Xuan(James) Zhai

CS 7/5382 – S22 – Midterm Take Home Written Portion

CS 5382/7382 Midterm Exam Spring 2022 – Written Portion
(40 points towards total Midterm Grade)

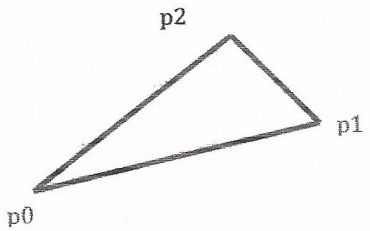
Please download, print and WRITE your answers in the space provided
Upload a scanned pdf before midnight Tuesday, March 22.

This exam is open book, open notes but is to be independent work. You may use other resources but your answers must be your own and show your own understanding. You may be asked to discuss at the oral exam.

- Cutting and pasting from online source is NOT allowed and substantial similarity from other sources may result in loss of credit.
- Working with other students is NOT ALLOWED in any form

---

1) (8 pnts) Consider the following triangle, with the front face in gray as shown here and defined by the vertices $p0, p1, p2$ in a right hand system.



Which of the following, if any, will compute a vector that is orthogonal to the plane that is defined by the triangle that points in the outward facing direction? (Please note very precisely the symbols used.)

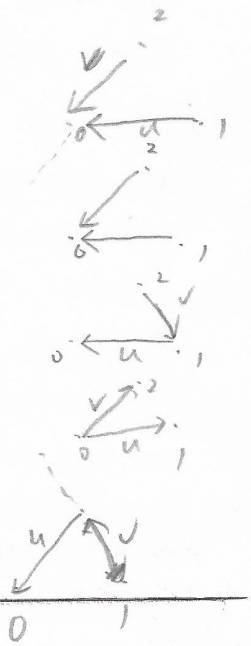
A) $u \times v$ *where* $u = p0 - p1$ *and* $v = p0 - p2$

B) $v \times u$ *where* $u = p0 - p1$ *and* $v = p0 - p2$

C) $u \times v$ *where* $u = p0 - p1$ *and* $v = p1 - p2$

D) $v \times u$ *where* $u = p1 - p0$ *and* $v = p2 - p0$

E) $u \times v$ *where* $u = p0 - p2$ *and* $v = p2 - p1$

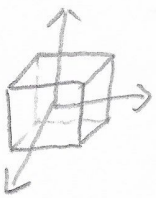


CS 7/5382 – S22 – Midterm Take Home Written Portion

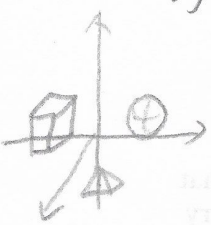
2) (8 pnts) Write an exam question and acceptable answer that illustrates an understanding of the relationship between the camera frame, the world frame and the viewing volume. This should contain components that illustrate the (a) concepts, the (b) mathematical foundations and (c) computational techniques employed in a GPU implementation of the framework and its role in image formation. For full points, the question and/or answer should include clarity and unambiguity of expression using mathematical expression. It may also include sketches and code snippets for clarity.

Q: What are the coordinate systems in OpenGL, what are the matrices, and why we need them?

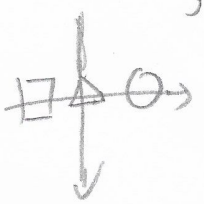
A. When we want to draw something with OpenGL, there are things we need to figure out first. What we want to draw? where is it? where are we looking from? How do we look at it?



1) The step one is draw your object at the local space. Your object will always be constructed by vertices, and in the local space they will be related to the origin (0,0,0). For each vertex, it will have x y z w data.



2) The second step is to move it to the world space, because in the world space you may have more than one object, and you need to determine their positions. To do so, you will need to do transformations like ~~tr~~translation, rotation, etc. To do that you will need to use the model matrix, it will do all the transformation. When you do the multiplication "Mxp", you will get the relative position of your objects (vertices) in the world coordinates.

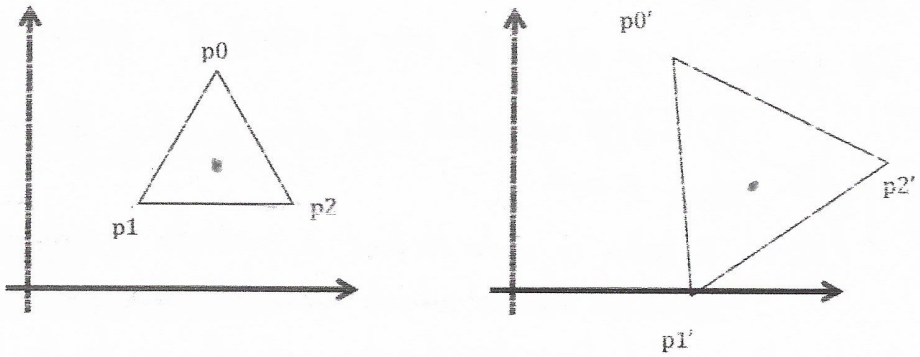


3) The step three is to determine where are we looking at it from. By default we are looking at the objects from (0,0,1), but we can change it to where we want. How to do it? It is the view matrix V. Every transformation is relative, changing view position can be regarded as relatively change the objects' position. To get the view matrix, we need to use the look at function with eye position, up direction, and at position. We need the up direction because we may rotate our eye when we look at it. We now just do the multiplication "VxMxp".

