

ACCA6021 LLOYD CASE 1818917 GRAPHICS AND GAMES PROGRAMMING

by Lloyd Case

Submission date: 29-Nov-2024 02:57PM (UTC+0000)

Submission ID: 244132919

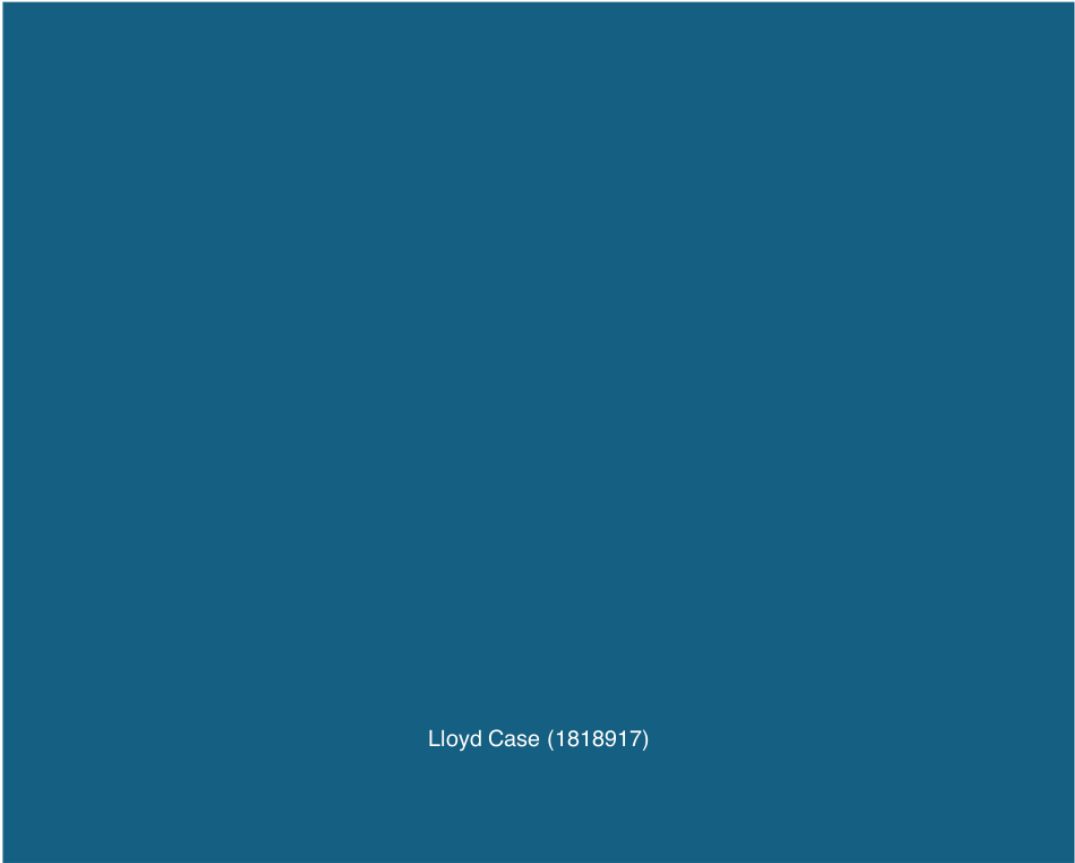
File name:

23995_Lloyd_Case_ACCA6021_LLOYD_CASE_1818917_GRAPHICS_AND_GAMES_PROGRAMMING_628112_284403655.docx
(467.59K)

Word count: 1471

Character count: 8180

ACCA6021
LLOYD CASE 1818917
GRAPHICS AND GAMES
PROGRAMMING



Lloyd Case (1818917)

Table of Contents

Overview of the stages of the graphics pipeline	2
What is the graphics pipeline?	2
Application Stage	2
Key Responsibilities.....	Error! Bookmark not defined.
Geometry Stage	2
Key Responsibilities.....	Error! Bookmark not defined.
Rasterization Stage	2
Key Responsibilities.....	Error! Bookmark not defined.
Fragment Stage.....	2
Key Responsibilities.....	Error! Bookmark not defined.
Output Stage	Error! Bookmark not defined.
Key Responsibilities.....	Error! Bookmark not defined.
Appendix	3
Code.....	3
References.....	7

Overview of the stages of the graphics pipeline

What is the graphics pipeline?

The graphics pipeline is used as a framework for computer graphics that shows the procedures for transforming 3D graphics into 2D graphics on a screen.

