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) Rendering is the process of creating a 3D model. ( )
   2) Double buffering helps to reduce flickering and increase frame rates. ( )
   3) You must always compile and link shaders every frame before rendering. ( )
   4) In homogeneous coordinates, a 2D point [x, y] becomes [x, y, 0]. ( )
   5) Rigid transformations preserve both distances and angles. ( )
   6) Vertex attributes can only include position data. ( )
   7) Uniform variables automatically reset to zero every rendering frame. ( )
   8) The order of applying affine transformation matrices does not matter. ( )
   9) The transformation matrix M can be interpreted both as changing geometry and as changing the frame in which geometry is described. ( )
   10) NumPy uses a column-major storage convention to store matrices. ( )
   11) In the rasterization pipeline, vertex processing happens after fragment processing. ( )
   12) The default OpenGL camera looks along the positive z-axis and has its up direction set to the x-axis. ( )
   13) The modeling transformation matrix M is calculated once per frame for all objects and cannot be changed during rendering. ( )
   14) In the viewport transformation, the z-value of a point is always discarded. ( )
   15) In OpenGL, the final vertex position in screen space is computed using the MVP matrix, which combines modeling, viewing, projection, and viewport transformations. ( )
   16) In hierarchical modeling, each part's movement is described with respect to the global (world) coordinate system. ( )
   17) In modern OpenGL, the built-in matrix stack is commonly used and preferred to render hierarchical models. ( )

Student Number: **Name:**

2. (5 pts) Which of the following statements about Rasterization and Ray Tracing are correct? Select **ALL** that apply.

   1) Rasterization determines visibility by casting rays through each pixel.

   2) Ray tracing handles shadows, reflections, and transparency using the same basic mechanism: intersecting rays with scene geometry.

   3) Rasterization is typically used for real-time rendering and benefits from parallel processing.

   4) Ray Tracing renders primitives directly to the screen, one by one.

   5) Recently, Ray Tracing has become possible for real-time rendering.

3. (5 pts) To render a simple white triangle using modern OpenGL (version 3.3 Core Profile), a series of initialization steps must be performed before the glDrawArrays() function is called in the rendering loop. Which of the following steps are required to correctly render the triangle? Select **ALL** that apply.

   1) Create and bind a Vertex Array Object (VAO)

   2) Create and bind a Vertex Buffer Object (VBO)

   3) Configure vertex attributes using glVertexAttribPointer() and glEnableVertexAttribArray()

   4) Load and activate the shader program using glUseProgram()

   5) Call glfwPollEvents() to process input and window events

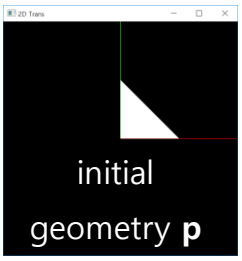
4. (4 pts) What is the result of the composite transformation **TFR** (**p' = TFRp**)? Note that the 2D spaces in the given figures range from -1 to 1 both for x and y directions.

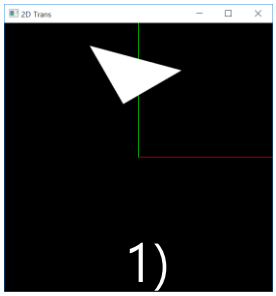
θ = 60 degrees

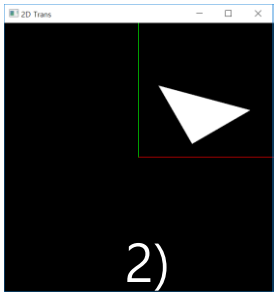
$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

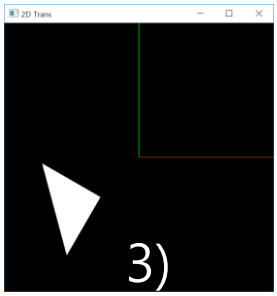
$$F = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

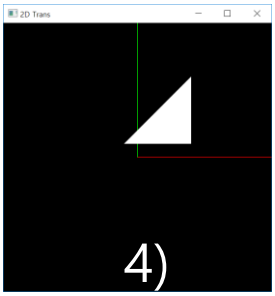
$$T = \begin{bmatrix} 1 & 0 & 0.4 \\ 0 & 1 & 0.1 \\ 0 & 0 & 1.0 \end{bmatrix}$$











Student Number: **Name:**

5. (5 pts) Which of the following properties are preserved by all affine transformations, but not by projective transformations? Select **ALL** that apply.

   1) Straight lines remain straight

   2) Parallel lines remain parallel

   3) Angles between lines are preserved

   4) Ratios of distances along a line are preserved

   5) The origin is preserved

6. (5 pts) Which of the following statements correctly describe how data is passed from an OpenGL application to a fragment shader? Select **ALL** that apply.

   1) layout(location = 0) in a vertex shader specifies a vertex attribute index for data from the application.

   2) Variables marked with uniform can be updated from the application each rendering frame.

   3) The out variable in the vertex shader is directly accessed by the application.

   4) In order for the fragment shader to receive data from the vertex shader, the variable must have the same name and type.

   5) A uniform variable must be declared as ‘in’ in the fragment shader to be accessed.

7. (4 pts) Which of the following statements about an affine transformation matrix is INCORRECT?

   1) An affine transformation matrix can transform a geometry with respect to the world frame.

