

Graphics Assignments

by Tynan Kahar

Submission date: 30-Nov-2024 01:58PM (UTC+0000)

Submission ID: 244193726

File name: 41519_Tynan_Kahar_Graphics_Assignments_628112_1620705541.docx (176.88K)

Word count: 2093

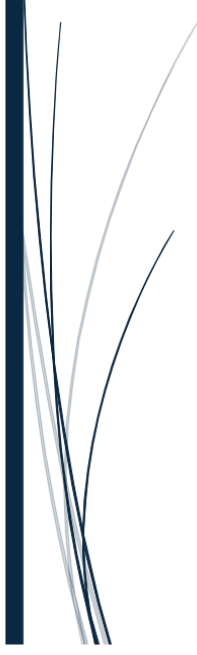
Character count: 10343

A thick dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

11/28/2024

Graphics and Games Programming

BSc Computer Games Development

Several thin, curved lines in shades of blue and grey originate from the bottom of the vertical bar and sweep upwards and to the right.

Tynan Kahar (2204886)
UWTSD

Contents

Task 1	1
a)	1
Vertex Specification.....	1
Vertex rendering.....	2
Vertex processing.....	2
Vertex shader	2
Tessellation.....	2
Geometry shader	2
Vertex post processing.....	3
Transform Feedback.....	3
Primitive assembly	3
Clipping	4
Face culling.....	4
Rasterization	4
Fragment processing.....	4
Per-sample Operations	5
b)	5
Task 2	8
References.....	9

Task 1

a)

Vertex Specification

The first stage of the graphics pipeline is the vertex specification. In this stage, all of the data is given to the CPU, more specifically all the vertices, which can be more simple geometry data (for example, 1.0, 0.0, 4.0). This includes [2] vertex array and vertex buffer objects. The vertex array objects specify the data all of the vertex's contain, and the vertex buffer stores it.

The data stored in vertex's is a series of [3] user input values. Each set of user inputs specifies a vertex's, but the inputs do not need to be normalized and be completely random.

Vertex rendering

After the data is specified in the previous stage the rendered using a draw command it will be presented as a base primitive.

Vertex processing

In this stage the vertices for the previous stage are processed. Most of the sub stages under this are optional and they are programable meaning that a program can be write on the GPU that influences the graphics pipeline

Vertex shader

The vertex shader will execute the [2] user-defined program on each of the vertex's and then position each vertex's.



The vertex shader can also fulfil another vertex processing position if there are no other vertex processing stages; it will take the job of [2]clip space positioning of the vertex's.

During this stage, one of the limitations is that each of your inputs can only create an equal amount of output; they can't be split or combined.

Tessellation

The tessellation stage is where detail is added to the graphics; it does this, but dividing the polygons into smaller polygons, for example, a ball with 100 polygons would be a lot less smooth than one that has 1000 in the same area.



It is also used to express depth; it does this by increasing the amount of polygons the closer the object is to the screen and decreasing them as they get further away; this is known as dynamic level of detail; it is done to save computation time.



A common example of this is with terrains, as they will be far blockier at farther distances, and without tessellation to increase the amount of polys, it would remain blocky.

Geometry shader

The geometry shader works with whole polygons rather than just individual vertices. There are three main things that it is used for, the first of which is that it is used to create new polygons, lines, and points. It can use tessellation to divide one polygon into multiples to increase the detail of specific objects.

The second use of the geometry shader is that it can modify the polygons, for example, by stretching them or reducing their size.

Finally, it can be used to completely remove polygons that are no longer needed, for example, the polygons of a body that is covered by clothing.

The geometry shader works as an editor in the pipeline and is mainly used for special effects.

Vertex post possessing

The vertex post-possession stage is where an array of different fixed function operations are used, meaning they are all hardcoded and are done automatically by the GPU without any unique programming needed.