4) The last step is to determine How we want to see it. For example, for human beings, our view range is like a sector, with maximum wide about 120°, and the view is a perspective projection. Thus, we need to think about, how far/near we can see; how high/wide we can see, and whether we see it from a point (perspective) or a plane (orthog) (view volume). So we need the projection matrix (P) and perspective division (using w). We finally get the equation: $q = P * V * M * p$ which is (in the clip coordinates (After NDC and Depth test, the position) It will eventually become a point in 2D to project)

CS 7/5382 – S22 – Midterm Take Home Written Portion

3) (8 pnts) Below is a representation of a triangle, on the left, and the results of applying a series of transformations on the right, in which the triangle appears to rotate about its own center point by an angle θ while doubling uniformly in size. (Note that gray circle in the figure is intended to illustrate the center point of the triangle in both figures. It is the point about which rotation occurs. It has the same coordinates in both the left and right figures.)



What is the specific series of transformations that, when applied to *p0, p1, p2*, would result in *p0', p'1, p2'* to achieve this result.
Express this precisely in terms of *p0, p1, p2* and θ

1) The first step is to get the center $\begin{cases} C_x = \frac{P_{0x}+P_{1x}+P_{2x}}{3} \\ C_y = \frac{P_{0y}+P_{1y}+P_{2y}}{3} \end{cases}$

2) The second step is a translation that move the center to original

$$P_0' = P_0 - C \; ; \; P_1' = P_1 - C \; ; \; P_2' = P_2 - C$$

3) The next step is a rotation, if θ is in a counter-clockwise rotation

$$P_0' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} P_0' \; ; \; P_1' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} P_1' \; ; \; P_2' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} P_2'$$

4) Step four is scaling ; if the size is uniformly doubled :

$$P_0' = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} P_0' \; ; \; P_1' = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} P_1' \; ; \; P_2' = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} P_2' \; ;$$

5) The final step is to move the center back,

$$P_0' = P_0' + C \; ; \; P_1' = P_1' + C \; ; \; P_2' = P_2' + C \; ;$$

CS 7/5382 – S22 – Midterm Take Home Written Portion

4) (8 pnts) What is an affine transformation and why is it important in computer graphics? Address the characteristics of an affine transformation in terms of mathematical foundations, representation and computational framework in a graphics pipeline implementation. For most credit, be precise in your answer and make clear and focused connections between the theoretical and computation frameworks.

1) By definition, Affine transformation is a (class) of mathematical operations. Geometrically, or in computer Graphics, It represents the transformations that will keep the straightness and parallelness of the object, but not necessarily the angle or distances between points. Affine transformation mainly encompass ① Translation ② Rotation ③ Scaling ④ shearing.

2) One of the simplest transformation in geometry is linear transformation, which can do scaling, shearing, and rotation.
In a 2D space, for a object [a series of vertices] "V", we can apply the transformation matrix:

$$\left(\begin{bmatrix} ① & ② \\ ③ & ④ \end{bmatrix} V \right)$$

① and ④ will control scaling along x and y, ② and ③ will control shearing along x and y.
← Rotation is, under the hood, a combination of scaling and shearing [2 shearing + 1 scaling], so it can also be achieved by modifying those four numbers.

3) Affine transformations extend linear transformation and allow us to do translation such that.

$$v_1 = \begin{bmatrix} ① & ② \\ ③ & ④ \end{bmatrix} V + \begin{bmatrix} ⑤ \\ ⑥ \end{bmatrix}$$, where ⑤ and ⑥ will determine the translation on x and y.

4) To represent affine transformation with matrices, we will use homogeneous coordinates. This means representing a 2-vector (x, y) as a 3-vector (x, y, 1). Thus, the matrix becomes.

$$\begin{bmatrix} ① & ② & ⑤ \\ ③ & ④ & ⑥ \\ 0 & 0 & 1 \end{bmatrix}$$ → (in opengl we will make it a 4x4 matrix (with w) for projection)

5) Affine transformation is important because it allows us to map vertices with the transformation of object. Moreover, it allows the object to move from a local space, to a world space (even a view space) with the model view matrix we applied.

CS 7/5382 – S22 – Midterm Take Home Written Portion

5) (8 pnts) Describe what happens in each of the following WebGL functions as it relates to graphics state and shader execution. Include enough detail to differentiate each from others that may be similar. If a

a) bindBuffer
b) bufferData
c) drawArrays
d) viewport
e) uniform1f

Based on so far what we've learned

a) After you created a buffer in GPU and got a pointer reference, you will need to bind the pointer with graphics state ARRAY_BUFFER, so that they will point to the same place on the buffer in the GPU. This will change the state to what the pointer pointing to. (graphics state) ARRAY_BUFFER is pointing to.

b) copy data from RAM to the GPU buffer where ARRAY_BUFFER is pointing to. (this step will also allocate the buffer). If you want to upload the points in the middle of a render function, you just need to call that one function.

c) when you call this function, it will draw the object on the viewport. It will take data from ARRAY_BUFFER (vertices + color). To determine the way of mapping those points, it will use the glVertexAttribPointer. Finally, based on the three parameters, it will draw the geometric primitives.

d) It will basically set the viewport. It uses the (x, y, width, height) to specifies the affine transformation from normalized device coordinates (NDC) to the window coordinates.

e) After you linked the uniform variable in with the shader, you will need to assign value to the uniform variable. This function will assign the variable in RAM to the uniform variable in shader.