KAPITEL 4 / *CHAPTER 4* [4]
ANALYSIS OF GRAPHICS PIPELINES

DOI: 10.30890/2709-2313.2024-30-00-019

Introduction

Rendering of three-dimensional images [1, 2] in modern three-dimensional computer graphics systems is carried out according to a certain sequence of stages that form a graphics pipeline. Graphics pipelines are widely used to determine the hardware implementation of the stages of constructing three-dimensional images. The development of hardware and software, parallelization algorithms, artificial intelligence leads to the need of new graphics pipeline structures development.

4.1. Analysis of graphics pipelines stages

The main subsystems of the graphics 3D pipeline [3, 4, 5] (Fig. 1) are the scene description subsystem, the geometric subsystem, the rendering subsystem, and the visualization subsystem.

At the stage of describing the scene, its logical structure is formed. Scene's objects, their relative location and states, camera characteristics and light sources are defined.

At the stage of geometrical transformations, polygonal models of the of scene objects' surfaces are formed. Usually, the surface of the object is represented by triangles, which are the simplest two-dimensional shapes and provide a guaranteed decomposition of the surface. Displacement, rotation, and scaling transformations are performed on the objects for their correct placement in the space of the scene. Two-dimensional coordinates of points on the surface of the object are determined with the help of projection transformations.

[4] ***Authors:*** *Romanyuk Oleksandr Nykyforovych, Bobko Oleksii Leonidovych, Zavalniuk Yevhen Kostyantynovych, Titova Nataliia Volodymyrivna, Romanyuk Serhii Oleksandrovych, Stakhov Oleksii Yaroslavovych*

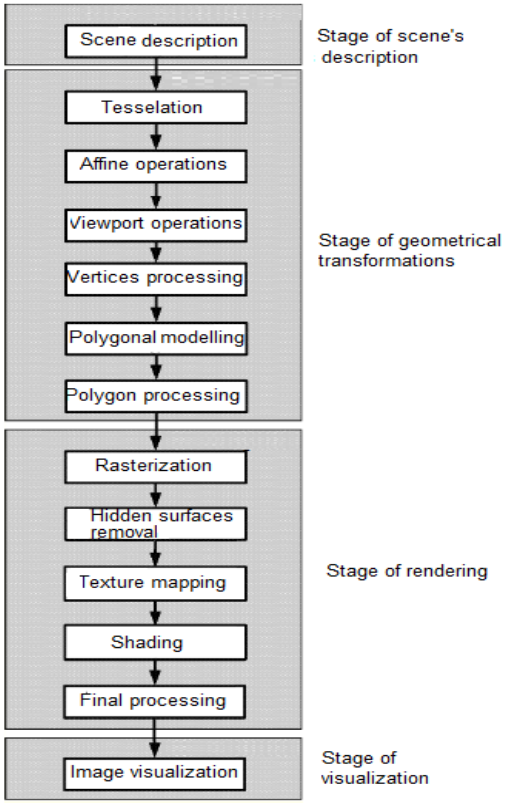

Figure 1 – General sequence of graphics pipeline stages

Source: [3]

At the rendering stage, up to 80% [2] of the calculations for the formation of three-dimensional scenes are performed. Rasterization of three-dimensional primitives and their subsequent shading are carried out.

During rasterization, the 3D polygons of an object's surface are converted into a set of 2D fragments for displaying on the screen. The rasterization process includes setting primitives, "moving along the edge", generating fragments. Primitives setup involves processing geometric primitives defined by vertices that have been transformed and projected into two-dimensional screen space coordinates. During "edge walking", the rasterizer determines which pixels on the screen intersect with geometric elements. This involves analyzing the edges of the primitives to find the pixels inside the boundaries of each triangle. After that, the rasterizer generates a fragment for each covered pixel.

Color intensity is determined for each pixel of the polygon. For this, the reflective properties of the material, the curvature of the surface of the object, the position of the

light source, and the position of the observer are taken into account. Usually, the value of the color intensity or the normal vectors between the vertices of the polygon is interpolated. Additionally, a texture mapping can be applied to the surface of the object. With the help of anti-aliasing methods, jagged contours are eliminated.

The final stage of the graphics pipeline lies in displaying the formed image on the screen. Here, the processed image fragments are combined and displayed as a complete scene. This involves copying the pixel data to the video memory, from where the image is output to the display.

