

SAMPLE

**Computer Graphics**

## Chapter One

### Introduction to Computer Graphics

#### Unit Introduction

Computer graphics is the art and science of communicating graphically through a computer's presentation and interactive tools (Foley et al., 1994). The human-to-computer route is typically facilitated via devices such as the keyboard, mouse, keypad, gaming joystick, or touch-sensitive interface in terms of the visual aspect of communication. But, even this is starting to alter: Innovative platforms built around computer visual algorithms subjected to film or depth-camera inputs facilitate the transmission of visual information back to a computer. For the computer-to-user path, however, the final recipients of the communication are people; hence, how humans interpret pictures is crucial in developing graphics programs; characteristics that humans disregard are not required to be displayed. Computer graphics is a multidisciplinary field that requires expertise in mathematics, arithmetic, perception, human interaction, technology, graphic design, and artistry (Foley et al., 1996). Researchers simulate light using science and do computations for animations. Mathematics is used to analyze geometry. Human sensory capacities determine bandwidth selection; humans are reluctant to spend time drawing items that are unlikely to be observed (Goldman, 2009). In order to optimize the distribution of network, storage, and processing cycles, designers employ engineering. Combining graphic design and artwork with human interaction maximizes the effectiveness of computer-to-human communication (Bartels et al., 1995). In this chapter, several application fields are explored, including the operation of traditional graphics systems and the impact of all of these fields on computer graphics development.

A limited definition of computer graphics might indicate that it relates to generating a depiction of a specific scene perspective from a template of the scene's artifacts and a concept of the light radiated into the scene (Enderle et al., 2012). In this perspective, one could assert that graphics is only a glorified form of amplification: One amplifies the light rays by the reflectivities of the objects being photographed to determine the light departing the surface areas of those objects and then does so again (handling the surface areas as new light sources and iteratively asserting the light-transport procedure) to determine all light that finally hits the camera. (This strategy is impractical in reality, but the concept persists.) In comparison, computer vision is equivalent to factorization; provided a viewing of a scene, it is responsible for calculating the scene's lighting and/or features (which a visual system may then "amplify/multiply" to replicate this similar visual) (Stollnitz et al., 1996). In reality, the vision system never answers the challenge described because it generally makes predictions about the scene, the illumination, or both. It might have numerous views of the image from various cameras or several views of the picture from the same camera at various times. In reality, graphics are

significantly more complex than the multiplication-based process of producing a scene, just like vision is more complex than factoring. The majority of contemporary studies in graphics focus on ways to generate geometric prototypes, strategies for portraying surface reflectance (and subsurface reflectance, and reflectance's of actively partaking media including haze and smoke, and so forth.), the animated feature of scenes by physical processes and by simplifications of those principles, the control of motion graphics, interplay with virtual elements, the innovation of non-photorealistic depictions, and, in recent times, a growing technological integration (Stork, 2009).

In consequence, the fields of computer visual and artificial vision are converging. Consider Raskar's research on a non-photorealistic lens as an illustration. The camera captures numerous images of a particular scene, each lighted by a separate flash unit. Using computer vision algorithms, contours and certain basic geometric features of scene components can be estimated based on these photos. Figure 1.1 depicts how they may be employed to generate a non-photorealistic depiction of the scene (Lewin, 1967).



***Figure 1.1. Image of the original scene (left) and the new rendering scene (right) (Source: John F. Hughes, Creative Commons License)***

Realistic picture capturing and rendering are emphasized in this book since it is where the discipline of graphics has achieved the most progress, demonstrating a magnificent implementation of relatively modern computer engineering to the modeling of comparatively old science models (Angel & Shreiner, 2011). Yet, there is a lot more to graphics beyond capturing and producing realistic images. Animation and interactivity, for example, are as essential, and these disciplines are mentioned in several sections in this text in addition to devoting entire chapters to them. How has progress in non-

simulation sectors been relatively difficult to attain? Maybe because these fields are qualitative and need mathematical frameworks like those supplied by physics, which are essentially quantitative.

