

Assignment 1 - Ethan Walton-Deaves - Written report

by Ethan Walton-Deaves

Submission date: 18-Nov-2022 02:02PM (UTC+0000)

Submission ID: 189459569

File name: 24855_Ethan_Walton-Deaves_Assignment_1_-_Ethan_Walton-Deaves_-_Written_report_467856_1430949377.doc (11.24M)

Word count: 2024

Character count: 10226

Assignment 1

Games Development

Task 1

Point p in this instance that point is located at (2,-2,0,1)

P is a vertex a point in object space.

When we want to move this point to a position we need to implement various matrices so we can relocate this point where we want it

Task 1 needs us to do two movements:

Translation : Movement along the (X,Y,Z) coordinates

Rotation : Movement in the orientation aspect

Task 1 Written Mathematics

Firstly Translation

To do a translation we first need its matrix.

In this situation its a 4x4 matrix and here is what that looks like

(100tx)

(010ty)

(001tz)

(0001)

T being the numbers on how much you want to move the matrix

These number being (2,0,-1)

subbing these numbers in you get

(1002)

(0100)

(001-1)

(0001)

Next we must multiply this with point P to make the translation.

(1002) (2)

(0100) (-2)

(001-1) (0)

(0001) (1)

When tackling this you must take the first row and multiply with the first column

Then the second row by the column

Then the third

Then the fourth.

Then add the numbers together and you will get a 1 by 4 matrix . This is the point where P is now located.

Next Rotation

Unlike the translation rotation has 3 different matrices. One for each X,Y,Z coordinates.

Here is the three matrices.

Around the X axis

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Around the Y axis

$$\begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Around the Z axis

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In this situation we are rotating around the Y axis.

I am using this matrix. Then multiply with point P

$$\begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ -2 \\ 0 \\ 1 \end{pmatrix}$$

Getting a 1x4 matrix and that is the new orientation of the point

Model Matrix

This helps us by combining all matrices that effect the point P.

Using my task as an example I have to translate and rotate my point.

Understanding this I can just multiply all the steps into one 4x4 matrix that point P will multiply with to get the new position on point P.

Order is important here.

Using my task as a basis I must first

Rotate then translate then multiply point P to get the model matrix called M.

$$M = (R \times T) \times P$$

$$P_{\text{final}} = M \times P$$

Here is the transformation and rotation mathematics by hand

Assignate's / working

Translation
 $P(2, 2, -2)$
 $T = (2, 0, -1)$

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+2 \\ 2+2 \\ -2-1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ -3 \\ 1 \end{bmatrix}$$
 $P_1(4, 2, -3, 1)$

Rotation about y Rotate about 4.5° about the y

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ -3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1) (\cos 4.5^\circ \times 4) + (\sin 4.5^\circ \times -3) \\ 2) 2 \\ 3) (-\sin 4.5^\circ \times 4) + (\cos 4.5^\circ \times -3) \\ 4) 1 \end{bmatrix}$$

1) ≈ 0.707 $P_2 \begin{bmatrix} 0.707 \\ 2 \\ -4.950 \\ 1 \end{bmatrix}$
2) 2
3) ≈ -1.950
4) 1