To implement the stages of the graphics pipeline on the GPU, shader programs are used [6]. The main application programming interfaces for the implementation of shaders are DirectX and OpenGL. DirectX provides multimedia tools for developing graphics applications for Microsoft Windows, Xbox. OpenGL is used to develop cross-platform (Windows, Linux, Apple) graphics applications.

The described sequence of three-dimensional image formation stages is only general. Depending on the shader development application programming interface chosen by the developer and the specifics of the rendering task, the sequence of stages of the graphics pipeline may vary.

We compare the main stages of implementation of the graphics pipeline for OpenGL [7, 8] and DirectX [9] (Fig. 2).

The stages of the implementation of the graphics pipeline for OpenGL (see Fig. 2) [7, 8] include the specification of vertices, the usage of a vertex shader, a tessellation shader, a geometry shader, post-processing of vertices, primitive assembly, rasterization, the usage of a fragment shader, and the execution of per-sample operations.

At the stage of vertex specification [7, 8], an ordered list of primitive vertices of the object's surface is formed. Vertices can be characterized by attributes such as spatial coordinates, texture coordinates, and normal coordinates. Vertex buffer objects (VBO) are used to store information about vertices, and vertex array objects (VAO) are used to determine the necessary characteristics of vertices.

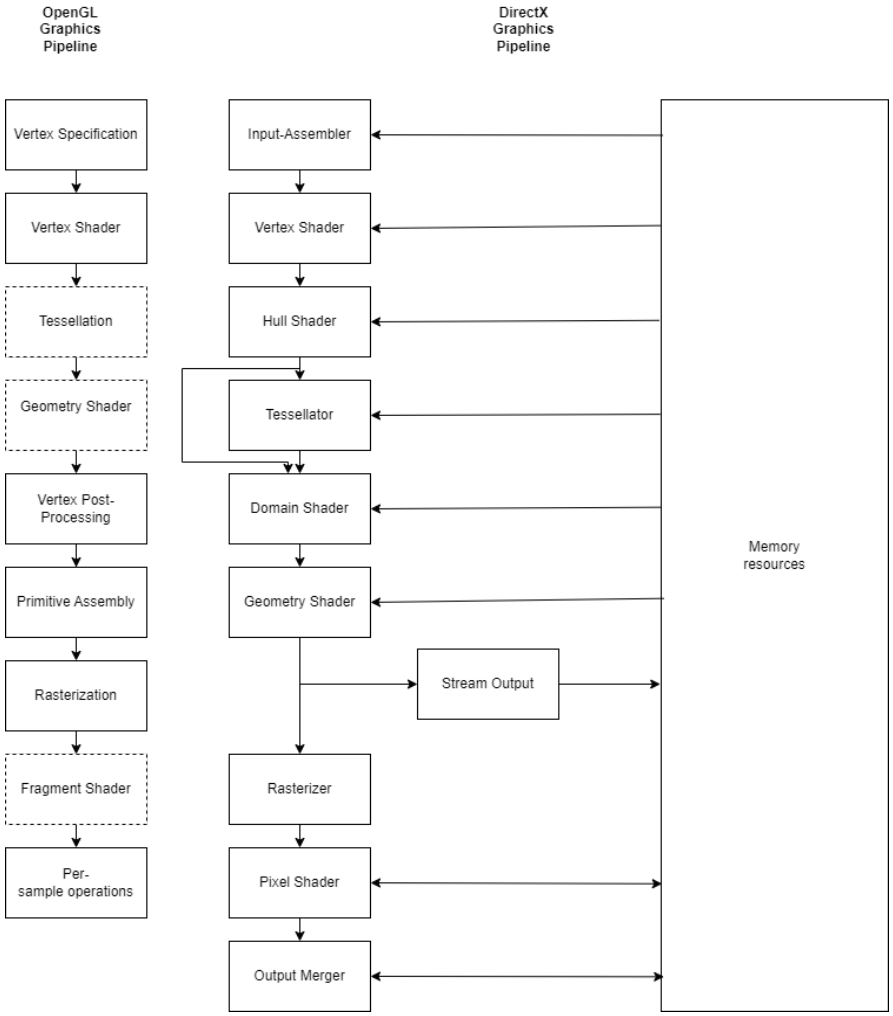

Figure 2 –Sequences of OpenGL and DirectX graphics pipelines stages

Source: [7, 8, 9]

The generated list of polygon vertices is submitted to the graphics pipeline for further processing.