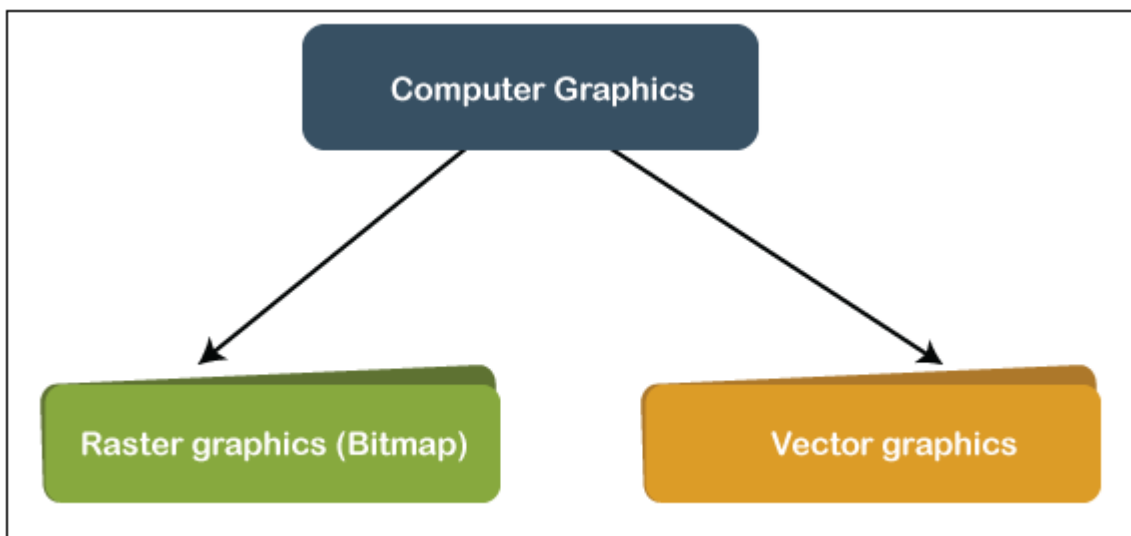
**Did you know?**

The first computer graphics were created in the early 1960s using a computer called the IBM 7090.

The phrase computer graphics refers to the creation and alteration of visuals using a computer. This book covers the computational and mathematical methods that may be employed to generate a variety of visuals, including photorealistic visual features, good technical representations, and stunning computer movies. Graphics can be either two-dimensional or three-dimensional, and pictures can be entirely fabricated or created by modifying photos (Salmon & Slater, 1987). This book discusses foundational mathematics and algorithms, particularly those used to generate synthetic imagery of three-dimensional (3-D) scenes and objects.

Creating computer graphics needs knowledge of dedicated hardware, file types, and typically a graphics application programming interface (API). The details of this expertise are a shifting objective because graphics is a continuously expanding discipline (Balreira et al., 2017). Hence, this book tries to prevent de-reliance on specialized hardware or API. It is recommended that readers complement the book with information pertinent to their computer hardware and software environments. Thankfully, the graphics industry has good common terms and ideas for the topic in this work to transfer well to most situations.

This chapter introduces certain basic terminology and presents some historical context and data resources relevant to computer graphics.



*Figure 1.2. Illustration of the types of computer graphics (Source: TAE, Creative Commons License)*

## **Learning Objectives**

After this section, readers will understand the following:

1. The fundamental principles of computer graphics
2. An overview of the development and study of the discipline of computer graphics
3. Fundamental divisions of computer graphics
4. Diverse implementations of computer graphics advanced technologies
5. Graphics API and visuals pipeline design concepts
6. Developing and programming strategies for a graphics software

## **Key terms**

1. Animation
2. Computer Graphics
3. Graphics Pipeline
4. GUI
5. GPU
6. Manipulate
7. Modeling
8. Pixels
9. Rendering
10. Visualization

### **1.1. Brief History**

Graphics study has maintained a goal-oriented route, but the aim has shifted with time; the initial scientists operated in an environment with little processor capacity (Carson et al., 1998). Therefore they regularly made decisions that produced results rapidly and readily. Initial attempts were split between attempting to create illustrations (such as schematics) and visuals (e.g., photorealistic visuals). Several presumptions have been made within every instance, typically as a compromise to existing processor capacity and display capabilities. When a solitary display was priced as much as or

greater than an inventor's pay, every image exhibited had to be of significant worth (Fallon, 1998). When showing a few dozen shapes took minutes or hours, it did make sense to approximate curvatures with comparatively few vertices. Since processor performance was expressed in MIPS (millions of instructions/second) but pictures comprised 250,000 to 500,000 pixels, it was impractical to conduct many computations for each pixel. Throughout the 1960s and 1970s, several institutes had only one graphic display at maximum. Typical summarizing presumptions were that all material objects reflected light similarly to flat enamel paint (even though more- advanced reflectance designs were utilized in a handful of computer systems), that illumination either lighted a surface straightforwardly or rebounded across the scene frequently that it ultimately provided an overall light source that illuminated objects even though they were not actively illuminated, and that the shades at the inner locations of any polygon could be implied from the appearance of the exterior polygon point (Bender et al., 2014).