When looking at an NVIDIA RTX 4090[1] which is the current top performing GPU on the market, its very clear that it is more than capable at doing way more than is currently necessary. The rendering capabilities of this device enable you as the user to render at 4K resolution with ray tracing enabled and have minimal effect on the performance[1]. Current GPUs are able to process the graphics pipeline in microseconds due to how powerful these devices are. As for performance the top GPUs can process over 100 million triangles a second and render using such complex scenes due to this[2].

Application Stage

The application stage is executed by software on the CPU. This stage involves managing the game logic, user input and all the data needed for rendering. It does this by organising objects as well as applying the transformations and lighting conditions[3].

Some of the key responsibilities of the application stage are scene management, this includes organising what you have created, the objects, their properties and how they interact with everything. It makes sure that objects in the application makes logical sense.

This stage handles user input and interaction such as keyboard presses and mouse clicks and movement. Based on this it also updates the position of the object that you are controlling for example.

Geometry Stage

The geometry stage processes the geometric data of the objects used[4]. This means that this stage transforms the vertex's position from local to screen space. This adjusts the object so that the person looking at the screen can see the intended object correctly.

The key responsibility for the geometry is to apply the transformations needed to view the objects in a 3D space, this includes lighten and any other necessary details required to make the object 3D. This is also required for texturing[5].

Rasterization Stage

The rasterization stage is used for assembling the vertex data into primitives and allows you to use good techniques such as removing what isn't currently on the screen to not waste and resources[6]. This ensures that there are resources saved as there is no need to display what you is currently not on the screen[7].

The key responsibilities are to make sure that the 3D objects are coordinated into a 2D space by applying perspective techniques to do this. This will determine how the space handles how the 3D objects are displayed in the screen[8].

Fragment Stage

The Fragment Stage also known as Pixel Processing Shader, determines the colour of each fragment/pixel. It also applies the textures, lighting and any other visuals you can see on the screen as well as handling transparency. This means that it effectively blends the objects to

look more realistic to what we expect. This also applies to performing depth testing so it doesn't use more resources than necessary and can blur objects in the distance if needed and this will also help with rendering efficiency[9].

Appendix



Code

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/constants.hpp>
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <filesystem>

class Appendix {
private:
    // task 1
    glm::vec3 vertex{ 1.0f, 2.0f, -3.0f }; // vertex each number is x y z
    float rotationAngle{ 30.0f }; // rotation angle float because its a single number
    glm::vec3 translation{ 2.5f, 0.0f, -1.0f }; // translation

    glm::vec3 viewpoint{ 0.0f, 4.0f, -1.0f }; // viewpoint
    glm::vec3 upVector{ 0.0f, 1.0f, 0.0f }; // up vector
    glm::vec3 lookAtPoint{ 0.5f, 0.0f, 0.0f }; // look at point

    float fov{ 38.0f }; // field of view float because its a single number
    float nearPlaneDistance{ 1.5f }; // near plane distance float because its a single number
    float farPlaneDistance{ 80.0f }; // far plane distance float because its a single number

    // task 2
    glm::vec3 planePoint{ 2.0f, 2.5f, 3.0f }; // plane point
    glm::vec3 planeNormal{ 1.0f, 2.0f, 0.0f }; // plane normal
```

```
public:
// task 1
void vertexTransformation() {
    std::cout << "Task 1 Vertex Transformation\n";
    std::cout << "\nVertex: " << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z << ")\n"; //
    this is pulling from glm::vec3 vertex{ 1.0f, 2.0f, -3.0f }; x y z are the 3 numbers

    float rotate = glm::radians(rotationAngle); // this is the rotation angle
    glm::mat4 rotationA = glm::rotate(glm::mat4(1.0f), rotate, glm::vec3(1.0f, 0.0f, 0.0f));
    vertex = glm::vec3(rotationA * glm::vec4(vertex, 1.0f));
    std::cout << "\nRotation Angle: " << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z <<
    << ")\n"; // x y z are variables for the numbers specified above

    glm::mat4 translationA = glm::translate(glm::mat4(1.0f), translation); // translation
    vertex = glm::vec3(translationA * glm::vec4(vertex, 1.0f));
    std::cout << "\nTranslation: " << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z <<
    << ")\n"; // x y z are variables for the numbers specified above

    glm::mat4 viewA = glm::lookAt(viewpoint, lookAtPoint, upVector); // view point
    vertex = glm::vec3(viewA * glm::vec4(vertex, 1.0f));
    std::cout << "\nView Point: " << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z <<
    << ")\n"; // x y z are variables for the numbers specified above

    glm::mat4 projectionA = glm::perspective( // fov
        glm::radians(fov),
        16.0f / 9.0f, // aspect ratio of 16:9
        nearPlaneDistance,
        farPlaneDistance
    );
    glm::vec4 projectedViewVolume = projectionA * glm::vec4(vertex, 1.0f);
    projectedViewVolume /= projectedViewVolume.w;
```