Vertex shaders [7, 8] provide separate processing of each vertex and are used to determine the final coordinates of the vertices, calculate the lighting vectors in the vertices, and adjust further rendering. Optimization of the stage is possible by saving the results of vertex processing in the cache memory for their repeated use.

Usage of the tessellation shader [7, 8] is optional. Tessellation shaders perform further decomposition of polygons of the surface of the object. Two substeps are used: the tessellation control shader (determines the required level of tessellation of the given primitive) and the tessellation evaluation shader (determines the characteristics of the generated vertices).

The use of a geometry shader [7, 8] is also optional. Based on the input primitives, zero or more output primitives are formed. Changing type, removing, vertex modification and tessellation of primitive are all possible, the latter is the purpose of a tessellation shader and is not recommended. It is especially useful to use geometric shaders for multi-layer rendering, where a primitive is rendered for many images.

At the stage of post-processing of vertices [7, 8], sections of primitives lying outside the scope of viewing volume are cut off (clipping). In addition, based on the analysis of the location of polygons in the window space, primitives oriented against the observer are rejected (face culling).

At the stage of assembly of primitives, ordered sequences of polygons are formed on the basis of processed vertices. The simplified form of the stage is also applied before using the tessellation and geometric shaders.

At the rasterization stage [7, 8], three-dimensional polygons are transformed into a set of two-dimensional fragments.

The use of a fragment shader [7, 8] provides determination of the intensity and depth of color at each point of the polygon. To calculate color intensities, interpolated polygon vertices data (vectors of normals, incidence and reflection of light vectors, color intensity) are used. The stage may not be used in some cases, for example, when simple formation of primitive depth buffers is required.

To speed up the shading of fragments [10], it is recommended to avoid excessive normalization of vectors (in the case of small surface curvature or preservation of the unit length of the vector during transformation), store computationally complex information in the corresponding textures, perform preliminary visualization of the depth of fragments, move surface calculations to the vertex shader, use the smallest allowable accuracy of calculations and the most simple type of shader, reduce the size of textures, take into account the level of detail when shading fragments.

The stage of per-sample operations [7, 8] includes the implementation of a number of tests on the processed fragment. The z-depth test allows you to determine whether a fragment is covered by a fragment closer to the observer. If not, the corresponding pixel depth value is updated in the z-buffer. The pixel ownership test shows whether a

pixel's window is overlapped by another window. The scissor test allows you to discard fragments outside a defined rectangular part of the screen.

The resulting pixel color intensity values are recorded in the frame buffer.

The sequence of stages of the DirectX graphics pipeline [9] (see Fig. 2) includes the stages of input assembler, use of vertex shader, hull shader, use of tesselator, domain shader, geometry shader, stream output, rasterization, pixel shader, output merger.

The stage of input assembler [9] lies in the formation of primitives for processing in the pipeline based on the given vertices. Formed primitives receive identification numbers to avoid their repeated processing.

The stage of using the vertex shader [9] similarly provides positional transformations and calculation of lighting vectors for polygon vertices.

The stage of applying the hull shader [9] is the first tessellation stage of dividing the solid surface of the object into triangles. Based on the low-order surface model's control point sets, the tessellation factors for isolated surface patches are determined. The tessellation factors determine the level of subdivision of the surface patch.

The tessellator step uses the tessellation factors generated in the previous step to divide the surface into smaller triangles.

The domain shader stage lies in calculating the attributes of new vertices based on the generated polygons and predicted tessellation factors.

The stage of using the geometry shader [9] analogously lies in the processing of whole primitives and ensures the formation of output strips of triangles, strips of lines or lists of vertices to simplify or complicate the object model.

The stage of streaming output [9] lies in the permanent recording of the calculated data in the memory buffer for the possibility of re-submitting them to the input of the pipeline.

The rasterization stage is analogously the preparatory stage for the pixel shader stage. The three-dimensional geometric description of the scene is transformed into a two-dimensional set of points for shading.

At the stage of using the pixel shader [9], the color intensity of each point of the

primitive is analogously determined, taking into account the parameters of its vertices.

The step of output merger [9] lies in forming the final array of pixel color intensities for recording in the frame buffer based on combining the results of the pixel shaders.