The importance of the stages within the vertex post is that it reduces the complexity of the following stages; it does this by making sure that the geometry is all in the correct format. It makes rendering more efficient by removing unused geometry. As well as working as the connector between the aforementioned shaders and the final rasterization. To simplify it is the preparation and cleanup stage.

Transform Feedback

The purpose of this stage is to hold onto output data from the shaders; it holds this data in buffer objects. The reason you would want to store your data like this is because it allows you to reuse the data the shader has already transformed, meaning if the shader had already calculated a complex vertex it can be used again without having to recalculate an example of this could be a particle system as it positions stored in the buffer as input for the next frame saving time from doing the whole process from scratch. It is also helpful when debugging.

Primitive assembly

During this stage the GPU collects the output vertex from the previous stages and turns them into shapes like triangles, lines, and points dependent on the amount of vertices; for example, 3 making a triangle, 2 making a line, and one making a point. If a triangle stripe is used, which is a more efficient way of creating triangles, by overlapping vertices to create more triangles with fewer vertices, you could make 10 triangles with this method using only 12 vertices; using the other method, it would take 30 vertices to achieve the same result.

This is also a point at which you could stop using the pipeline if you don't plan on rendering an image to the screen and intend on saving the data for later using the transformation feedback stage.

Clipping

The clipping part of the graphics pipeline is where polygons outside of the camera's view are removed. This is done to ensure only the polygons that are visible are processed further and used for rendering, as it is inefficient to render objects that cannot be seen.

It finds what is not within its view using the frustum; anything outside of its defined ratios is removed. If something is half in the frustum, and half out it clipped, creating a new smaller Polygone.

Face culling

The face culling stage is similar to the clipping stage as it removes unnecessary polygons but as the once outside of the frustum are already removed in the clipping stage the face culling stage removes polygons that wouldn't be visible even though they are with the frustum, for example polygons that are facing away from the camera .

Rasterization

The rasterization stage is the stage where the 3D shapes created are converted into 2D pixels that will be shown on the screen. This is done by the GPU, which finds which pixels on the screen are in the place of which polygons, and then each pixel is a give and fragment that represents its data, like colour and depth. Finally, it decides what polygons are associated with each of the pixels on the 2D screen.

This stage is crucial as it is the stage that brings a 3D image into 2D, and modern GPUs can perform this extremely fast, allowing for real-time rendering.

Fragment processing

During this stage, fragments that were made in the previous stage are developed further to design their final colour, depth, and visibility on the screen. Each of the fragments represents a piece, but further calculations are required to determine if they will be displayed.

The way this works is by using a fragment shader on each of the fragments; this fragment shader is programable like the other shaders, and it applies texture and effects like shadows and lighting as well as reflections.

It then runs depth testing to determine whether a fragment is hidden behind another fragment with the same position on the screen. This is done using a depth buffer; anything that is found to be hidden is removed

Per-sample Operations

This is the final stage of the graphics pipeline and is where final decisions are made on what needs to be removed using stuff like the stencil test to make sure that only the segments that match up with the shape of the window are processed, as well as depth testing to ensure objects behind the window are not displayed over it. After that is all complete, the pixels are sent to the framebuffer and are ready to be displayed.

b)

Initial Vertex: (3, 1, -3)

Translation: (-1.0, 2, -1)

Rotation Angle: 80.0o

Viewpoint: (0, 4.5, -1)

Up Vector :(0, 1, 0)

Look At Point :(2, 0, 0)

Field of View: 37.0°

Near Plane: 1.0

Far Plane: 90.0

To find if the transformed point is in the view volume first this that I would need to do is to, translate the point along the X axis and to do that the translation of the x-axis would need to be added to the vertex the formular would look like this:

New Vertex = (x-1,y,z)

