Hit84] from Hitachi, allowing for compact graphics architectures. Other chips, such as i860 [Int89] from Intel, do not provide hardware support for screen refresh and timing, thus they must be supplemented by external refresh logic.

The **frame buffer memory** is a high-capacity, specially organized memory to store the digital image represented by the pixel matrix. For each elemental rectangle of the screen — that is for each pixel — a memory word is assigned in the frame buffer defining the color. Let the number of bits in this word be n. The value of n is 1 for **bi-level** or **black-and-white devices**, 4 for cheaper color and gray-shade systems, 8 for advanced personal computers, and 8, 12, 24 or 36 for graphics workstations. The color is determined by the intensity of the electron beams exciting the red, green and blue phosphors, thus this memory word must be used to modulate the intensity of these beams. There are two different alternatives to interpret the binary information in a memory word as modulation parameters for the beam intensities:

1. **True color mode** which breaks down the bits of memory word into three subfields; one for each color component. Let the number of bits used to represent red, green and blue intensities be n_r, n_g and n_b respectively, and assume that $n = n_r + n_g + n_b$ holds. The number of

producible pure red, green and blue colors are 2^{n_r}, 2^{n_g} and 2^{n_b}, and the number of all possible colors is 2^n. Since the human eye is less sensitive to blue colors than to the other two components, we usually select n_r, n_g and n_b so that: $n_r \approx n_g$ and $n_b \leq n_r$. True color mode displays distribute the available bits among the three color components in a static way, which has a disadvantage that the number of producible red colors, for instance, is still 2^{n_r} even if no other colors are to be shown on the display.

2. **Indexed color** or **pseudo color mode** which interprets the content of the frame buffer as indices into a color table called the **lookup table** or **LUT** for short. An entry in this lookup table contains three m-bit fields containing the intensities of red, green and blue components in this color. (**Gray-shade systems** have only a single field.) The typical value of m is 8. This lookup table is also a read-write memory whose content can be altered by the application program. Since the number of possible indices is 2^n, the number of simultaneously visible colors is still 2^n in indexed color mode, but these colors can be selected from a set of 2^{3m} colors. This selection is made by the proper control of the content of the lookup table. If $3m >> n$, this seems to be a significant advantage, thus the indexed color mode is very common in low-cost graphics subsystems where n is small. The lookup table must be read each time a pixel is sent to the display; that is, about every 10 nanoseconds in a high resolution display. Thus the lookup table must be made of very fast memory elements which have relatively small capacity. This makes the indexed color mode not only lose its comparative advantages when n is large, but also infeasible.

Concerning the color computation phase, the indexed color mode has another important disadvantage. When a color is generated and is being written into the frame buffer, it must be decided which color index would represent it in the most appropriate way. It generally requires a search of the lookup table and the comparison of the colors stored there with the calculated color, which is an unacceptable overhead. In special applications, such as 2D image synthesis and 3D image generation assuming very simple illumination models and only white lightsources, however, the potentially calculated colors of the primitives can be determined before the actual computation, and the actual colors can be replaced by the color indices in the

color calculation. In 2D graphics, for example, the "**visible color**" of an object is always the same as its "**own color**". (By definition the "own color" is the "visible color" when the object is lit by the sun or by an equivalent maximum intensity white lightsource.) Thus filling up the lookup table by the "own colors" of the potentially visible objects and replacing the color of the object by the index of the lookup table location where this color is stored makes it possible to use the indexed color mode. In 3D image synthesis, however, the "visible color" of an object is a complex function of the "own color" of the object, the properties of the lightsources and the camera, and the color of other objects because of the light reflection and refraction. This means that advanced 3D image generation systems usually apply true color mode, and therefore we shall only discuss true color systems in this book. Nevertheless, it must be mentioned that if the illumination models used are simplified to exclude non-diffuse reflection, refraction and shadows, and only white lightsources are allowed, then the visible color of an object will be some attenuated version of the own color. Having filled up the lookup table by the attenuated versions of the own color (the same hue and saturation but less intensity) of the potentially visible objects, and having replaced the color information by this attenuation factor in visibility computations, the color index can be calculated from the attenuation factor applying a simple linear transformation which maps [0..1] onto the range of indices corresponding to the attenuated color versions of the given object. This technique is used in *Tektronix* graphics terminals and workstations.