Here is the workings out though hand and multiplying the two 4x4 matrices

P

$$\begin{bmatrix} 3.0777 & 0 & 0 & 0 \\ 0 & -2.0777 & 0 & 0 \\ 0 & 0 & 1.0499 & 1.000 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$R_1 C_1 (1 \times 3.0777) + (0 \times 0) + (0 \times 0) + (0 \times 0)$
 $R_1 C_2 (1 \times 0) + (0 \times -2.0777) + (0 \times 0) + (0 \times 0)$
 $R_1 C_3 (1 \times 0) + (0 \times 0) + (0 \times 1.0499) + (0 \times 1)$
 $R_1 C_4 (1 \times 0) + (0 \times 0) + (0 \times 0) + (0 \times 1)$
 $R_2 C_1 (0 \times 3.0777) + (-1 \times 0) + (0 \times 0) + (0 \times 0)$
 $R_2 C_2 (0 \times 0) + (-1 \times -2.0777) + (0 \times 0) + (0 \times 0)$
 $R_2 C_3 (0 \times 0) + (-1 \times 0) + (0 \times 1.0499) + (0 \times 1)$
 $R_2 C_4 (0 \times 0) + (-1 \times 0) + (0 \times 0) + (0 \times 1)$
 $R_3 C_1 (0 \times 3.0777) + (0 \times 0) + (-2.0244 \times 0) + (1.0588 \times 0)$
 $R_3 C_2 (0 \times 0) + (0 \times -2.0777) + (-2.0244 \times 0) + (1.0588 \times 0)$
 $R_3 C_3 (0 \times 0) + (0 \times 0) + (-2.0244 \times 1.0499) + (1.0588 \times 1)$
 $R_3 C_4 (0 \times 0) + (0 \times 0) + (-2.0244 \times 1.000) + (1.0588 \times 1)$
 $R_4 C_1 (0 \times 3.0777) + (0 \times 0) + (0 \times 0) + (1 \times 0)$
 $R_4 C_2 (0 \times 0) + (0 \times -2.0777) + (0 \times 0) + (1 \times 0)$
 $R_4 C_3 (0 \times 0) + (0 \times 0) + (0 \times 1.0499) + (1 \times 1)$
 $R_4 C_4 (0 \times 0) + (0 \times 0) + (0 \times 0) + (1 \times 1)$

Perspective Projection Projection

This is used to project vertices within a space that is dictated by the aspect ratio, the near and far planes and the field of view. It also edits the scale of the vertex based off the distance away from the camera point to a plane where everything is displayed (near plane).

Here is a few equations before inputting things into its matrix

$$q = 1/\tan(\text{field of view} / 2)$$

$$A = q / \text{aspectratio}$$

$$B = z_{\text{near}} + z_{\text{far}} / z_{\text{near}} - z_{\text{far}}$$

$$C = 2 \times (z_{\text{near}} \times z_{\text{far}}) / (z_{\text{near}} - z_{\text{far}})$$

Here is that matrix. Above is the equations to sub into the matrix bellow

$$(A \ 0 \ 0 \ 0)$$

$$(0 \ q \ 0 \ 0)$$

$$(0 \ 0 \ B \ C)$$

$$(0 \ 0 \ -1 \ 0)$$

This is needed in part 3 of task 1.

Orthographic transformation

This is another matrix built by using many of the above objects such as z_{near} and z_{far} . This helps determine what gets rendered when showing pixels on the projection plane. This matrix helps us determine the space where the camera view is looking at and allows the rendering inside that space. Anything not in that space will not be rendered. This is also called culling and is a part of task 2.

L,R,T,B Values are corresponding to the height of the screen and the width of the screen normally in this case we will assign these numbers to the objects as follows

$$L = -1$$

$$R = 1$$

$$T = -1$$

$$B = 1$$

This makes a square screen where pixels will be shown. This also means that if the number is outside these coordinates then it will not be in screen view and not be loaded.

Perspective Projection matrix

Here is the matrix

$$(f/\text{aspect} \ 0 \ 0 \ 0)$$

$$(0 \ f \ 0 \ 0)$$

$$(0 \ 0 \ z_{\text{near}}+z_{\text{far}}/z_{\text{near}}-z_{\text{far}} \ 2 \cdot z_{\text{near}} \cdot z_{\text{far}}/z_{\text{near}} - z_{\text{far}})$$

$$(0 \ 0 \ 0 \ -1 \ 0)$$

Model view

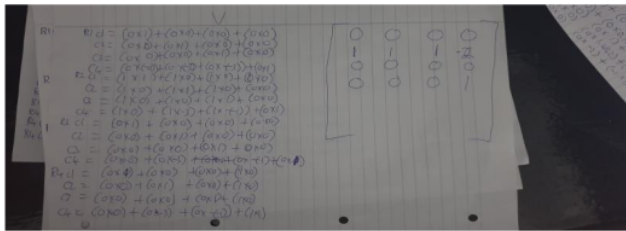
This is where you create the “eye” of the world that will look into the world

The “eye” or camera sits on the origin point looking down the -z axes since this is a right handed system.