NV = (3-1,1,3)

$$NV = (2, 1, -3)$$

Following this I will rotate around the x-axis to do this you would need to use the formula:

$$\begin{array}{lcl} I1 & 0 & 0 \\ I0 & \cos \theta & -\sin \theta \\ I0 & \sin \theta & \cos \theta \end{array} \begin{array}{l} |x| \\ |y| \\ |z| \end{array} = \begin{array}{l} |x| \\ |y| \\ |z| \end{array} \begin{array}{l} x \\ y \cos \theta - z \sin \theta \\ y \sin \theta + z \cos \theta \end{array}$$

$$\begin{array}{lcl} I1 & 0 & 0 \\ I0 & \cos(80^\circ) & -\sin(80^\circ) \\ I0 & \sin(80^\circ) & \cos(80^\circ) \end{array} \begin{array}{l} |x| \\ |y| \\ |z| \end{array} = \begin{array}{l} |x| \\ |y| \\ |z| \end{array} \begin{array}{l} x \\ (1) \cos(80^\circ) + (-3) \sin(80^\circ) \\ (1) \sin(80^\circ) + (-3) \cos(80^\circ) \end{array}$$

$$X' = 2$$

$$Y' = 3.128$$

$$Z' = 0.464$$

The next thing to do would be to convert the 3d show from the view point into a view space meaning turning 3d into 2d

First the viewing direction or forward vector would need to be found this is done using the camera position and the look at direction.

FV = Normalize(Look At Point-Viewpoint)

$$\text{Normalize}([2, 0, 0] - [0, 4.5, -1]) = \text{Normalize}([2, -4.5, 1])$$

$$\|FV\| = \sqrt{2^2 + (-4.5)^2 + 1^2} = 5.025$$

(/2 square root)

$$FV = [0.398, -0.895, 0.199]$$

Next is the right vector which at a right angle to the up and forward vector

$$RV = \text{Normalize}(Up * FV)$$

Cross product;

$$RV = \text{Normalize}([1 \cdot 0.199 - 0 \cdot -0.895, 0 \cdot 0.398 - 0 \cdot 0.199, 0 \cdot -0.895 - 1 \cdot 0.398])$$

$$RV = \text{Normalize}([0.199, 0, -0.398])$$

Magnitude;

$$\|RV\| = (0.199^2 + 0^2 + (-0.398)^2)^{1/2} \approx 0.445$$

(/2 square root)

$$RV = [0.447, 0, -0.894]$$

Finally the True up vector would be needed which is found using the formular $TUP = RV * FV$

$$TUP = [0.398, -0.895, 0.199] \times [0.447, 0, -0.894]$$

$$TUP = \text{Normalize}([0.801, 0.596, 0.4])$$

The viewing matrix is made with the formuler

$$VM = \begin{bmatrix} RV_x & RV_y & RV_z & -RV_x \cdot \text{Viewpoint} & -RV_y \cdot \text{Viewpoint} & -RV_z \cdot \text{Viewpoint} \\ FV_x & FV_y & FV_z & -FV_x \cdot \text{Viewpoint} & -FV_y \cdot \text{Viewpoint} & -FV_z \cdot \text{Viewpoint} \\ TUP_x & TUP_y & TUP_z & -TUP_x \cdot \text{Viewpoint} & -TUP_y \cdot \text{Viewpoint} & -TUP_z \cdot \text{Viewpoint} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$VM = \begin{bmatrix} 0.447 & 0 & -0.894 & -0.894 & -1.602 & 0 \\ 0.801 & 0.445 & 0.400 & 0.400 & -1.602 & 0 \\ -0.398 & 0.896 & -0.199 & -4.229 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Finally transforming the leading into the view space

The formular

$$VS = VM * VP$$

$$\text{View-Space Vertex} = [-0.415, 1.578, -2.316, 1.000]$$

Now it is converted to view space. the perspective needs be added

The perspective matrix using field of view, aspect ration and near and fair planes

$$VM = \begin{bmatrix} 1/\tan(FoV/2) & 0 & 0 & 0 & 1 \\ 0 & 1/\tan(FoV/2) & 0 & 0 & 1 \\ 0 & 0 & -(f+n)/(f-n) & -2fn/(f-n) & 1 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$



Translation Along the X-axis: (2.0,1.0,-3.0)

Rotation Around the X-axis: (2.0,3.128,0.464)

Conversion to View Space: (-0.415,1.578,-2.316)

Perspective Projection: (-0.535,2.036,0.149)

The transformed point would not be within the view volume because of its y coordinate that is outside of the allocated range.