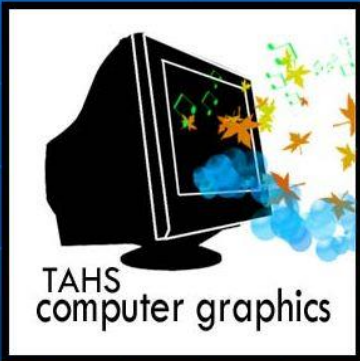
Progressively, more complex models—of contour, illumination, and reflectance—were introduced (Potts, 1975). However, even currently, the prevalent model for characterizing the light in an image contains the term "ambient," which refers to a specific degree of light that is "everywhere in the scene" without the need for a clear source, guaranteeing that all observable objects are at least partially lighted. An informal phrase was introduced to describe features of light transmissions, such as inter-object reflections, which could not be directly estimated using computers of the early 1970s; yet, it is still in use nowadays. Whereas many books consider the historical origins of light travel, this book will employ a different strategy and explain the ideal scenario (the physical modeling of light travel), how existing algorithms estimate this ideal, how certain previous methods did likewise, and how the remnants of these simplifications continue to be used in daily practices. The benefit of the initial look is that it enables one to experiment with rendering and modeling scenes before understanding how light is reflected (Salomon, 2006). With the transition from vector to raster devices, such as CRTs or Liquid crystal displays, in the early 1980s, and with progressively but gradually growing resolution, shape (the physical measurements of the showcases), and dynamic spectrum (the proportion of the brightest to the blindest possible image pixels) over the last twenty-five years, graphics displays have vastly improved over the years. Graphics processor efficiency has also increased in conjunction with Moore's Law (KRULL, 1994). Furthermore, the design of graphics processors is becoming progressively parallel; the extent to which this can go is disputed.

In addition to constant growth, there have been significant breakthroughs in processor and display technology: One of them was the transition from vector devices towards raster displays and their quick penetration of the mini-computer and workstation markets (Woodwark, 1991). In addition, the emergence of commodity market graphics cards (as well as their corresponding operating systems) enabled the development of software that ran on various computers. Somewhat concurrent with the widespread deployment of raster displays was the deployment of Xerox PARC's WIMP GUIs. At that

point, graphics transitioned from experimental research equipment to an unacknowledged aspect of daily computer interaction.

Notable is the advent of the programmable or configurable graphics card as the final advancement. Rather than transmitting polygons or graphics to a graphics card, the software can now submit tiny programs that describe how following polygons and images will be processed en route to the screen (Machover, 1978). All these "shaders" enabled the generation of entirely new features without needing increased CPU time (even though the GPU—Graphics Processor Unit—was straining very tough!) (Machover, 1997). One can predict future significant advances in graphics processing power in the coming decades.

# The History of Computer Graphics?



- **1950:** Ben Laposky created the first graphic images, an Oscilloscope, generated by an electronic (analog) machine. The image was produced by manipulating electronic beams and recording them onto high-speed film.
- **1960:** William Fetter coins the computer graphics to describe new design methods.
- **1972:** Nolan Kay Bushnell – Pong, video arcade game.
- **1985:**
  - Pixar Animation Studios – “Luxo Jr.”, 1989, “Tin toy”
  - NES – Nintendo home game system

*Figure 1.3. Representation of the historical events in the field of computer graphics (Source: Tyanni Niles, Creative Commons License)*

## 1.2. Graphics Areas

Placing categories on any topic is risky, yet most computer graphics professionals will accept the following broad areas:

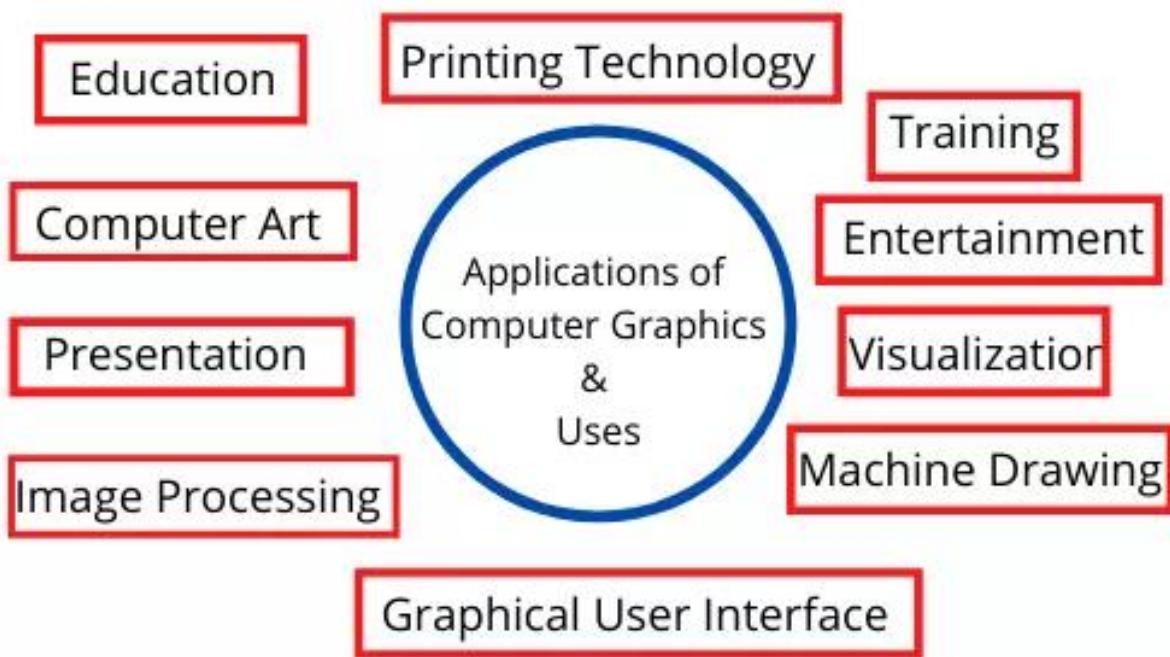
- Modeling:** This involves the mathematical description of physical and visual features in a computer-storable format (Ihaka & Gentleman, 1996). For instance, a cappuccino mug could be defined as a set of arranged 3D elements with an interpolation rule connecting the dots and a reflective model describing how illumination interacts with the cup.
- Rendering:** It generates shaded visuals from three-dimensional computer models (Shi & Pan, 2001).

- iii. **Animation:** It is a method that employs repetitions of pictures to give the appearance of motion. Animation employs modeling and rendering but includes the crucial issue of motion over the period, which is not typically addressed in fundamental modeling & rendering.

Numerous more fields incorporate computer graphics, and whether these constitute core graphics areas is a matter of perspective. Each of these will be mentioned briefly in the book. These connected areas include the ones that follow:

- i. **User interaction:** This relates to the interface amongst input tools such as the mouse and keyboard, the program, the user's visual response, and other forms of sensory responses (Potts, 1975). Traditionally, this field has been linked with graphics partly because graphics scientists had instant access to today's common input/output devices.
- ii. **Virtual reality:** It aims to engage the viewer in a 3D simulated space. Generally, this demands stereo graphics plus head motion responses (Aristidou et al., 2018). Furthermore, sound and force response should be present in the virtual space. Because to the fact that this field involves sophisticated 3D graphics and screen technology, it is frequently related to graphics.
- iii. **Visualization:** It aims to provide people with visual insights into complex data. Frequently, graphic difficulties must be handled in a situation involving visualization.
- iv. **Image processing:** It pertains to editing 2D pictures and is utilized in both the graphic and sight fields (Spitz & Requicha, 2000).
- v. **3D scanning:** It employs range-finding methods for creating 3D models with accurate dimensions (Enderle et al., 2013). These models help develop aesthetically rich visuals, and their processing typically involves graphics algorithms.
- vi. **Computational photography:** It is the application of computer graphics, machine vision, and methods for image processing to create new imaging modalities for recording objects, sceneries, and settings (Gomes & Velho, 1995).





*Figure 1.4. Illustration of applications of computer graphics (Source: Avinash Pandey, Creative Commons License)*

### 1.3. Major Applications

Almost every activity can take the utilization of computer graphics. However, the following sectors are the biggest users of computer graphics tech:

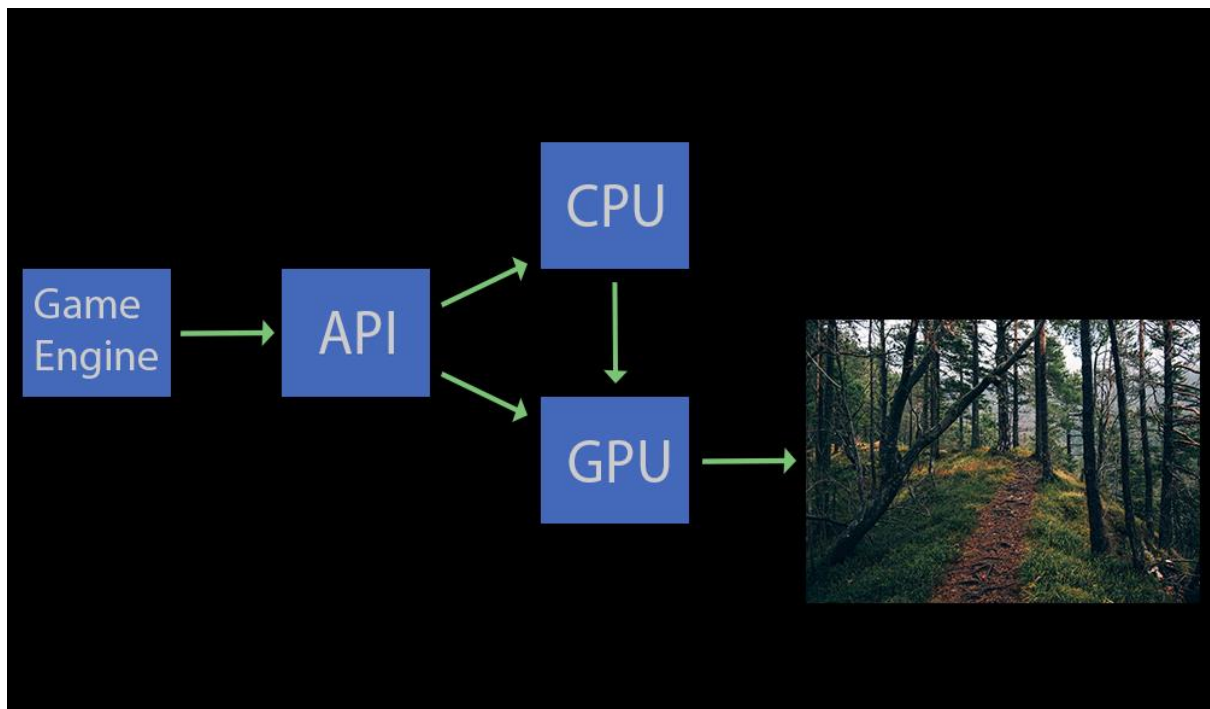
- i. Video games utilize more complex 3D modeling and processing techniques.
- ii. Animations are frequently created straight from 3D graphics. Numerous classic 2D animations employ 3D-rendered backdrops, which enable a consistently altering perspective without requiring much artistic effort (Papathomas et al., 1988).
- iii. Almost every form of computer graphics is used for visual effects. Nearly every contemporary movie contains digital blending to combine independently filmed landscapes and illustrations. In addition, some films use 3D models and animations to produce artificial scenes, artifacts, and even individuals that most spectators would never guess are not genuine.
- iv. Animated movies utilize several identical methods as visual effects, however, without always aiming for realistic pictures,
- v. Computer-aided design (CAD) and computer-aided manufacturing (CAM). These fields employ computing technology for generating parts and assemblies on the computer and afterward direct the production procedure that uses these virtual models (Schneider, 1978). Many structural components, for instance, are designed in a three-dimensional computer modeling program and then autonomously manufactured on a computer-controlled machining process.

- vi. Simulation can be viewed as realistic video games. An aviation simulator, for instance, employs advanced 3D visuals to recreate the sensation of piloting an airplane (Stollnitz et al., 1996). Such simulators can be particularly valuable for basic training in safety-critical areas, such as automobiles, and for situation preparation for advanced users, such as instruction for certain fire-fighting circumstances that are too expensive or hazardous to build practically.
- vii. Medical imaging generates useful visuals by scanning patient information. For instance, a computed tomography (CT) database consists of a vast 3D rectangular grid of density values. Computer graphics technology generates colored representations that assist physicians in extracting the most pertinent data from such datasets.
- viii. Information visualization produces images of data that are not always "natural" in appearance (Guo, 2014). For instance, the temporal pattern of the market of 10 distinct companies cannot be visually represented, yet creative graphing methods can assist users in pattern recognition in such data.

#### **1.4. Graphics APIs**

Interacting with a graphics API is a crucial component of utilizing graphics libraries. An application program interface (API) is a standardized function group that conducts various associated operations. A graphics API is a standard group of functions that execute fundamental activities such as rendering images and 3D objects into display windows (Hitchner & Sowizral, 2000).