   2) An affine transformation matrix can define a new affine frame with respect to the world frame.

   3) Multiplying an affine transformation matrix to a point converts the point's representation from a body frame to the world frame.

   4) An affine transformation matrix transforms both the direction and the location of a vector.

   5) An affine transformation matrix may include rotation, translation, and scaling.

8. (4 pts) You are given two affine transformation matrices:
   R: a rotation matrix, T: a translation matrix.
   You apply these transformations to a point p as follows: $p'' = T \cdot R \cdot p$. Which of the following correctly describes the order of transformation applied to p, with respect to the world frame?

   1) Apply translation first, then rotation.

   2) Apply rotation first, then translation.

   3) Apply both transformations simultaneously.

   4) First rotate the world frame, then translate the body frame.

   5) It depends on whether the point is a vector or a point.

Student Number: **Name:**

9. (6 pts) The following GLSL code snippet is from a vertex shader that performs a 2D affine transformation 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)__ * __(c)__;
    gl_Position = vec4(p2D_new.x, p2D_new.y, 0.0, __(d)__);
    vout_color = vec4(vin_color, 1.0);
}
```

10. (6 pts) Fill in the blanks to compute the camera frame basis vectors using the LookAt method.
Given:
Eye point: **e**
Look-at point (center point): **c**
Up vector: **t**
Note:
normalize(v) returns a unit vector in the direction of vector v.
cross(a, b) returns the cross product of vectors a and b

```
w = normalize(__(a)__)
u = normalize(cross(__(b)__, w))
v = cross(w, __(c)__)
```

11. (5 pts) A rigid transformation in 3D is represented by the following 4×4 matrix:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

,where R is a 3×3 rotation matrix and t is a 3×1 translation vector.
Write down the inverse T^{-1} of this rigid transformation matrix.

Student Number: **Name:**

12. (3 pts) Fill in the blanks in the Python code that sets up the MVP matrix for vertex transformation.

```
P = glm.ortho(-1, 1, -1, 1, -1, 1)
V = glm.lookAt(glm.vec3(0, 0, 1), glm.vec3(0, 0, 0),
glm.vec3(0, 1, 0))
M = glm.mat4(1.0)

MVP = __(a)__ * __(b)__ * __(c)__
```

13. (5 pts) Select **ALL** statements that are true about clip space and normalized device coordinates (NDC) space.

1) Clip space is a 4D coordinate space, while NDC space is a 3D space obtained after perspective division.

2) The w-component in clip space determines how much perspective distortion is applied during the projection.

3) The projection matrix transforms coordinates from view space to clip space.

4) The transformation from clip space to NDC space is non-linear with respect to depth.

5) Perspective division divides the x, y, z components of clip space by w to produce NDC coordinates.

6) The vertex shader must output gl_Position in NDC space.

14. (5 pts) The following callback function is called whenever the window is resized. Here, g_P is a global variable used as the projection matrix when constructing the MVP matrix inside the rendering loop.

```
def framebuffer_size_callback(window, width, height):
    global g_P
    glViewport(0, 0, width, height)
    ortho_height = 10.
    ortho_width = ortho_height * width / height
    g_P = glm.ortho(-ortho_width*.5, ortho_width*.5,
                    -ortho_height*.5, ortho_height*.5, -10, 10)
```

Select **ALL** correct statements about the visual result of this implementation:

1) The visible region of the scene is updated to match the new window size.

2) The projection matrix is updated dynamically to preserve the object's aspect ratio.

3) This implementation ensures that the aspect ratio of the rendered objects (i.e., their width-to-height ratio) remains consistent even when the window is resized.

4) When the window's height increases, the object appears smaller.

5) When the window's width increases, the object's size remains the same.

6) The viewport remains fixed in size regardless of window resizing.

Student Number: **Name:**

15. (5 pts) The function draw_cube_array() renders a 5×5×5 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.

16. (6 pts) The following simplified definition shows part of the Node class used in a hierarchical modeling example:

```
class Node:
    def __init__(self, transform, shape_transform, children=[]):
        self.transform = transform
        self.shape_transform = shape_transform
        self.children = children
        self.parent = None

    def set_parent(self, parent):
        self.parent = parent

    def get_global_transform(self):
        if self.parent:
            return self.parent.get_global_transform() * self.transform
        else:
            return self.transform

    def get_shape_transform(self):
        return self.shape_transform
```

Below is a simplified version of the draw_node() function used to render each node:

Student Number: **Name:**

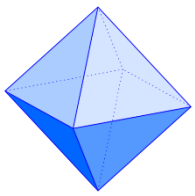
```
def draw_node(vao, node, VP, loc_MVP, loc_color):
    MVP = __(a)__ * node.__(b)__() * node.__(c)__()
    glBindVertexArray(vao)
    glUniformMatrix4fv(loc_MVP, 1, GL_FALSE, glm.value_ptr(MVP)
    ...
```

Fill in the blanks (a), (b), (c) to compute the correct MVP matrix for the given node.

17. (6 pts) Write the appropriate number in the blank (a), (b), and (c):

    When drawing a 3D octahedron with the **separate triangles scheme**, **__(a)__** scalar numbers are included in the **vertex array**.

    When drawing a 3D octahedron with the **indexed triangles scheme**, **__(b)__** scalar numbers are included in the **vertex array** and **__(c)__** scalar numbers are included in the **index array**.



18. (4 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