Even if true color mode is used — that is, the color is directly represented by the frame buffer data — the final transformation offered by the lookup tables can be useful because it can compensate for the non-linearities of the graphics monitors, known as γ-**distortion**. Since the individual color components must be compensated separately, this method requires the lookup table to be broken down into three parallelly addressable memory blocks, each of them is responsible for compensating a single color component. This method is called γ-**correction**.

As the display list for vector graphics, the frame buffer controls the intensity of the three electron beams, but now the surface of the display is scanned in the order of pixels left to right and from top to bottom in the screen. The hardware unit responsible for taking out the pixels from the frame buffer in this order, transforming them by the lookup tables and modulating the intensity of the electron beams is called the **video refresh**

controller. Since the intensity of the electron beams can be controlled by analog voltage signals, the color values represented digitally in the lookup tables or in the frame buffer must be converted to three analog signals, one for each color coordinate. This conversion is done by three digital-analog (D/A) converters. In addition to periodically refreshing the screen with the data from the frame buffer, video refresh controllers must also generate special synchronization signals for the monitors, which control the movement of the electron beam, specifying when it has to return to the left side of the screen to start scanning the next consecutive row (horizontal retrace) and when it has return to the upper left corner to start the next image (vertical retrace). In order to sustain the image on the screen, the video refresh controller must generate periodically scanning electron beams which excite the phosphors again before they start fading. The flicker-free display requires the screen to be refreshed about 60 times a second.

