

AdamWilliams_GGP_CW1

by Adam Williams

Submission date: 25-Nov-2022 04:14AM (UTC+0000)

Submission ID: 190040591

File name: 25476_Adam_Williams_AdamWilliams_GGP_CW1_467856_1873567158.docx (110.29K)

Word count: 869

Character count: 4431

Title Page

Module name:

Graphics and Games Programming (ACCA6021)

Assignment Title:

Application of Graphics Principles

Student Name & Number:

Adam Williams (2008159)

Task 1

Translating a Matrix

Starting off you have to turn your vector into a homogenous coordinate. Homogenous coordinates have an extra coordinate, which is referred to as “w” which is usually set at a value of 1.0. Once this is done you can then do a vertex transformation which the formula for is shown below.

$$\begin{bmatrix} m0 & m4 & m8 & m12 \\ m1 & m5 & m9 & m13 \\ m2 & m6 & m10 & m14 \\ m3 & m7 & m11 & m15 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} xm0 + ym4 + zm8 + wm12 \\ xm1 + ym5 + zm9 + wm13 \\ xm2 + ym6 + zm10 + wm14 \\ xm3 + ym7 + zm11 + wm15 \end{bmatrix}$$

An example of this with the numbers I was given is here.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 + 0 + 0 + 0 \\ 0 + 3 + 0 + 0 \\ 0 + 0 + -2 + 0 \\ 0 + 0 + 0 + 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ -2 \\ 1 \end{bmatrix}$$

Applying a translation distance

An example of applying a translation distance to a 4-coordinate vector is shown here. The numbers to start with are from my program, and here are the calculations step by step to show how the program does the calculations.

$$\begin{bmatrix} 1 & 0 & 0 & 1.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1.68835 \\ 2.67385 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} (-1.68835 * 1) + (2.67385 * 0) + (-2 * 0) + (1 * 1.5) \\ (-1.68835 * 0) + (2.67385 * 1) + (-2 * 0) + (1 * 0) \\ (-1.68835 * 0) + (2.67385 * 0) + (-2 * 1) + (1 * -1) \\ (-1.68835 * 0) + (2.67385 * 0) + (-2 * 0) + (1 * 1) \end{bmatrix} = \begin{bmatrix} -1.68835 + 0 + 0 + 1.5 \\ 0 + 2.67385 + 0 + 0 \\ 0 + 0 + -2 + -1 \\ 0 + 0 + 0 + 1 \end{bmatrix} = \begin{bmatrix} -0.18353 \\ 2.67385 \\ -3 \\ 1 \end{bmatrix}$$

Task 3

Including "openGL" Directories

```
1  #include "glm/glm.hpp" // vec2, vec3, mat4, radians
2  #include "glm/ext.hpp" // perspective, translate, rotate
3  #include <GL/glew.h>
4  #include "GLFW/glfw3.h"
5  #include <GL/glu.h>
6  #include <GL/gl.h>
7  #include <iostream>
8
9  using namespace std;
```

I start off the code making sure to include all the necessary "openGL" directories I will be using.

Main Code

```
10
11  int main(void)
12  {
13      //defining starting point
14      glm::vec4 p(1.0, 3.0, -2, 1);
15      cout << "point, p is " << "(" << p.x << ", " << p.y << ", " << p.z << ", " << p.w << ")" << endl;
16  }
```

The main source code starts off by defining the starting point as a "vec4". This is a 4-coordinate vector which is necessary as it allows the vector to be homogenous. I then output the point back to the user.

```
17      // define translation vector
18      glm::vec4 t(1.5, 0, -1, 1);
19      //create the translation matrix T2
20      glm::mat4 T2 = glm::mat4(1.0, 0.0, 0.0, 0.0, // note this is the leftmost column, not the top row
21                              0.0, 1.0, 0.0, 0.0,
22                              0.0, 0.0, 1.0, 0.0,
23                              t.x, t.y, t.z, 1.0);
24
25      //define rotation matrix
26      float rotation = 70;
27      glm::mat4 R = glm::mat4(glm::cos(rotation), glm::sin(rotation), 0.0, 0.0, // note this is the leftmost column, not the top row
28                              -glm::sin(rotation), glm::cos(rotation), 0.0, 0.0,
29                              0.0, 0.0, 1.0, 0.0,
30                              0.0, 0.0, 0.0, 1.0);
31
```

Then the two translation matrices are setup, the first one to be setup is labelled "T2", which is used for the translation distance calculation, and the second one setup is labelled "R". These translation matrices will be used later by the current vector for the needed calculation.

"T2" firstly declares the translation distance vector, and then uses those values from that vector within the matrix.

"R" declares the float known as rotation which is used within the matrix for the cos and sin calculations.

```

32     cout << "p will be rotated by 70 degrees" << endl;
33     glm::vec4 P2 = R * p; //rotate it
34     cout << "the new value of p is " << "(" << P2.x << ", " << P2.y << ", " << P2.z << ", " << P2.w << ")" << endl;
35     cout << "t is " << "(" << t.x << ", " << t.y << ", " << t.z << ", " << t.w << ")" << endl;
36     glm::vec4 P3 = T2 * P2; //translate it
37     cout << "the new value of p is " << "(" << P3.x << ", " << P3.y << ", " << P3.z << ", " << P3.w << ")" << endl;
38
39     return 0;
40

```

This final section runs the calculations and outputs them to the user.

The first calculation of this section starts by declaring a new “vec4” known as “P2” to store the answer into. This then takes “p” which is the starting point vector and calculates it with the rotation matrix that was setup. The answer is then displayed to the user.

