

Student Number:

Name:

Write down answers in-between questions. Please answer using short sentences.
The back of each page can be used for practice, but DO NOT write down the answer on the back.

Be sure to write your student number and name on each page.

1. (1 pt each) True / False questions:

- 1) Rasterization processes each pixel individually by shooting a ray through it. ()
- 2) In homogeneous coordinates, a 2D point $[x, y]$ becomes $[x, y, 0]$. ()
- 3) 2D linear transformations can only be represented with 3×3 matrices. ()
- 4) The "LookAt" method requires only the eye point and look-at point to define the camera frame. ()
- 5) In OpenGL, the final vertex position in screen space is computed using the MVP matrix, which combines modeling, viewing, projection, and viewport transformations. ()
- 6) Rendering a hierarchical model requires computing the local transformation of each node only. ()

2. (3 pts) Let the following transformations be defined in 2D:

T: translation by $(1, 0)$

R: rotation by 90° counterclockwise

Consider the composite transformation: $p' = \mathbf{TR}p$

Compute the final position p' for $p = (1, 0)$.

3. (3 pts) Choose **ALL** correct statements about double buffering.

- 1) It reduces flickering
- 2) It uses two frame buffers
- 3) It increases rendering errors
- 4) It swaps front and back buffers

4. (5 pts) There are two affine transformations **T** and **S** on n -dimensional spaces. Their linear parts are represented by $n \times n$ matrix $\mathbf{M}_t$ and $\mathbf{M}_s$, and their translational parts are represented by $n \times 1$ column vectors $\mathbf{t}_t$ and $\mathbf{t}_s$, respectively. Let's say you have a n -dimensional point $\mathbf{p}$. What is the new point $\mathbf{p}'$ where the point $\mathbf{p}$ is transformed by the transformation **T** **compose** **S**? Fill in the blanks (a), (b), and (c).

$$\mathbf{p}' = \underline{\text{(a)}} \mathbf{p} + \underline{\text{(b)}} + \underline{\text{(c)}}$$

Student Number:

Name:

5. (4 pts) Consider the following OpenGL call::

```
glDrawArrays(GL_TRIANGLES, 0, 6)
```

Each vertex consists of 3 floats representing (x, y, z).

Which of the following vertex arrays correctly produces a rectangle using two triangles without overlapping or crossing triangles? Choose **ALL** correct answers.

- | | |
|--|--|
| <p>1) <pre>vertices =
glm.array(glm.float32,
 -0.5, -0.5, 0.0,
 0.5, -0.5, 0.0,
 0.5, 0.5, 0.0,

 -0.5, -0.5, 0.0,
 0.5, 0.5, 0.0,
 -0.5, 0.5, 0.0
)</pre></p> | <p>2) <pre>vertices =
glm.array(glm.float32,
 -0.5, -0.5, 0.0,
 0.5, -0.5, 0.0,
 0.5, 0.5, 0.0,

 0.5, -0.5, 0.0,
 0.5, 0.5, 0.0,
 -0.5, 0.5, 0.0
)</pre></p> |
| <p>3) <pre>vertices =
glm.array(glm.float32,
 -0.5, -0.5, 0.0,
 0.5, -0.5, 0.0,
 -0.5, 0.5, 0.0,

 0.5, -0.5, 0.0,
 0.5, 0.5, 0.0,
 -0.5, 0.5, 0.0
)</pre></p> | <p>4) <pre>vertices =
glm.array(glm.float32,
 -0.5, -0.5, 0.0,
 0.5, -0.5, 0.0,
 0.5, 0.5, 0.0,

 0.5, 0.5, 0.0,
 -0.5, 0.5, 0.0,
 -0.5, -0.5, 0.0
)</pre></p> |

6. (3 pts) Which of the following correctly describes how color data is passed from the vertex shader to the fragment shader?

- 1) Using uniform variables
- 2) Using vertex attributes
- 3) Using out (vertex shader) and in (fragment shader) variables
- 4) Using gl_Position

7. (4 pts) Consider the following vertex data:

```
vertices = glm.array(glm.float32,  
    # position      # color  
    -1.0, -1.0, 0.0, 1.0, 0.0, 0.0,  
    1.0, -1.0, 0.0, 0.0, 1.0, 0.0,  
    0.0, 1.0, 0.0, 0.0, 0.0, 1.0,  
)
```

Student Number:

Name:

Which configurations correctly set the **color attribute (location = 1)**?

- 1) glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*glm.sizeof(glm.float32), ctypes.c_void_p(3 * glm.sizeof(glm.float32)))
- 2) glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3* glm.sizeof(glm.float32), None)
- 3) glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6* glm.sizeof(glm.float32), None)
- 4) glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*glm.sizeof(glm.float32), ctypes.c_void_p(6 * glm.sizeof(glm.float32)))

8. (4 pts) Consider the following code.