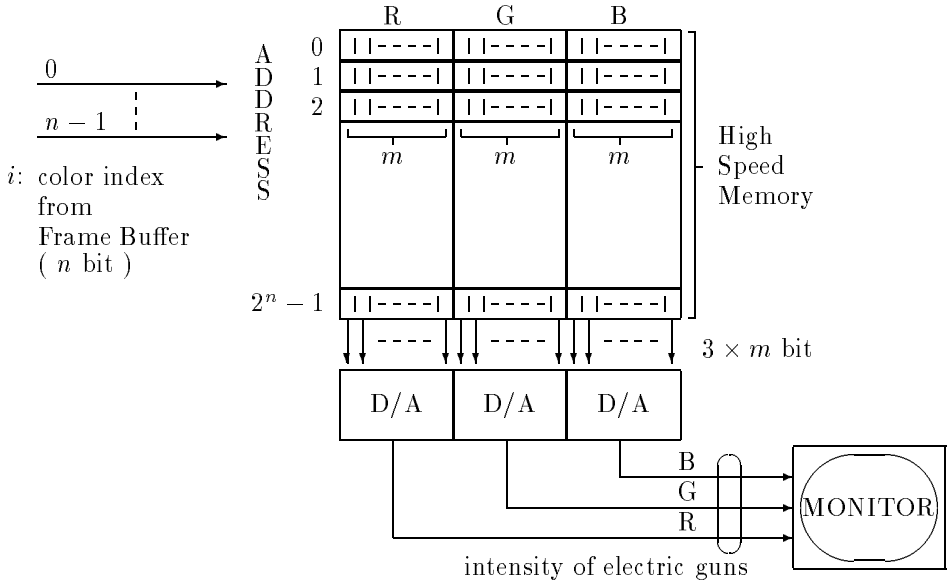

Figure 1.3: Organization of a video lookup table

The number of the pixel columns and rows is defined by the **resolution** of the graphics system. Typical resolutions are 640×480, 1024×768 for inexpensive systems and 1280×1024, 1600×1200 for advanced system. Thus an advanced workstation has over 10^6 pixels, which means that the time available to draw a single pixel, including reading it from the frame buffer and transforming it by the lookup table, is about 10 nanoseconds. This speed requirement can only be met by special hardware solutions in the video refresh controller and also by the parallel access of the frame buffer, because the required access time is much less than the cycle time of memory chips used in frame buffers. (The size of frame buffers — $1280 \times 1024 \times 24$ bits $\approx$ 3 Mbyte — does not allow for the application of high speed memory chips.) Fortunately, the parallelization of reading the pixel from the frame buffer is feasible because the display hardware needs the pixel data in a coherent way, that is, pixels are accessed one after the other left to right, and from top to bottom. Taking advantage of this property, when a pixel color is modulating the electron beams, the following pixels of the frame buffer row can be loaded into a shift register which in turn rolls out the pixels one-by-one at the required speed and without accessing the frame buffer. If the shift register is capable of storing N consecutive pixels, then the frequency of frame buffer accesses is decreased by N times.

A further problem arises from the fact that the frame buffer is a double access memory since the display processor writes new values into it, while the video refresh controller reads it to modulate the electron beams. Concurrent requests of the display processor and the refresh controller to read and write the frame buffer must be resolved by inhibiting one request while the other is being served. If N, the length of the shift register, is small, then the cycle time of read requests of the video refresh controller is comparable with the minimum cycle time of the memory chips, which literally leaves no time for display processor operations except during vertical and horizontal retrace. This was the reason that in early graphics systems the display processor was allowed to access the frame buffer just for a very small portion of time, which significantly decreased the drawing performance of the system. By increasing the length of the shift register, however, the time between refresh accesses can be extended, making it possible to include several drawing accesses between them. In current graphics systems, the shift registers that are integrated into the memory chips developed for frame buffer applications (called Video RAMs, or VRAMs) can hold a complete

pixel row, thus the refresh circuit of these systems needs to read the frame buffer only once in each row, letting the display processor access the frame buffer almost one hundred percent of the time.

As mentioned above, the video-refresh controller reads the content of the frame buffer periodically from left to right and from top to bottom of the screen. It uses counters to generate the consecutive pixel addresses. If the frame buffer is greater than the resolution of the screen, that is, only a portion of the pixels can be seen on the screen, the "left" and the "top" of the screen can be set dynamically by extending the counter network by "left" and "top" initialization registers. In early systems, these initialization registers were controlled to produce panning and scrolling effects on the display. Nowadays this method has less significance, since the display hardware is so fast that copying the whole frame buffer content to simulate scrolling and panning is also feasible.

There are two fundamental ways of refreshing the display: **interlaced**, and **non-interlaced**.

Interlaced refresh is used in broadcast television when the display refresh cycle is broken down into two phases, called *fields*, each lasting about 1/60 second, while a full refresh takes 1/30 second. All odd-numbered scan lines of the frame buffer are displayed in the first field, and all even-numbered lines in the second field. This method can reduce the speed requirements of the refresh logic, including frame buffer read, lookup transformation and digital-analog conversion, without significant flickering of images which consist of large homogeneous areas (as normal TV images do). However in CAD applications where, for example, one pixel wide horizontal lines possibly appear on the screen, this would cause bad flickering. TV images are continuously changing, while CAD systems allow the users to look at static images, and these static images even further emphasize the flickering effects. This is why advanced systems use non-interlaced refresh strategy exclusively, where every single refresh cycle generates all the pixels on the screen.