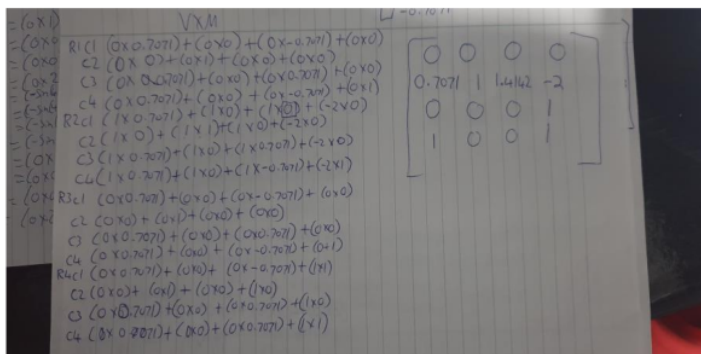
When we want to “move” the camera instead of the camera moving we just move the whole world instead. If we go back to our matrices translation and rotation and get the camera point we must get the camera to the origin so we need to do a inverse translate or a inverse rotate. Here are those matrices.

Translation	Rotation
$\begin{pmatrix} 1 & 0 & 0 & -tx \\ 0 & 1 & 0 & -ty \\ 0 & 0 & 1 & -tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} u & u & u & 0 \\ v & v & v & 0 \\ n & n & n & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Taking these two matrices and find the M for view
see below picture for the multiplication of the 2 matrices



In doing this every vertex will be pushed to there new point in world space. This push will be inverse so it looks like the camera is moving positivity. Once found we can find Mview by multiplying the V with the M as seen below.

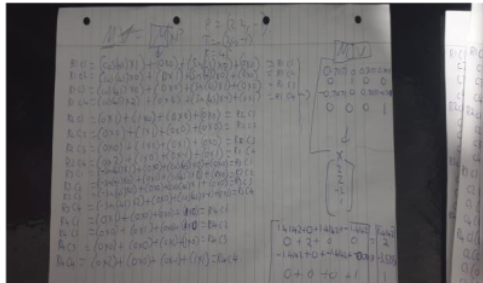
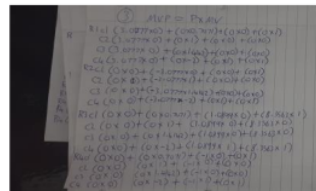
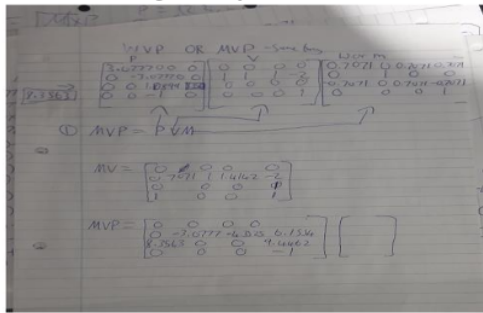


projection model view

mvp = Perspective X modelView X M

This is the last matrix we need so that we can use it to transfer the point to worlds space then add a perspective to it. So firstly we multiply modelView by the model. Then after getting modelviewmodel multiply perspective to get the mvp. We can then use this against any vertex we want to implement in our world

Here is the workings out by hand



Task 2

culling

Is the process that helps the program by de-loading any vertexes outside the view space.

Given the numbers $p = (2.5, 2, 4)$ and $n = (-1, 2, 0)$ while $z = 0$

Using the plane equation

$$ax - x + by - y + cz - z = 0$$

After subbing in theses numbers you get the equation

$$-1(X-2.5) + 2(Y-2) + 0(Z-4) = 0$$

Which gives us

$$2x - 4y + 3 = 0$$

Task 3

Important lines of code used and what they are used for

Line 1

`glm::vec4 P1(2.0, 2.0, -2.0, 1.0);`

Explanation

GLM is the math library I will be using to help with my coding of these matrices.
The vector which was called point P in task 1

Line 2

```
glm::mat4 Proj = glm::perspective(glm::radians(36.0f), 1.0f, 2.0f, 70.0f);
```

Explanation

The start of the perspective matrix and adding all the objects needed to create that matrix.
This includes the field of view , the aspect ratio, near plain, far plain respectively.

Line 3

```
glm::mat4 Translate1 = glm::translate(glm::mat4(1.0f), glm::vec3(2.0f, 0.0f, -1.0f));
```

Explanation

This is a translate matrix and is assigned 3 points to move the Point P using the X,Y,Z coordinates respectively. This is seen in the vec 3 of the line. That is an input point for the X,Y,Z coordinates.