Task 2

To determine if the vertex (3,1,-3) is going to be culled by the plane where $p_0 = (2, 2.5, 3)$ and $n = (1, 2, 0)$. And plane equation is $x*n + p_0*n = 0$

The first step is to expand the plane equation

$$p_0 \cdot n = (2)(1) + (2.5)(2) + (3)(0) = 2 + 5 + 0 = 7$$

$$X \cdot n + 7 = 0$$

$$x + 2y + 0z + 7 = 0$$

Next the signed distance will need to be calculated

$$d = \frac{x*n + p_0 \cdot n}{\|n\|}$$

after plugging in the values $D=5.37$ this means that the vertex (3,1,-3) is not culled as D would have to be less than 0 if it was outside of the plane and would need to be culled, as D is greater than 0 it is not culled.



Task 3

```
namespace Test
{
    TestCamera::TestCamera(GLFWwindow* window)
    {
        m_window(window),
        cameraPosition(0.0f, 4.5f, -1.0f),
        cameraFront(2.0f, 0.0f, 0.0f),
        cameraUp(0.0f, 1.0f, 0.0f),
        cameraSpeed(1.0f)
    }
}
```



```
// Set the projection matrix
glm::mat4 projection = glm::perspective(glm::radians(37.0f), 1600.0f / 900.0f, 1.0f, 90.0f);
m_Shader->setUniformMat4f("projection", projection);
```

```
glm::mat4 Tr1 = glm::translate(glm::mat4(1.0f), glm::vec3(3.0f, 1.0f, -3.0f));
glm::mat4 Tr2 = glm::translate(glm::mat4(1.0f), glm::vec3(-1.0f, 2.0f, -1.0f));
glm::mat4 Rotation = glm::rotate(glm::mat4(1.0f), glm::radians(80.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glm::mat4 model;

model = Tr1;
model = Rotation * Tr2 * model;
```

```
// Create a windowed mode window and OpenGL context
GLFWwindow* window = glfwCreateWindow(1600, 900, "GLFW Window", NULL, NULL);
if (!window) {
    std::cerr << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
```

References

- [1]
University of Pretoria, "Appendix A Fundamentals of the Graphics Pipeline Architecture."
Available:
<https://repository.up.ac.za/bitstream/handle/2263/28196/04appendices.pdf?sequence=5>
- [2]
"Rendering Pipeline Overview - OpenGL Wiki," *Khronos.org*, 2019. Available:
https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
- [3]
"Vertex Shader - OpenGL Wiki," *Khronos.org*, 2022. Available:
https://www.khronos.org/opengl/wiki/Vertex_Shader#Inputs. [Accessed: Nov. 30, 2024]

[4]

V Scott Gordon and J. L. Clevenger, *Computer Graphics Programming in OpenGL with C++*. Walter de Gruyter GmbH & Co KG, 2024.

[5]

Mike Shah, "[Episode 4] [Theory] The Programmable Graphics Pipeline (Interview Question) - Modern OpenGL," *YouTube*, Jun. 05, 2022. Available:

<https://www.youtube.com/watch?v=kpA5X6el6fM>. [Accessed: Oct. 06, 2024]

[6]

"Translating points (video) | Translations," *Khan Academy*. Available:

<https://www.khanacademy.org/math/geometry/hs-geo-transformations/hs-geo-translations/v/translating-points>

[7]

"cameras · PyTorch3D," *Pytorch3d.org*, 2024. Available: <https://pytorch3d.org/docs/cameras>.