1.5 Image synthesis

Image synthesis is basically a transformation from model space to the color distribution of the display defined by the digital image. Its techniques

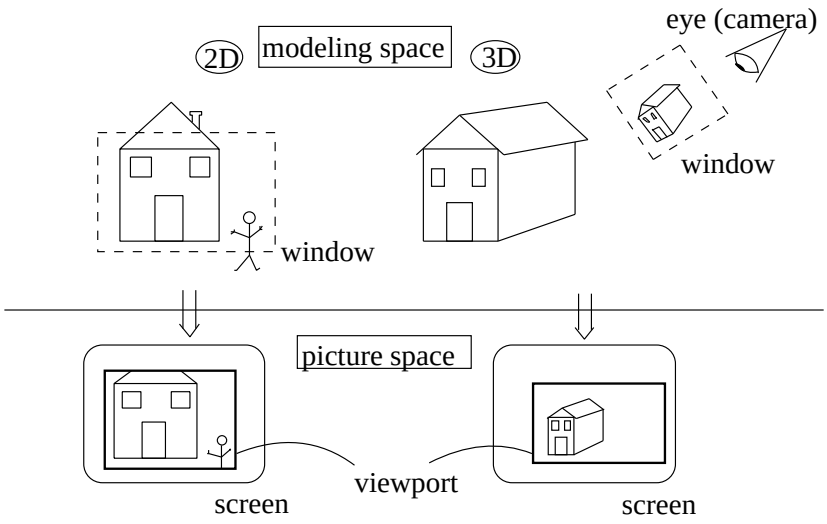

Figure 1.4: Comparison of 2D and 3D graphics

greatly depend on the space where the geometry of the internal model is represented, and we make a distinction between two- and three-dimensional graphics (2D or 3D for short) according to whether this space is two- or three-dimensional (see figure 1.4). In 2D this transformation starts by placing a rectangle, called a **2D window**, on a part of the plane of the 2D modeling space, then maps a part of the model enclosed by this rectangle to an also rectangular region of the display, called a **viewport**. In 3D graphics, the window rectangle is placed into the 3D space of the virtual world with arbitrary orientation, a **camera** or **eye** is placed behind the window, and the photo is taken by projecting the model onto the window plane having the camera as the center of projection, and ignoring those parts mapped outside the window rectangle. As in 2D, the photo is displayed in a viewport of the computer screen. Note that looking at the display only, it is not possible to decide if the picture has been generated by a two- or three-dimensional image generation method, since the resulting image is always two-dimensional. An exceptional case is the holographic display, but this topic is not covered in this book. On the other hand, a technique, called **stereovision**, is the proper combination of two normal images for the two eyes to emphasize the 3D illusion.

In both 2D and 3D graphics, the transformation from the model to the color distribution of the screen involves the following characteristic steps:

- **Object-primitive decomposition:** As has been emphasized, the internal world stores information in a natural way from the point of view of the modeling process so as to allow for easy modification and not to restrict the analysis methods used. In an architectural program, for example, a model of a house might contain building blocks such as a door, a chimney, a room etc. A general purpose image synthesis program, however, deals with primitives appropriate to its own internal algorithms such as line segments, polygons, parametric surfaces etc., and it cannot be expected to work directly on objects like doors, chimneys etc. This means that the very first step of the image generation process must be the decomposition of real objects used for modeling into primitives suitable for the image synthesis algorithms.

- **Modeling transformation:** Objects are defined in a variety of local coordinate systems: the nature of the system will depend on the nature of the object. Thus, to consider their relative position and orientation they have to be transferred to the global coordinate system by a transformation associated with them.

- **World-screen transformation:** Once the modeling transformation stage has been completed, the geometry of the model will be available in the global world coordinate system. However, the generated image is required in a coordinate system of the screen since eventually the color distribution of the screen has to be determined. This requires another geometric transformation which maps the 2D window onto the viewport in the case of 2D, but also involving projection in the case of 3D, since the dimension of the representation has to be reduced from three to two.

- **Clipping**: Given the intuitive process of taking photos in 2D and 3D graphics, it is obvious that the photo will only reproduce those portions of the model which lie inside the 2D window, or in the infinite pyramid defined by the camera as the apex, and the sides of the 3D window. The 3D infinite pyramid is usually limited to a finite frustum of pyramid to avoid overflows, thus forming a **front clipping plane**

and a **back clipping plane** parallel to the window. The process of removing those invisible parts that fall outside either the 2D window or the viewing frustum of pyramid is called clipping. It can either be carried out before the world-screen transformation, or else during the last step by inhibiting the modification of those screen regions which are outside the viewport. This latter process is called **scissoring**.