Every graphics application needs to be able to utilize two interrelated APIs: a graphics API for visual outputs and a user-interface API for receiving input from the user. There are currently two prevalent frameworks for graphics and user interface APIs (Chen & Cheng, 2005). The foremost is an integrated approach, typified by Java, in which the graphics interface and the user-interface tools are linked, portable modules that are completely standard and maintained as a component of the language. The latter is exemplified by Direct3D and OpenGL, in which the drawing instructions are a software library component related to a programming language such as C++. The user-interface program is a separate entity that may differ from one system to another (Wilkins, 2001). In this method, it is hard to create portable code; however, it may be feasible to employ a highly portable library layer to enclose system-specific user-interface code for small projects.



*Figure 1.5. Graphical Representation of how an API works (Source: Tech Junkie, Creative Commons License).*

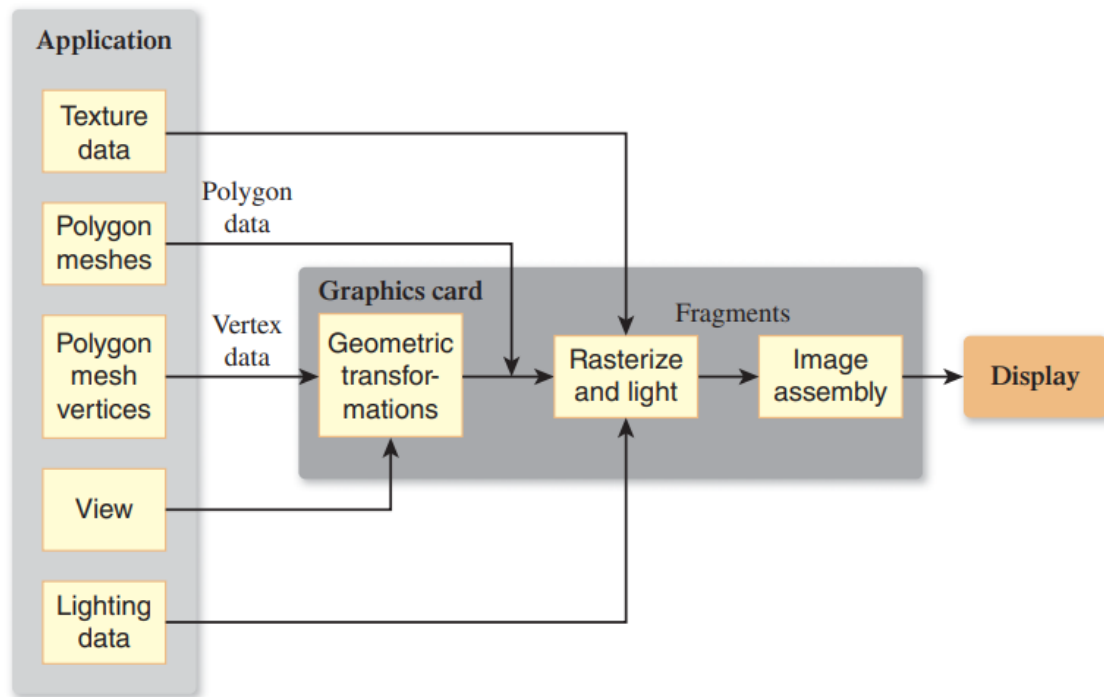
### 1.5. Graphics Pipeline

Nowadays, every desktop pc has a robust 3D graphics pipeline. This is a specialized software/hardware system for drawing 3D blocks in aspect quickly. These systems are often optimized for handling three-dimensional triangles with common vertices. The fundamental pipeline functions translate the 3D vertex positions to the 2D display coordinates and color the triangles to look real and in the correct back-to-front sequence (Blinn, 1993).