[Accessed: Nov. 30, 2024]

[8]

"OpenGL Course - Create 3D and 2D Graphics With C++," www.youtube.com. Available:

<https://www.youtube.com/watch?v=45MlykWJ-C4>

[9]

"Coordinate Spaces | David Lively's Blog," *Davidlively.com*, 2024. Available:

<http://davidlively.com/programming/graphics/coordinate-spaces/>. [Accessed: Nov. 30, 2024]

Graphics Assignments

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

79/100

Engagement with Literature: 10/15

Knowledge and Understanding: 15/20

Cognitive and Intellectual Skills: 25/30

Practical Skills: 25/30

Transferable Skills: 4/5

Overall 79/100

Comment:

Avoid Youtube as a source of reference, there will always be a more credible source!

There were a number of places in the text that would have benefitted from a good reference and this would have ultimately netted you a slightly higher score!

A fantastic write up on the graphics pipeline but alas, your equation for task 1 was let down by a mistake early on that ultimately skewed all of your data, but by correcting it were spot on!

Great attempt overall.

When it comes to demonstrating equations I find for formatting sake its better to provide a screenshot as it will ensure consistency across devices. Also worth noting that Labelling an equation as a figure is typically done to maintain consistency with the formatting style used in the document. In some academic styles, such as in certain fields of science and engineering, equations are treated as visual elements or figures, especially when they are central to the content of the paper. By labelling an equation as a figure, it

allows for clear referencing throughout the text and ensures that it is easily distinguishable from the main narrative. This approach helps readers locate and focus on key mathematical expressions more effectively.

See inline comments also

PAGE 1

PAGE 2

PAGE 3



Comment 1

References should go at the end of the sentence, rather than in the middle.



Comment 2

An ideal point for reference



Comment 3

References here would have been great

PAGE 4

PAGE 5

PAGE 6



Comment 4

In academic writing, particularly in scientific and technical papers, the use of the first person (e.g., "I", "we") is generally discouraged as it can make the writing appear subjective and less formal. The focus of academic work should be on the research, analysis, or findings rather than the researcher. Using an objective tone ensures that the argument or evidence takes precedence over personal opinions or experiences, which enhances the perceived credibility and professionalism of the work. For example, instead of saying, "I conducted the experiment," you could write, "The experiment was

conducted," which maintains an impersonal and objective stance. To improve in this area, aim to use passive voice judiciously or rephrase sentences to emphasise the action rather than the actor. Additionally, consider adopting phrases like "the research indicates" or "it was observed" to convey findings without self-referencing. Reviewing published papers in your field can provide valuable examples of how to maintain an objective tone while ensuring clarity and precision.



Comment 5

You should have used a matrix for the translation too



Comment 6

I think this error here skewed your entire data!

PAGE 7

PAGE 8

PAGE 9



Comment 7

words formatting strikes again!

Typically what I do to avoid this is to screenshot them when they look as they should and paste that in place.

Labelling an equation as a figure is typically done to maintain consistency with the formatting style used in the document. In some academic styles, such as in certain fields of science and engineering, equations are treated as visual elements or figures, especially when they are central to the content of the paper. By labelling an equation as a figure, it allows for clear referencing throughout the text and ensures that it is easily distinguishable from the main narrative. This approach helps readers locate and focus on key mathematical expressions more effectively.

PAGE 10



Comment 8

Great stuff.



Comment 9

Labelling an equation as a figure is typically done to maintain consistency with the formatting style used in the document. In some academic styles, such as in certain fields of science and engineering,

equations are treated as visual elements or figures, especially when they are central to the content of the paper. By labelling an equation as a figure, it allows for clear referencing throughout the text and ensures that it is easily distinguishable from the main narrative. This approach helps readers locate and focus on key mathematical expressions more effectively.