- **Visibility computations:** Window-screen transformations may project several objects onto the same point on the screen if they either overlap or if they are located behind each other. It should be decided which object's color is to be used to set the color of the display. In 3D models the object selected should be the object which hides others from the camera; i.e. of all those objects that project onto the same point in the window the one which is closest to the camera. In 2D no geometry information can be relied on to resolve the visibility problem but instead an extra parameter, called **priority**, is used to select which object will be visible. In both 2D and 3D the visibility computation is basically a sorting problem based on the distance from the eye in 3D, and on the priority in 2D.

- **Shading:** Having decided which object will be visible at a point on the display its color has to be calculated. This color calculation step is called shading. In 2D this step poses no problem because the object's **own color** should be used. An object's "own color" can be defined as the perceived color when only the sun, or an equivalent lightsource having the same energy distribution, illuminates the object. In 3D, however, the perceived color of an object is a complex function of the object's own color, the parameters of the lightsources and the reflections and refractions of the light. Theoretically the models and laws of geometric and physical optics can be relied on to solve this problem, but this would demand lengthy computational process. Thus approximations of the physical models are used instead. The degree of the approximation also defines the level of compromise in image generation speed and quality.

Comparing the tasks required by 2D and 3D image generation we can see that 3D graphics is more complex at every single stage, but the difference really becomes significant in visibility computations and especially in

shading. That is why so much of this book will be devoted to these two topics.

Image generation starts with the manipulation of objects in the virtual world model, later comes the transformation, clipping etc. of graphics primitives, and finally, in raster graphics systems, it deals with pixels whose constant color will approximate the continuous image.

Algorithms playing a part in image generation can thus be classified according to the basic type of data handled by them:

1. **Model decomposition algorithms** decompose the application oriented model into **graphics primitives** suitable for use by the subsequent algorithms.

2. **Geometric manipulations** include transformations and clipping, and may also include visibility and shading calculations. They work on graphics primitives independently of the resolution of the raster storage. By arbitrary definition all algorithms belong to this category which are independent of both the application objects and the raster resolution.

3. **Scan conversion algorithms** convert the graphics primitives into pixel representations, that is, they find those pixels, and may determine the colors of those pixels which approximate the given primitive.

4. **Pixel manipulations** deal with individual pixels and eventually write them to the raster memory (also called *frame buffer memory*).

1.6 Modeling and world representation

From the point of view of image synthesis, **modeling** is a necessary preliminary phase that generates a database called **virtual world representation**. Image synthesis operates on this database when taking "synthetic photos" of it. From the point of view of modeling on the other hand, image synthesis is only one possible analysis procedure that can be performed on the database produced. In industrial computer-aided design and manufacturing (CAD/CAM), for example, geometric models of products can be used to calculate their volume, mass, center of mass, or to generate a

sequence of commands for a numerically controlled (NC) machine in order to produce the desired form from real material, etc. Thus image synthesis cannot be treated separately from modeling, but rather its actual operation highly depends on the way of representing the **scene** (collection of objects) to be rendered. (This can be noticed when one meets such sentences in the description of rendering algorithms: "Assume that the objects are described by their bounding polygons ..." or "If the objects are represented by means of set operations performed on simple geometric forms, then the following can be done ...", etc.)

Image synthesis requires the following two sorts of information to be included in the virtual world representation:

1. *Geometric information.* No computer program can render an object without information about its shape. The shape of the objects must be represented by numbers in the computer memory. The field of **geometric modeling** (or solid modeling, shape modeling) draws on many branches of mathematics (geometry, computer science, algebra). It is a complicated subject in its own right. There is not sufficient space in this book to fully acquaint the reader with this field, only some basic notions are surveyed in this section.