Line 4

```
glm::mat4 Rotate1 = glm::rotate(glm::mat4(1.0f), glm::radians(45.0f), glm::vec3(0, 1, 0));
```

Explanation

This is the rotation matrix. It is given a angle and is turned into radians then we need to clarify which axes we are using. This also uses X,Y,Z coordinates and a 1 is placed in the orientation coordinates that is needed.

Line 5

```
glm::mat4 View = glm::lookAt(glm::vec3(0, 3, -1), glm::vec3(1.5, 0, 0), glm::vec3(0, 1, 0));
```

Explanation

Look at uses a viewpoint, U,V,N and lookat point to determine the inverse that all points must travel in order for the camera to be at orogen.

Line 6

```
glm::mat4 M = Rotate1 * Translate1;
```

Explanation

This is the calculate the ModelMatrix (M) multiplying the rotation 4x4 matrix by the translation 4x4 matrix

Line 7

```
glm::vec4 pcheck = M * P1;  
cout << "P in object ";  
cout << "(" << pcheck.x << ", " << pcheck.y << ", " << pcheck.z << ")"<<endl;
```

Explanation

To display the model matrix in the launch window and to see the numbers calculated so far by multiplying the modelmatrix by the point P.

Line 8

```
glm::mat4 MV = View * M;
```

Explanation

This is the ModelView matrix which is calculated by the View matrix multiplied by the model matrix. Both are 4x4 matrices.

Line 9

```
glm::mat4 MVP = Proj * MV;
```

Explanation

This is the matrix that adds perspective projection. Both are 4x4 matrices

Line 10

```
glm::vec4 pcheck2 = MVP * P1;
```

Explanation

To see where the point is in perspective view space

Bibliography

[1]	Stackoverflow – GLUlookat specification Available at: https://stackoverflow.com/questions/42977477/glulookat-specification?rq=1
[2]	The cherno " projection matrices in Opengl" Mar 7 , 2018 [online] Available at: https://youtu.be/xZs6K7VLM7A
[3]	Khronos – gluLookat – define a viewing transformation Available at: https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluLookAt.xml
[4]	Research ncl – Vertex transformation Available at: https://research.ncl.ac.uk/game/mastersdegree/graphicforgames/vertextransformation/Tutorial%20%20-%20Vertex%20Transformation.pdf
[5]	Pikuma "Perspective Prijection Matrix(Math for Game Developers)" Oct 8 ,2021 Available at: https://youtu.be/EqNcqBdrNyI
[6]	The Chernov "Model view projection matrices in Opengl" Apr 4 ,2018 Available at: https://youtu.be/x_Ph2cuEWRE
[7]	The Chernov "Math in OpenGL" Feb 25 , 2018 Available at: https://youtu.be/VuYnjsDOx60
[8]	UCSC CSE160 "Model Matrix or view matrix?" Feb 13, 2020 Available at: https://youtu.be/tQV84NyYMKQ
[9]	Jeffery Chastine " Tutorial 10 – Introduction to Matrix Math in OpenGL" Mar 12, 2013 Available at: https://youtu.be/V6sJpjYzfaQ
[10]	GSN Composer " OpenGL Modelview Matrix and 3D Transformations [Shaders Monthly #3]" Jan 7 , 2022 Available at: https://youtu.be/cKbC-Jkd-3I
[11]	UCSC CSE160 "Lookat() - explain what it is" Feb 13, 2020 Available at: https://youtu.be/1SMtxVoAaMc
[12]	Scratchapixel – The perspective and Orthographic Projection Matrix Available at: https://www.scratchapixel.com/lessons/3d-basic-

	rendering/perspective-and-orthographic-projection-matrix/projection-matrix-introduction
[13]	Scratchapixel – The OpenGL Perspective Projection Matrix Available at: https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix
[14]	OpenGL-tutorial – Tutorial 3 : Matrices Available at: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/#the-model-view-and-projection-matrices
[15]	Learn OpenGL -Face culling Available at: https://learnopengl.com/Advanced-OpenGL/Face-culling
[16]	OnlineMSchool – Online calculator Available at: https://onlinemschool.com/math/assistance/cartesian_coordinate/plane/