The next calculation uses the “P2” vector as this is the new rotation of the vector, a new “vec4” is created called “P3” to store the answer off this calculation. “P2” is then used with the distance matrix that was setup to get the answer. That answer is then out to the user.

Camera Space and Viewpoint

```

42 void setProjectionMatrix(int width, int height, float conditionalaspect)
43 {
44     glMatrixMode(GL_PROJECTION);
45     glLoadIdentity();
46     if ((float)width / (float)height < conditionalaspect)
47         float zoomfactor = ((float)width / (float)height) / conditionalaspect;
48     }
49     gluPerspective(atan(tan(50.0 * 3.14159 / 360.0) / zoomfactor) * 360.0 / 3.14159, (float)width / (float)height, 1.5, 100);
50     // last two values are near plane and far plane respectively
51     gluLookAt(0, 6, -1, //viewpoint x y z
52             1.5, 0, 0, //look at point x y z
53             0, 1, 0); // up vector
54
55
56

```

The camera viewpoint kept running into errors in testing.

When using “gluLookAt” to setup the camera view the first 3 values represent the x, y and z coordinates of the camera’s viewpoint, values 4 to 6 represent the x, y and z coordinates of the camera’s “look at” point, and the final 3 values represents the up vector coordinates of the camera. I also used “gluPerspective” to set the near and far plane, the final 2 values within the “gluPerspective” are for the near and far plane respectively.

Final Output

```

point, p is (1,3,-2,1)
p will be rotated by 70 degrees
the new value of p is (-1.68835,2.67385,-2,1)
t is (1.5,0,-1,1) and will be used to translate p
the new value of p is (-0.188353,2.67385,-3,1)

```

These are the results output by the program. As shown here the result for the translation distance calculations match up with my step by step workings.

Appendix

```
#include "glm/glm.hpp" // vec2, vec3, mat4, radians
#include "glm/ext.hpp" // perspective, translate, rotate
#include <GL/glew.h>
#include "GLFW/glfw3.h"
#include <GL/glu.h>
#include <GL/gl.h>
#include <iostream>

using namespace std;

int main(void)
{
    //defining starting point
    glm::vec4 p(1.0, 3.0, -2, 1);
    cout << "point, p is " << "(" << p.x << ", " << p.y << ", " << p.z << ", " << p.w
    << ")" << endl;

    // define translation vector
    glm::vec4 t(1.5, 0, -1, 1);
    //create the translation matrix T2
    glm::mat4 T2 = glm::mat4(1.0, 0.0, 0.0, 0.0, // note this is the leftmost
column, not the top row
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        t.x, t.y, t.z, 1.0);

    //define rotation matrix
    float rotation = 70;
    glm::mat4 R = glm::mat4(glm::cos(rotation), glm::sin(rotation), 0.0, 0.0, //
note this is the leftmost column, not the top row
        -glm::sin(rotation), glm::cos(rotation), 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0);

    cout << "p will be rotated by 70 degrees" << endl;
    glm::vec4 P2 = R * p; //rotate it
    cout << "the new value of p is " << "(" << P2.x << ", " << P2.y << ", " << P2.z
    << ", " << P2.w << ")" << endl;
    cout << "t is " << "(" << t.x << ", " << t.y << ", " << t.z << ", " << t.w << ")"
    and will be used to translate p" << endl;
    glm::vec4 P3 = T2 * P2; //translate it
    cout << "the new value of p is " << "(" << P3.x << ", " << P3.y << ", " << P3.z
    << ", " << P3.w << ")" << endl;

    return 0;
}

/*void setProjectionMatrix(int width, int height, float conditionalaspect)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if ((float)width / (float)height < conditionalaspect)
        float zoomfactor = ((float)width / (float)height) / conditionalaspect;
    }
    gluPerspective(atan(tan(50.0 * 3.14159 / 360.0) / zoomFactor) * 360.0 /
3.14159, (float)width / (float)height, 1.5, 100);
    // last two values are near plane and far plane respectively
    gluLookAt(0, 6, -1, //viewpoint x y z
        1.5, 0, 0, //look at point x y z
        0, 1, 0); // up vector
```

}*/

FINAL GRADE

GENERAL COMMENTS

Instructor

52/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

GRADING FORM: SMU GENERIC_GJ

ADAM WILLIAMS

52

RESEARCH-INFORMED LITERATURE



Please use refs! You must use them!!
It's clear you looked at appropriate documentation
for code at least. You also applied some

7.5

transformations correctly so please ref your sources.

It would have been good to see a review of the graphics pipeline and current gpu methods/ state of the art discussions here too.

KNOWLEDGE AND UNDERSTANDING

 You show an essential understanding of the geometric/vertex transformations used and applied this understanding to create appropriate matrices - though you transformed about the x axis not the required y as asked for. You are aware of the need to reverse the order of the matrix mults. to apply "local" thinking to the transformation as requested. Well done on this. The perspective and view point coding was not completed - although I note you did attempt it. Try to document the errors and show investigation of the debug process you applied, even if it didn't work.

11

ANALYSIS

 See previous section for comments regarding this. You missed out the last part - task 3.

16

PRACTICAL APPLICATION AND DEP.

 Some essential ideas applied. Some later work (perspective transforms, etc need to be completed - same with task 3 (culling a point).

15

SKILLS FOR PROFESSIONAL PRAC.



Structure clear - parts missing and no refs. But you have taken the time to document/present matrices clearly.

2.5