2. *Material properties.* The image depends not only on the geometry of the scene but also on those properties of the objects which influence the interaction of the light between them and the lightsources and the camera. Modeling these properties implies the characterization of the object surfaces and interiors from an optical point of view and the modeling of light itself. These aspects of image synthesis are explained in chapter 3 (on physical modeling of 3D image synthesis).

1.6.1 Geometric modeling

The terminology proposed by Requicha [Req80] still seems to be general enough to describe and characterize geometric modeling schemes and systems. This will be used in this brief survey.

Geometric and graphics algorithms manipulate *data structures* which (may) *represent* physical solids. Let D be some domain of data structures. We say that a data structure $d \in D$ represents a physical solid if there is a

mapping m: $D \rightarrow E^3$ (E^3 is the 3D Euclidean space), for which $m(d)$ *models* a physical solid. A subset of E^3 models a physical solid if its shape can be produced from some real material. Subsets of E^3 which model physical solids are called **abstract solids**. The class of abstract solids is very small compared to the class of all subsets of E^3.

Usually the following properties are required of abstract solids and representation methods:

1. *Homogeneous 3-dimensionality.* The solid must have an interior of positive volume and must not have isolated or "dangling" (lower dimensional) portions.

2. *Finiteness.* It must occupy a finite portion of space.

3. *Closure under certain Boolean operations.* Operations that model working on the solid (adding or removing material) must produce other abstract solids.

4. *Finite describability.* The data structure describing an abstract solid must have a finite extent in order to fit into the computer's memory.

5. *Boundary determinism.* The boundary of the solid must unambiguously determine which points of E^3 belong to the solid.

6. (*Realizability.* The shape of the solid should be suitable for production from real material. Note that this property is not required for producing virtual reality.)

The mathematical implications of the above properties are the following. Property 1 requires the abstract solid to belong to the class of **regular sets**. In order to define regular sets in a self-contained manner, some standard notions of set theory (or set theoretical topology) must be recalled here [KM76], [Men75], [Sim63]. A neighborhood of a point p, denoted by $N(p)$, can be any set for which $p \in N(p)$. For any set S, its complement (cS), interior (iS), closure (kS) and boundary (bS) are defined using the notion of neighborhood:

$$\begin{array}{rcl} cS & = & \{p \mid p \notin S\}, \\ iS & = & \{p \mid \exists N(p): N(p) \subset S\}, \\ kS & = & \{p \mid \forall N(p): \exists q \in N(p): q \in S\}, \\ bS & = & \{p \mid p \in kS \text{ and } p \in kcS\}. \end{array} \tag{1.1}$$

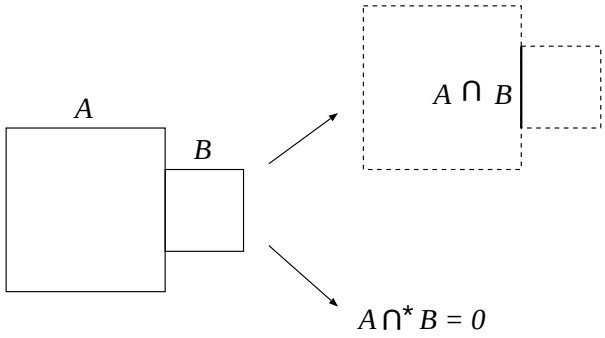

Figure 1.5: An example when regularized set operation ($\cap^*$) is necessary

Then a set S is defined as regular, if:

$$S = kiS. \tag{1.2}$$

Property 2 implies that the solid is *bounded*, that is, it can be enclosed by a sphere of finite volume. Property 3 requires the introduction of the **regularized set operations**. They are derived from the ordinary set operations ($\cup, \cap, \setminus$ and the complement c) by abandoning non-3D ("dangling") portions of the resulting set. Consider, for example, the situation sketched in figure 1.5, where two cubes, A and B, share a common face, and their intersection $A \cap B$ is taken. If $\cap$ is the ordinary set-theoretical intersection operation, then the result is a dangling face which cannot correspond to a real 3D object. The regularized intersection operation ($\cap^*$) should give the empty set in this case. Generally if $\circ$ is a binary set operation ($\cup, \cap$ or $\setminus$) in the usual sense, then its regularized version $\circ^*$ is defined as:

$$A \circ^* B = ki\,(A \circ B)\,. \tag{1.3}$$

The unary complementing operation can be regularized in a similar way:

$$c^*A = ki\,(cA)\,. \tag{1.4}$$

The regular subsets of E^3 together with the regularized set operations form a Boolean algebra [Req80]. Regularized set operations are of great importance in some representation schemes (see CSG schemes in subsection

1.6.2). Property 4 implies that the shape of the solids is defined by some formula (or a finite system of formulae) F: those and only those points of E^3 which satisfy F belong to the solid. Property 5 has importance when the solid is defined by its boundary because this boundary must be valid (see B-rep schemes in subsection 1.6.2). Property 6 requires that the abstract solid is a *semianalytic* set. It poses constraints on the formula F. A function f: $E^3 \to R$ is said to be analytic (in a domain) if $f(x, y, z)$ can be expanded in a convergent power series about each point (of the domain). A subset of analytic functions is the set of *algebraic* functions which are polynomials (of finite degree) in the coordinates x, y, z. A set is semianalytic (semialgebraic) if it can be expressed as a finite Boolean combination (using the set operations $\cup, \cap, \setminus$ and c or their regularized version) of sets of the form:

$$S_i = \{x, y, z\colon\ f_i(x, y, z) \leq 0\}\,, \tag{1.5}$$

where the functions f_i are analytic (algebraic). In most practical cases, semialgebraic sets give enough freedom in shape design.

The summary of this subsection is that suitable models for solids are subsets of E^3 that are *bounded, closed, regular* and *semianalytic (semialgebraic)*. Such sets are called **r-sets**.

1.6.2 Example representation schemes

A **representation scheme** is the correspondence between a data structure and the abstract solid it represents. A given representation scheme is also a given method for establishing this connection. Although there are several such methods used in practical *geometric modeling systems* only the two most important are surveyed here.

Boundary representations (B-rep)

The most straightforward way of representing an object is describing its boundary. Students if asked about how they would represent a geometric form by a computer usually choose this way.

A solid can be really well represented by describing its boundary. The boundary of the solid is usually segmented into a finite number of bounded subsets called **face**s or **patch**es and each face is represented separately. In the case of polyhedra, for example, the faces are planar polygons and hence

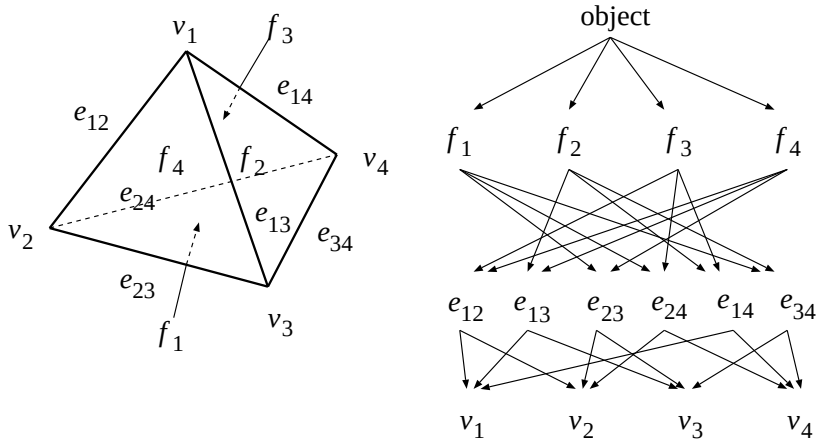

Figure 1.6: B-rep scheme for a tetrahedron

can be represented by their bounding edges and vertices. Furthermore, since the edges are straight line segments, they can be represented by their bounding vertices. Figure 1.6 shows a tetrahedron and a possible B-rep scheme. The representation is a directed graph containing object, face, edge and vertex nodes. Note that although only the lowest level nodes (the vertex nodes) carry geometric information and the others contain only "pure topological' information in this case, it is not always true, since in the general case, when the shape of the solid can be arbitrarily curved (sculptured), the face and edge nodes must also contain shape information.