Appendix

```
#include "glm/glm.hpp"

#include "glm/ext.hpp"

#include <iostream>

using namespace std;

int main(void)
{
    //Basic code for transformations

    //Vertex 2,2,-2
    //rotation 45
    //translation 2,0,-1
    // //viewpoint (0,3,-1)
    //up vector (0,1,0)
    //lookat point (1.5,0,0)
    //field of view 360
    //near plane dis (2)
    //far plane dis (70)

    //glm::vec4 p1(2.0, 2.0, -2.0, 1);
```

```

    ///translation
    //float lx = 2, ly = 0, lz = -1;
    //glm::mat4 T1 = glm::mat4(1.0, 0.0, 0.0, 0.0,
    //    0.0, 1.0, 0.0, 0.0,
    //    0.0, 0.0, 1.0, 0.0,
    //    lx, ly, lz, 1.0);
    ///roation
    //
    //float theta = glm::pi<float>() / 4.0; // rotate 45deg ??
    //glm::mat4 R1 = glm::mat4(glm::cos(theta), 0.0, -glm::sin(theta), 0.0,
    //    0.0, 1.0, 0.0, 0.0,
    //    glm::sin(theta), 0.0, glm::cos(theta), 0.0,
    //    0.0, 0.0, 0.0, 1.0);
    //glm::vec4 P1 = T1 * p1;
    //glm::vec4 P2 = R1 * P1;
    //glm::mat4 M(1.0);
    //M = M * T1 * R1;
    //glm::vec4 P3 = M * p1;
    //cout << "T1 ";
    //cout << "(" << P1.x << ", " << P1.y << ", " << P1.z << ")" << endl;
    //cout << "R1 ";
    //cout << "(" << P2.x << ", " << P2.y << ", " << P2.z << ")" << endl;
    //cout << "M ";
    //cout << "(" << P3.x << ", " << P3.y << ", " << P3.z << ")"<<endl;

    glm::vec4 P1(2.0, 2.0, -2.0, 1.0);

    //perspective

    glm::mat4 Proj = glm::perspective(glm::radians(36.0f), 1.0f, 2.0f, 70.0f);
    //glm::mat4 frus = (-1, 1, -1, 1, 2, 70, Proj);

    //view of camera / Culling everything out of the picture
    //glm::mat4 Proj2 = glm::ortho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    //translation
    glm::mat4 Translate1 = glm::translate(glm::mat4(1.0f), glm::vec3(2.0f, 0.0f, -1.0f));
    //roation
    glm::mat4 Rotate1 = glm::rotate(glm::mat4(1.0f), glm::radians(45.0f), glm::vec3(0, 1,
0));
    //View
    glm::mat4 View = glm::lookAt(glm::vec3(0, 3, -1), glm::vec3(1.5, 0, 0), glm::vec3(0,
1, 0));

    //glm::vec4 pa = Translate1 * P1;

    glm::vec4 pb = Rotate1 * pa;*/

    glm::mat4 M = Rotate1 * Translate1;

    glm::vec4 pcheck = M * P1;
    cout << "P in object ";
    cout << "(" << pcheck.x << ", " << pcheck.y << ", " << pcheck.z << ")"<<endl;

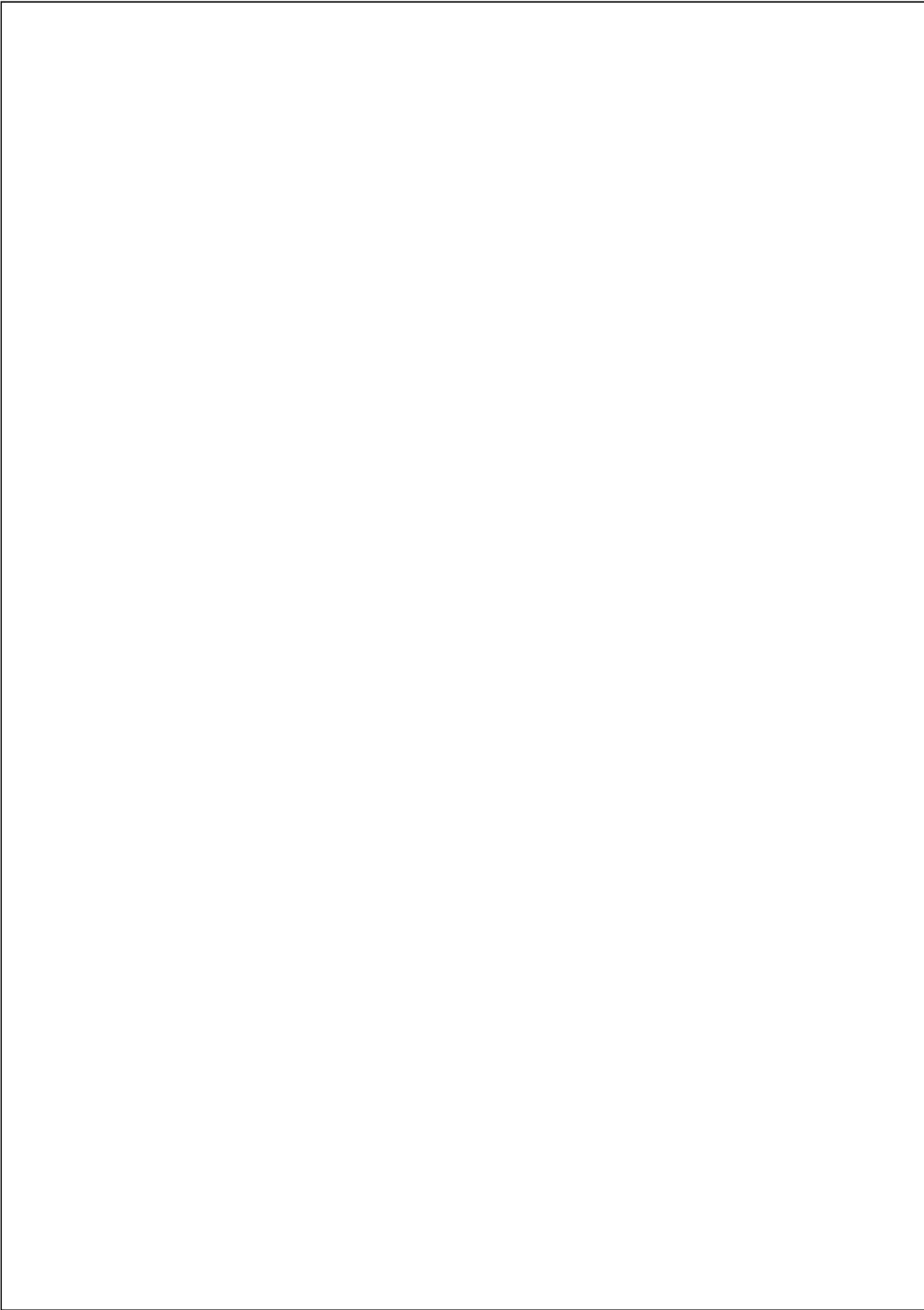
    glm::mat4 MV = View * M;

    glm::mat4 MVP = Proj * MV; // * M;

    glm::vec4 pcheck2 = MVP * P1;

    cout << "P in world added projection ";
    cout << "(" << pcheck2.x << ", " << pcheck2.y << ", " << pcheck2.z << ")" << endl;
}