```
std::cout << "\nProjected View Volume: " << "(" << projectedViewVolume.x << ", " <<
projectedViewVolume.y << ", " << projectedViewVolume.z << ")\n";

bool inViewVolume = (std::abs(projectedViewVolume.x) <= 1.0f &&
std::abs(projectedViewVolume.y) <= 1.0f && projectedViewVolume.z >= -1.0f &&
projectedViewVolume.z <= 1.0f);

std::cout << "\nIt will be in view volume!\n"; // i think that it will be in view volume

}

// task 2
void calculatePlaneDistance() {
std::cout << "\nTask 2 Culling Plane\n";

float distance;

std::cout << "\nPlane Point: (" << planePoint.x << ", " << planePoint.y << ", " <<
planePoint.z << ")\n"; // x y z are variables for the numbers specified above

std::cout << "\nPlane Normal: (" << planeNormal.x << ", " << planeNormal.y << ", " <<
planeNormal.z << ")\n"; // x y z are variables for the numbers specified above

std::cout << "\nThe vertex will be culled " << "\n"; // predict that the vertex will be culled
}

void runTask() {
vertexTransformation(); // run the vertex transformation (task 1)
calculatePlaneDistance(); // run the culling plane (task 2)
}

};

int main() {
Appendix task;
task.runTask();
return 0;
}
```

```

}
Microsoft Visual Studio Debug Console
Task 1 Vertex Transformation
Vertex: (1, 2, -3)
Rotation Angle: (1, 3.23205, -1.59888)
Translation: (3.5, 3.23205, -2.59888)
View Point: (-3.84518, -0.075856, -0.776181)
Projected View Volume: (-8.0929, -0.283828, -2.90072)
It will be in view volume!
Task 2 Culling Plane
Plane Point: (2, 2.5, 3)
Plane Normal: (1, 2, 0)
The vertex will be culled
C:\Users\lloyd\Desktop\Assignemnt Lloyd Case\Debug\Assignemnt Lloyd Case.exe (process 22884) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

Tests

```

class Appendix {
private:
    // task 1
    glm::vec3 vertex{ 1.0f, 2.0f, -3.0f }; // vertex each number is x y z
    float rotationAngle{ 30.0f }; // rotation angle float because its a single number
    glm::vec3 translation{ 2.5f, 0.0f, -1.0f }; // translation

    glm::vec3 viewpoint{ 0.0f, 4.0f, -1.0f }; // viewpoint
    glm::vec3 upVector{ 0.0f, 1.0f, 0.0f }; // up vector
    glm::vec3 lookAtPoint{ 0.5f, 0.0f, 0.0f }; // look at point

    float fov{ 38.0f }; // field of view float because its a single number
    float nearPlaneDistance{ 1.5f }; // near plane distance float because its a single number
    float farPlaneDistance{ 80.0f }; // far plane distance float because its a single number

    // task 2
    glm::vec3 planePoint{ 2.0f, 2.5f, 3.0f }; // plane point
    glm::vec3 planeNormal{ 1.0f, 2.0f, 0.0f }; // plane normal

public:
    // task 1
    void vertexTransformation() {
        std::cout << "Task 1 Vertex Transformation\n";
        std::cout << "Vertex: "
            << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z << ")\n"; // this is pulling from glm::vec3 vertex{ 1.0f, 2.0f, -3.0f }; x y z are the 3 numbers

        float rotate = glm::radians(rotationAngle); // this is the rotation angle
        glm::mat4 rotationA = glm::rotate(glm::mat4(1.0f), rotate, glm::vec3(1.0f, 0.0f, 0.0f));
        vertex = glm::vec3(rotationA * glm::vec4(vertex, 1.0f));
        std::cout << "Rotation Angle: "
            << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z << ")\n"; // x y z are variables for the numbers specified above

        glm::mat4 translationA = glm::translate(glm::mat4(1.0f), translation); // translation
        vertex = glm::vec3(translationA * glm::vec4(vertex, 1.0f));
        std::cout << "Translation: "
            << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z << ")\n"; // x y z are variables for the numbers specified above

        glm::mat4 viewA = glm::lookAt(viewpoint, lookAtPoint, upVector); // view point
        vertex = glm::vec3(viewA * glm::vec4(vertex, 1.0f));
        std::cout << "View Point: "
            << "(" << vertex.x << ", " << vertex.y << ", " << vertex.z << ")\n"; // x y z are variables for the numbers specified above
    }
};