In general, the graphics pipelines of OpenGL and DirectX correspond closely to each other. However, there is a difference in the terminology of the graphic pipeline stages and the peculiarities of approaches to individual stages of image formation. In particular, the OpenGL graphics pipeline contains only one stage of tessellation, which includes substages for determining the required level of surface decomposition (Tessellation Control Shader) and calculating the vertices of the formed primitives (Tessellation Evaluation Shader). At the same time, the DirectX graphics pipeline accommodates three separate stages for tessellation. At the first stage (Hull Shader) the decomposition level of the surface area is determined, at the second stage (Tessellator) the surface is divided into smaller primitives, at the third (Domain Shader) new vertices are calculated.

4.2. Analysis of graphics pipelines development

Let's consider the dependence of the features of the implementation of the stages of the graphics pipeline on the development of video cards. There are five generations of video card graphics pipeline models [11] (Fig. 3).

Graphic pipelines of the first generation video cards [11] (Voodoo Graphics, 1996) are characterized by the implementation of rasterization and raster operations using a graphics processor. Other graphics pipeline operations are implemented on the central processing unit (CPU).

A peculiarity of the stages of the graphics pipeline of second-generation video cards [11] (GeForce/Radeon 7500, 1998) is the implementation of vertex transformation and lighting adjustment (T&L) operations on graphics processors. That is, for the first time, all the main operations of the graphics pipeline were intended to

be implemented on the GPU.

Graphics pipelines of third-generation video cards [11] (GeForce3/Radeon 8500, 2001) are characterized by the introduction of software (shader) implementation for T&L calculations.

A peculiarity of the graphic pipelines of the fourth generation video cards [11] (Radeon 9700/GeForce FX, 2002) is the implementation of all the main stages of three-dimensional images formation using shader programs.

Stages of the graphics pipeline of the fifth generation video cards [11] (GeForce 8, Radeon R600, Intel GMA X3000) implemented with the help of shaders are characterized by the possibility of execution on any GPU block. Graphics pipelines of this type are called "unified". For example, all GPU units can be used to process the complex geometry of a 3D object.