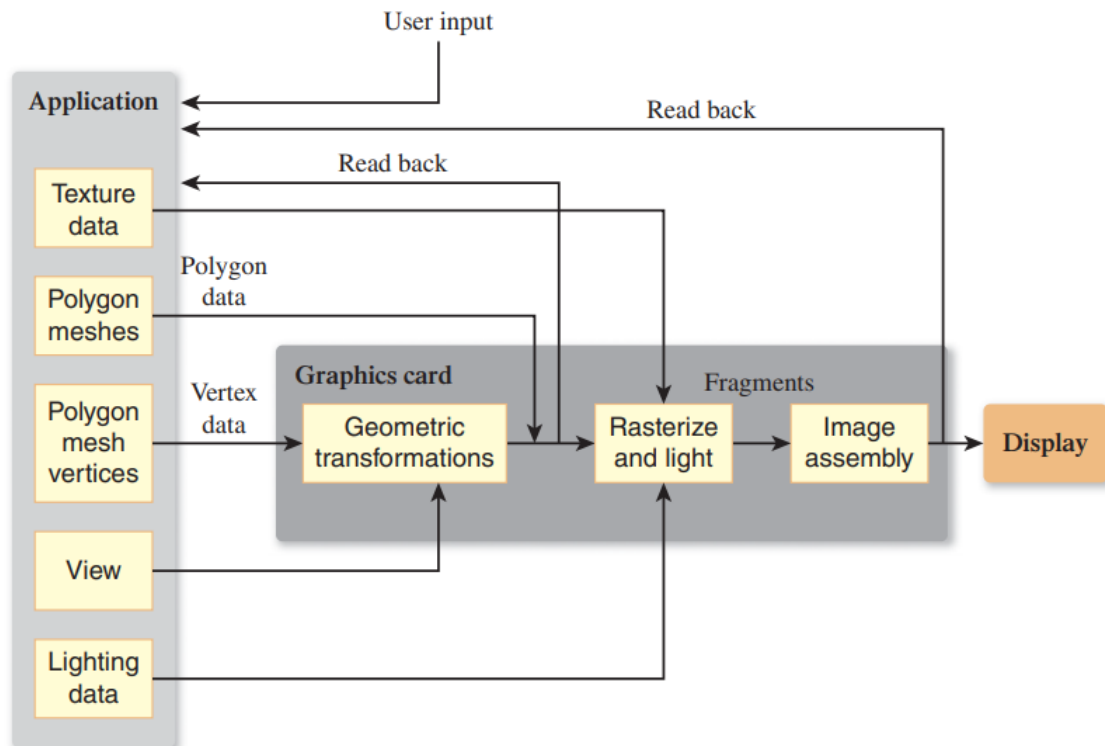
Rendering triangles in correct back-to-front sequence was a significant research challenge in computer graphics (Ragan-Kelley et al., 2011). However, it is now virtually universally addressed utilizing z-buffer, which employs a memory address buffer to resolve the issue forcefully.

It appears that geometric modification in the graphics pipeline may be performed almost exclusively in 4D coordinates, including three traditional geometric and a 4<sup>th</sup> uniform coordinate that facilitates perspective visualization. These four-dimensional coordinates are handled using a four-by-four matrix and four vectors. Hence, the graphics pipeline includes extensive equipment for effectively computing and synthesizing matrices and vectors. This 4D reference frame is among the most intricate and elegant computer science constructions and it is, without a doubt, the greatest cognitive obstacle to overcome while studying computer graphics (Chen et al., 2021). Most of the first chapter of any graphics book is devoted to such coordinates.

The amount of triangles drawn significantly impacts the rate at which visuals can be created (Blinn, 1991). Because interaction is more essential than the aesthetic value in many scenarios, decreasing the number of triangles used to depict a model is advantageous. In particular, when seen from a longer range, the model requires fewer triangles than observed from a closer range. This indicates that depicting a model with varied degrees of detail (LOD) is advantageous.



*Figure 1.6. Flowchart of the graphics pipeline (Source: Andries Van Dam, Creative Commons License)*



*Figure 1.7. Flowchart of the more detailed graphics pipeline (Source: Morgan McGuire, Creative Commons License)*

## 1.6. Numerical Issues

Several visual applications are essentially 3D numerical codes. Frequently, quantitative concerns are critical in such applications. In the "good old times," it was very hard to tackle such difficulties in a durable and adaptable fashion because computers had multiple intrinsic models for integers and, to make matters harder, managed errors in diverse and inconsistent manners. Thankfully, nearly all modern computers adhere to the IEEE floating-point standards, which IEEE Standards Association evokes (Sanyal et al., 2010). This allows the developer to make several handy assertions regarding how particular numerical circumstances will be addressed.

Despite IEEE floating-point offers many characteristics useful for writing numerical algorithms, only a handful are essential for most graphic applications. First and foremost, it is essential to comprehend that IEEE floating-point has three "special" meanings for all real numbers:

- i. **Infinity ( $\infty$ ):** It is a fair number that exceeds all other acceptable numbers.
- ii. **Minus infinity ( $-\infty$ ):** It is a fair number smaller than all other acceptable numbers.
- iii. **Not a number (NaN):** This is an incorrect integer that results from an invalid action, such as null divided by null (Stam, 2009).