```

```
glm::mat4 projectionA = glm::perspective( // fov
    glm::radians(fov),
    16.0f / 9.0f, // aspect ratio of 16:9
    nearPlaneDistance,
    farPlaneDistance
```

Looking at the code to make sure there are no syntax errors. Making sure there are no errors with the code itself that could cause a miscalculation. The code seems correct and does work correctly with no errors. The code is laid out in a nice and understandable way.

```
// task 2
void calculatePlaneDistance() {
    std::cout << "Task 2 Culling Plane\n";

    float distance;

    std::cout << "Plane Point: (" << planePoint.x << ", " << planePoint.y << ", " << planePoint.z << ")\n"; // x y z are variables for the numbers specified above
    std::cout << "Plane Normal: (" << planeNormal.x << ", " << planeNormal.y << ", " << planeNormal.z << ")\n"; // x y z are variables for the numbers specified above
    std::cout << "The vertex will be culled " << "n"; // predict that the vertex will be culled
}
```

The culling plane code is incomplete as was unsure how to code it correctly with a working method, so the code has been left minimal it simply displays the information given in the assignment without any of the culling plane calculations used.



References

References

- [1] . *GeForce RTX 4090*. Available: <https://www.nvidia.com/en-gb/geforce/graphics-cards/40-series/rtx-4090/>.
- [2] Anonymous Anonymous Graphic pipeline. Graphic pipeline. 2023. Available: <https://www.sciencedirect.com/topics/computer-science/graphic-pipeline#:~:text=A%20Graphic%20Pipeline%20is%20a,%2C%20projection%20transformatio,n%2C%20and%20texturing..>
- [3] Anonymous Anonymous GPU gems 2; programming techniques for high-performance graphics and general-purpose computation. GPU gems 2; programming techniques for high-performance graphics and general-purpose computation. *Scitech Book News Scitech Book News* 2929(2), pp. 131. pp. 131. 2005. .
- [4] . *Chapter 2: The Graphics Pipeline*. Available: <https://graphicscompendium.com/intro/01-graphics-pipeline>.
- [5] T. Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki and S. Hillaire. T. Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki and S. Hillaire. *Real-Time Rendering Real-Time Rendering* 2018. DOI: 10.1201/b22086.

[6] . https://en.wikipedia.org/wiki/Graphics_pipeline.

[7] J. F. Blinn. J. F. Blinn. Jim blinn's corner-a trip down the graphics pipeline: Subpixelic particles. Jim blinn's corner-a trip down the graphics pipeline: Subpixelic particles. *CG-M CG-M 1111(5)*, pp. 86–90. pp. 86–90. 1991. . DOI: 10.1109/38.90570.

[8] P. J. Campbell. P. J. Campbell. Reviews -- computer graphics: Principles and practice (2nd ed) by james D. foley, andries van dam, steven K. feiner and john F. hughes. Reviews - computer graphics: Principles and practice (2nd ed) by james D. foley, andries van dam, steven K. feiner and john F. hughes. *Mathematics Magazine Mathematics Magazine 6565(4)*, pp. 277. pp. 277. 1992. .

[9] Anonymous Anonymous GPU gems 2; programming techniques for high-performance graphics and general-purpose computation. GPU gems 2; programming techniques for high-performance graphics and general-purpose computation. *Scitech Book News Scitech Book News 2929(2)*, pp. 131. pp. 131. 2005. .

ACCA6021 LLOYD CASE 1818917 GRAPHICS AND GAMES PROGRAMMING

GRADEMARK REPORT

FINAL GRADE

58/100

GENERAL COMMENTS

Engagement with Literature: 8/15

Knowledge and Understanding: 10/20

Cognitive and Intellectual Skills: 20/30

Practical Skills: 18/30

Transferable Skills: 2/5

Overall 58/100

Comment:

Overall a good attempt, and ultimately part one would have benefitted from a further discussion into the math behind the transformations and a by hand attempt at the calculations.

There are a few typos throughout, a proofread would have solved!

The appendix should have been at the end of the report, after the references and then referred to in the text.

With this in consideration the report is severely under the expected word count of 1600.

see screenshot of how to perform a culling test: with the provided data the point would have been not culled.

<https://imgur.com/a/culling-test-GX02PVJ>

screenshots should be marked as figures with captions, and then a table of figures provided under the table of contents, to allow for easier navigation for the reader and a more professional flow.

Let me know if you want an example or help with any of the mentioned above, happy to discuss.

See inline comments also

PAGE 1
PAGE 2
PAGE 3
PAGE 4
 Comment 1 The appendix would typically fall at the very end of the document
PAGE 5
PAGE 6
PAGE 7
PAGE 8
 Comment 2 no comment
PAGE 9