The *validity* of a B-rep scheme (cf. property 5 in the beginning of this subsection) requires the scheme to meet certain conditions. We distinguish two types of validity conditions:

1. *combinatorial (topological)* conditions: (1) each edge must have precisely two vertices; (2) each edge must belong to an even number of faces;

2. *metric (geometric)* conditions: (1) each vertex must represent a distinct point of E^3; (2) edges must either be disjoint or intersect at a common vertex; (3) faces must either be disjoint or intersect at a common edge or vertex.

These conditions do not exclude the representation of so-called **non-manifold** objects. Let a solid be denoted by S and its boundary by ∂S. The solid S is said to be **manifold** if each of its boundary points $p \in \partial S$ has a neighborhood $N(p)$ (with positive volume) for which the set $N(p) \cap \partial S$ (the neighborhood of p on the boundary) is *homeomorphic* to a disk. (Two sets are homeomorphic if there exists a continuous one-to-one mapping which transforms one into the other.) The union of two cubes sharing a common edge is a typical example of a non-manifold object, since each point on the common edge becomes a "non-manifold point". If only two faces are allowed to meet at each edge (cf. combinatorial condition (2) above), then the scheme is able to represent only manifold objects. The **winged edge** data structure, introduced by Baumgart [Bau72], is a boundary representation scheme which is capable of representing manifold objects and inherently supports the automatic examination of the above listed combinatorial validity conditions. The same data structure is known as **doubly connected edge list (DCEL)** in the context of computational geometry, and is described in section 6.7.

Constructive solid geometry (CSG) representations

Constructive solid geometry (CSG) includes a family of schemes that represent solids as Boolean constructions or combinations of solid components via the regularized set operations $(\cup^*, \cap^*, \setminus^*, c^*)$. CSG representations are binary trees. See the example shown in figure 1.7. Internal (nonterminal) nodes represent set operations and leaf (terminal) nodes represent subsets (r-sets) of E^3. Leaf objects are also known as *primitives*. They are usually simple bounded geometric forms such as blocks, spheres, cylinders, cones or unbounded halfspaces (defined by formulae such as $f(x, y, z) \leq 0$). A more general form of the CSG-tree is when the nonterminal nodes represent either set operations or rigid motions (orientation and translation transformations) and the terminal nodes represent either primitives or the parameters of rigid motions.

The *validity* of CSG-trees poses a smaller problem than that of B-rep schemes: if the primitives are r-sets (that is general unbounded halfspaces are not allowed), for example, then the tree always represents a valid solid.

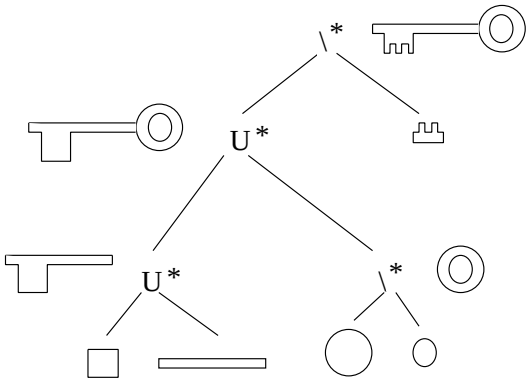

Figure 1.7: A CSG scheme

Note that a B-rep model is usually closer to being ready for image synthesis than a CSG representation since primarily surfaces can be drawn and not volumes. Transforming a CSG model into a (approximate) B-rep scheme will be discussed in section 4.2.2 in the context of model decomposition.