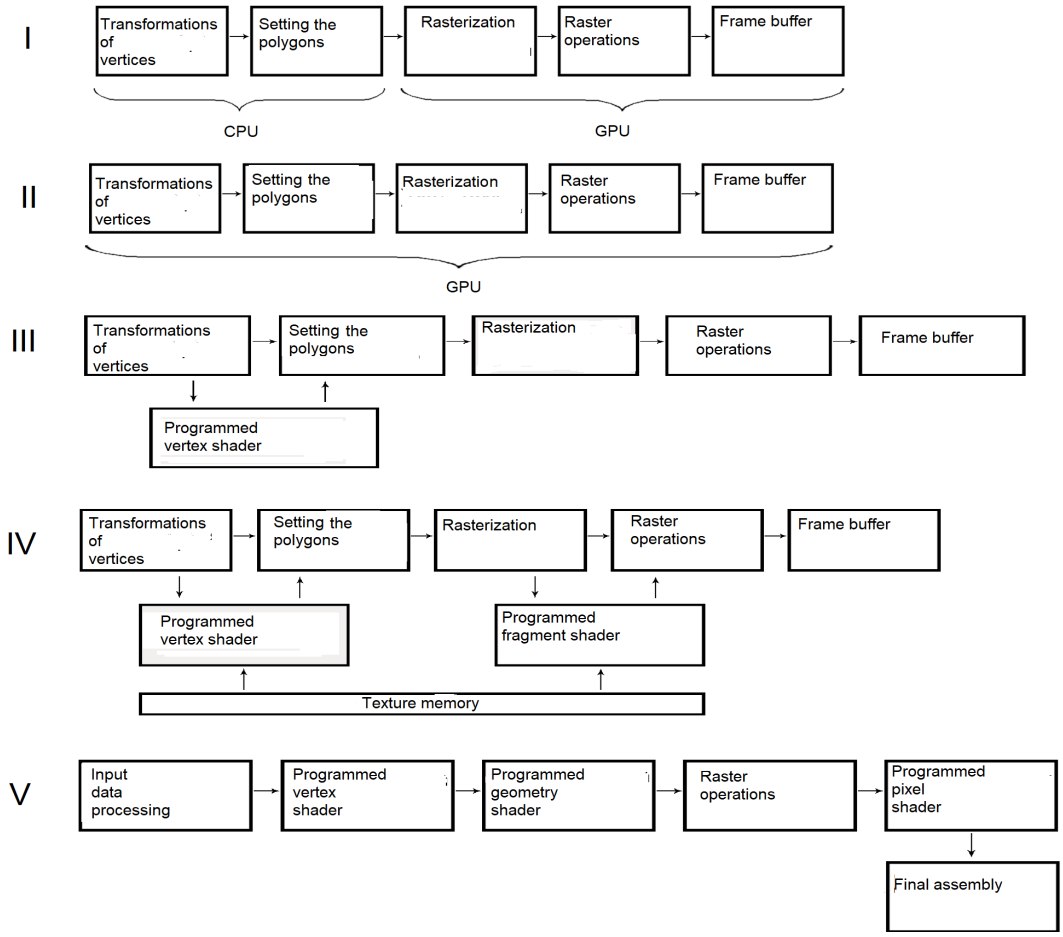

Figure 3 –Graphics pipelines stages of the first–fifth generation video cards

Source: [11]

A significant disadvantage of standard graphics pipeline models is their linear organization. To change partially processed data, you must wait for all stages of the pipeline to complete. Therefore, an augmented pipeline [12] (Fig. 4) was introduced, where it is possible to send data to the video memory from the intermediate stage of image formation.

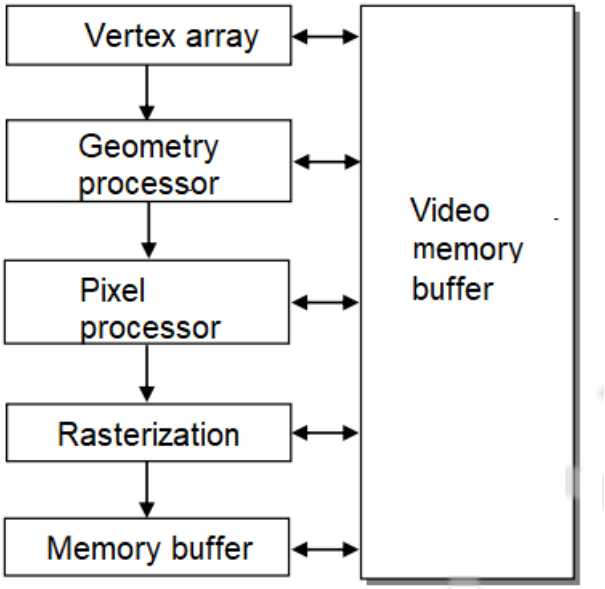

Figure 4 –Augmented graphics pipeline

Source: [12]

To ensure high-performance image formation, the graphics pipeline calculations are parallelized.

The most popular directions of parallelization of graphics pipeline tasks are parallelization of input primitives and parallelization of the pixel characteristics calculation. It is possible to sort primitives [13, 14] before the stage of geometric transformations ("sort first"), sort primitives after geometric transformations before rasterization ("sort middle"), sort fragments between rasterization and shading ("sort last"), sort pixels between shading and visualization ("image sort last"), sorting objects at each stage of image formation between different instances of graphic pipelines ("sort everywhere").

For example, the model of the graphic pipeline from M. Kenzel et al. [15] includes the use of the "sort middle" principle to sort primitives in global memory between

blocks of geometric transformations and rasterization. Inside the block of geometric transformations, data is sorted in local memory between components of processing vertices and primitives according to the "sort everywhere" principle. Inside the rasterization block, the data is sorted in local memory between the boxes rasterizer, the tile rasterizer, and the fragment processing component according to the same principle. The approach ensures highly efficient use of available GPU resources and efficient use of memory bandwidth.

The development of artificial intelligence algorithms provides opportunities for optimizing the stages [16] of existing graphic pipelines. In particular, M. V. Harris and S. K. Semval [17] proposed the alternate use of a traditional graphic pipeline and a graphic pipeline based on deep learning in dynamic systems of frame formation. This approach provides a reduction in the time of visualization of a sequence of three-dimensional scenes. The advanced graphics pipeline based on deep learning (ADLGP) includes the division of the scene into small frame blocks of 4*4 px size, the generation of semantic text descriptions corresponding to the frame blocks, the generation of frame blocks from the text descriptions using the AttnGAN model and their assembly in final image, image quality enhancement using SRGAN.

Summary

The sequence of stages of the graphics pipeline depends on the specifics of the rendering task, the development of video cards, shaders and application programming interfaces. Promising directions for improving graphic pipelines are parallelization and provision of cyclic data processing, application of artificial intelligence algorithms.