Application code:

```
uMix_loc = glGetUniformLocation(shaderProgram, "uMix")

glUseProgram(shaderProgram)
glUniform1f(uMix_loc, 0.5)
```

Vertex shader:

```
#version 330 core

layout (location = 0) in vec3 vin_pos;
layout (location = 1) in vec3 vin_color;

uniform float uMix;

out vec3 vColor;

void main()
{
    gl_Position = vec4(vin_pos, 1.0);
    vColor = (1.0 - uMix) * vin_color + uMix * vec3(1.0, 1.0, 1.0);
}
```

If `vin_color = (0.2, 0.4, 0.8)`, what is the value of `vColor`? Fill in the blanks.

`vColor = ( _____ , _____ , _____ )`

9. (3 pts) Which of the following best explains why adding two points is not defined in an affine space?

- 1) It produces a vector instead of a point
- 2) The result depends on the choice of coordinate system
- 3) It cannot be computed numerically
- 4) It always results in zero

Student Number:

Name:

10. (4 pts) An affine transformation matrix M is given. Which statements are valid interpretations of M ? Choose **ALL** correct answers.

- 1) It transforms the geometry expressed in the world frame.
- 2) It defines a body frame with respect to the world frame.
- 3) It converts the coordinates of a point from the body frame to the world frame.
- 4) It always converts world-frame coordinates to body-frame coordinates.
- 5) It can be interpreted differently depending on the coordinate frame of the input vector.

11. (4 pts) The following GLSL code snippet is from a vertex shader that performs a 2D affine transformation M using homogeneous coordinates. Fill in the blanks so that the shader works correctly.

```
#version 330 core

layout (location = 0) in vec3 vin_pos;
layout (location = 1) in vec3 vin_color;

out vec4 vout_color;

uniform mat3 M;

void main()
{
    vec3 p2D_in_hcoord = vec3(vin_pos.xy, (a));
    vec3 p2D_new = (b) * p2D_in_hcoord;
    gl_Position.xy = (c).xy;
    gl_Position.w = (d);
    vout_color = vec4(vin_color, 1.0);
}
```

12. (6 pts) Consider a rigid transformation:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} (a) & (b) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (c) & (d) \\ 0 & 1 \end{bmatrix}$$

Using this decomposition, the inverse is:

$$T^{-1} = \left(\begin{bmatrix} (a) & (b) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (c) & (d) \\ 0 & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} (c) & (d) \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} (a) & (b) \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} (e) & (f) \\ 0 & 1 \end{bmatrix}$$

where R is a 3×3 rotation matrix and t is a 3×1 translation vector.

Fill in the blanks:

(a) = _____ (b) = _____ (c) = _____ (d) = _____
 (e) = _____ (f) = _____

Student Number:

Name:

13. (3 pts) The vertex "A" is one of the vertices of an object. A's position w.r.t. the object's body frame

[2, 0, 0, 1]
[0, 1, 0, 2]
[0, 0, 1, 3]
[0, 0, 0, 1]

($A^{\{b\}}$) is (2,-1,4) and modeling transformation of the object is: . What is the position of the vertex "A" w.r.t. world frame ($A^{\{w\}}$)?

14. (3 pts) To simulate moving the camera by (1,0,0), you can instead use:

```
# modeling matrix
M = glm.translate(glm.vec3(   (a)  ,   (b)  ,   (c)   ) )
```

Fill in the blanks (a), (b), and (c).

15. (4 pts) The transformation from the viewing frustum to a cuboid with near and far planes equal to

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & fn \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

those of the frustum is P_{fc} , where n and f is the distances to near and far planes from the camera, respectively (that is, $n>0$ and $f>0$). If a point $\mathbf{p}$ is on the **far plane** of the viewing frustum and its x, y coordinates are (a, b) w.r.t. camera frame, what is the 3D position of $\mathbf{p}$ transformed by P_{fc} ? Write down the position **after applying perspective division**. Assume that the camera is always looking in the $-z$ direction in the camera space.

16. (6 pts) We want to map:

$$(p_x, p_y) \text{ in } (x_l, y_l) \times (x_h, y_h) \rightarrow (p'_x, p'_y) \text{ in } (-1, -1) \times (1, 1),$$

where (x_l, y_l) and (x_h, y_h) define the input rectangle, and $(-1, -1)$ and $(1, 1)$ define the output rectangle.

The transformation is composed of three steps:

1) Translate by ((a) , (b))

2) Scale by ((c) , (d))

3) Translate by ((e) , (f))

Fill in the blanks (a)–(f).