```



Assignment 1 - Ethan Walton-Deaves - Written report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

68/100

PAGE 1



Comment 1

Should be (2,2.-2)

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

GRADING FORM: SMU GENERIC_GJ

RESEARCH-INFORMED LITERATURE



Your report covers the practical elements very well but a review of the graphics pipeline and current methods used in GPUs and graphics is lacking somewhat. Missed out??
Otherwise a detailed walkthrough of the pipeline processes presented in the maths. Occasional mistakes evident.

6

Use inline citations to support arguments, assertions, etc.

KNOWLEDGE AND UNDERSTANDING



A very good understanding of the pipeline - especially the vertex geometry aspects. The maths had some errors - your camera transformation needs further work. Compare this with the results of the glm::lookAt() transform calculator.

15

tCorrect plane eqn used - well done. SOme further context required here when presenting your results.

ANALYSIS



Appropriate transformations and linear algebra investigated and used to transform points through the pipeline. Well done but be aware of the issue with the model view transform - it was incorrectly formed. See `glm::lookAt()` for correct values. It would be good to compare these in your report.

21

PRACTICAL APPLICATION AND DEP.



Some good work here demonstrating the mathematical operations required for transformations and other linear algebra.

23

SKILLS FOR PROFESSIONAL PRAC.



Use refs in your report. Bibliography provided
Avoid too many electronic references. Use an equation editor to improve presentation of your work.

2.5