The developers of IEEE floating-point chose a variety of exceedingly handy programming choices. In addressing instances like division by zeroes, several of these pertain to the three exceptional values listed previously (Gross, 1998). In these situations, an error is recorded, but the developer can typically disregard it. Particularly, the following principles regarding division by indefinite quantities hold for any positive integer:

$$+a/(+\infty) = +0,$$

$$-a/(+\infty) = -0,$$

$$+a/(-\infty) = -0,$$

$$-a/(-\infty) = +0.$$

Other operations that involve infinite values behave as expected.

Once more, for a positive "a," the following pattern is observed (Lang & Antelo, 2005):

$$\infty + \infty = +\infty,$$

$$\infty - \infty = \text{NaN},$$

$$\infty \times \infty = \infty,$$

$$\infty/\infty = \text{NaN},$$

$$\infty/a = \infty,$$

$$\infty/0 = \infty, 0/0 = \text{NaN}.$$

For a Boolean expression having infinite values, the anticipated principles apply:

- i. All finite and valid numbers are to be smaller than  $+\infty$ .
- ii. All finite and valid numbers to be larger than  $-\infty$ .
- iii.  $-\infty$  is smaller than  $+\infty$  (Marschner & Shirley, 2018).

The rules for expressions with NaN values are straightforward:

- i. Every mathematical equation containing NaN yields NaN.

Any Boolean expression that contains NaN is untrue (Christie et al., 2008).

Arguably the most advantageous feature of IEEE floating-point involves how divide-by-null is treated; for every positive real integer a, the following principles about divide by value 0 hold true (Machover, 1994):

$$+a/+0 = +\infty,$$

$$-a/+0 = -\infty.$$

Numerous numerical computations become considerably easier if the developer fully benefits from IEEE principles. Take, for instance, the expression (Mitchell & Netravali, 1988):

$$a = \frac{1}{\frac{1}{b} + \frac{1}{c}}.$$

These expressions apply to resistive elements and optics. If divide-by-zero caused a program failure, as in several systems prior to IEEE floating-point, then two if clauses would be needed to verify for low or zero value of b or c (Maciejewski, 1990). However, if b or c is 0, the IEEE floating-point will return a value of 0 for a, as requested. Taking leverage of the Boolean features of NaN is yet another typical strategy for evading special tests. Think about the following segment of code:

```
a = f(x)
if (a > 0), then
do something
```

Here, the given function “f” might answer “ugly” values as  $\infty$  or even NaN, but the “if” condition is yet defined: it is false for a = NaN, or even a =  $-\infty$  and true for a =  $+\infty$ . With care in determining which values are provided, the if statement can frequently make the correct selection without additional checks. This renders programs increasingly compact, powerful, and effective.

**Remember:**

The human eye can detect over 10 million different colors, but most computer screens can only display a fraction of that number. This is due to limitations in the color gamut of the screen and the number of bits used to represent each color.

### 1.7. Efficiency

There exist no magical methods for increasing the efficiency of code. With rigorous compromises, performance is accomplished, and these choices vary between designs (Bialac, 1985). According to a useful guideline, developers should focus more on shared memory trends than operations counts for the future. This is the antithesis of the most effective strategy from twenty years ago. This change occurred because memory speed has still not kept up with processor efficiency. Since this tendency is

anticipated to persist, the significance of constrained and consistent memory access for optimizing should keep increasing.

A sensible method for creating code quickly is to follow this sequence, performing only the necessary steps:

- i. Compose the code using the simplest syntax feasible. Calculate intermediate data on-demand as opposed to keeping them.
- ii. Compile in optimal settings.
- iii. Utilize whatever monitoring tools are available to identify major problems.
- iv. Analyze data structures to discover methods for enhancing localization. Whenever appropriate, the data unit size should correspond to the cache/page size of the desired system (Hoines, 1987).

If profiling indicates inefficiencies in numerical computation, analyze the compiler-generated assembly language for overlooked efficiencies. Modify the original code to fix any identified issues.

The basic stage is the most vital of all these. Many "optimizations" render the code more difficult to understand without improving performance. In addition, spending time optimizing code upfront is often better invested in fixing bugs or adding functionality. Additionally, be wary of advice from outdated sources; some conventional methods, like utilizing integers rather than real numbers, may no longer improve performance, as newer Processors can typically perform floating-point calculations as quickly as integer computations. Profiling is always required to determine the value of optimization for a given machine and compiler (Stork, 2015).