Student Number:

Name:

17. (5 pts) The function `draw_cube_array()` renders a $5 \times 5 \times 5$ array of cubes using a shared VAO and MVP matrix. Refer to the following function definition:

```
def draw_cube_array(vao, MVP, loc_MVP):
    glBindVertexArray(vao)
    for i in range(5):
        for j in range(5):
            for k in range(5):
                model = glm.translate(glm.vec3(i, j, k)) *
glm.scale(glm.vec3(0.5, 0.5, 0.5))
                MVP_cube = MVP * model
                glUniformMatrix4fv(loc_MVP, 1, GL_FALSE,
glm.value_ptr(MVP_cube))
                glDrawArrays(GL_TRIANGLES, 0, 36)
```

Suppose the line defining `model` is changed to:

```
                model = glm.scale(glm.vec3(0.5, 0.5, 0.5)) *
glm.translate(glm.vec3(i, j, k))
```

How will this change affect the rendered result? Select **ALL** correct statements below:

- 1) Both versions produce the same visual output.
 - 2) In the modified version, the spacing between cubes is reduced and they appear closer to the origin.
 - 3) In the modified version, the cube positions are scaled along with the cubes themselves.
 - 4) The modified version makes all cubes invisible due to incorrect transformation.
 - 5) The original version scales the cube's shape first, then moves it to the grid position, with respect to the global coordinates.
18. (5 pts) Consider the following callback function::

```
def framebuffer_size_callback(window, width, height):
    global g_P
    glViewport(0, 0, width, height)

    ortho_width =     (a)    
    ortho_height =     (b)    

    g_P = glm.ortho(-ortho_width*.5, ortho_width*.5,
                    -ortho_height*.5, ortho_height*.5,
                    -10, 10)
```

The code should satisfy the following:

- Changing the window **width** changes the object scale
- Changing the window **height** does NOT change the object scale
- The object must maintain its original aspect ratio (width/height), i.e., no stretching or squeezing, regardless of changes in window size.

Fill in the blank (a) and (b).

Student Number:

Name:

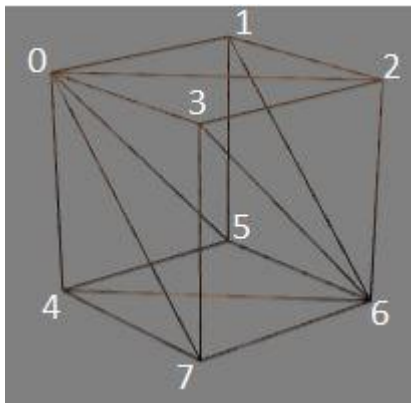
19. (3 pts) In hierarchical modeling, what does the **global transform** of a node represent?

- 1) The transformation from the root node's frame to the world frame
- 2) The transformation from the node's frame to its parent's frame
- 3) The accumulated transformation from the root to the node
- 4) The shape scaling and orientation of the node itself
- 5) The camera-space position of the node

20. (4 pts) The figure below shows the vertex indices of a cube represented as a triangle mesh. The following code creates an index array to render the cube using the `glDrawElements()` function.

Fill in the blanks (a) and (b) to correctly complete the index array.

Use counterclockwise (CCW) vertex ordering, as viewed from outside the cube, to indicate front faces.



```
indices = glm.array(glm.uint32,
    0,2,1,
    0,3,2,
    4,5,6,
    4,6,7,
    0,1,5,
    ____ (a) ____,
    3,6,2,
    3,7,6,
    1,2,6,
    1,6,5,
    0,7,3,
    ____ (b) ____,
    1)
```

21. (4 pts) Which of the following correctly describe an orbit camera (Blender-style)? (Choose **ALL** correct)

- 1) The camera rotates around a fixed target point (look-at point) , while always looking at it
- 2) The camera moves freely without a reference point
- 3) Zoom changes the distance to the target (look-at point)
- 4) The camera always looks in the same direction
- 5) Panning moves both camera (eye point) and target (look-at point) together.

Student Number:

Name:

22. (4 pts) Which statements correctly compare a free camera and an orbit camera? (Choose **ALL** correct)

- 1) Free camera is suitable for FPS-style navigation
- 2) Orbit camera is better for inspecting objects
- 3) Free camera always rotates around a fixed point
- 4) Orbit camera maintains a target point
- 5) Both cameras behave identically

23. (6 pts) The following Node class represents a node in a hierarchical model and builds a tree structure. Fill in the blank (a) and (b) to complete `update_tree_global_transform()` that calculates and updates the global transformation of a node and its all descendants.

```
class Node:
    def __init__(self, parent, shape_transform, color):
        self.parent = parent
        self.children = []
        if parent is not None:
            parent.children.append(self)

        self.transform = glm.mat4() # local transform
        self.global_transform = glm.mat4()

        self.shape_transform = shape_transform
        self.color = color

    def set_transform(self, transform):
        self.transform = transform

    def update_tree_global_transform(self):
        if self.parent is not None:
            self.global_transform = __ (a) __
        else:
            self.global_transform = self.transform

        for child in self.children:
            __ (b) __
```

Student Number:

Name:

24. (4 pts) Consider the following two code snippets:

```
# version A
node.shape_transform = glm.scale((2,1,1))

# version B
node.set_transform(glm.scale((2,1,1)))
```

Assume node is an instance of the Node class above, with the blanks correctly completed.

How do these two versions differ? (Choose ALL correct)

- 1) Version A changes only the shape of the node
- 2) Version B changes the position of the node in the hierarchy
- 3) Version A affects child nodes
- 4) Version B affects child nodes
- 5) Both produce identical results