NATIONAL UNIVERSITY OF SINGAPORE

CS3241 — COMPUTER GRAPHICS

(Semester 1 AY2019/2020)

Time Allowed: **2 Hours**

ANSWERS

INSTRUCTIONS TO STUDENTS

1. This assessment paper contains **SEVEN (7)** questions and comprises **ELEVEN (11)** printed pages, including this page.
2. This is an **OPEN BOOK** assessment.
3. Answer **ALL** questions **within the space provided** in this booklet.
4. The full score of this paper is **75 marks**.
5. Write legibly with a **pen or pencil**. **Untidiness will be penalized**.
6. Do not tear off any pages from this booklet.
7. Write your **Student Number** below **using a pen**. Do not write your name.

(write legibly with a pen)

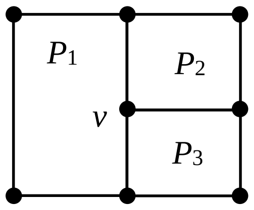
Student Number:	A								

This portion is for examiner's use only

Q#	1	2	3	4	5	6	7	Σ
MAX	8	15	9	10	10	13	10	75
SC								

QUESTION 1 [8 MARKS]

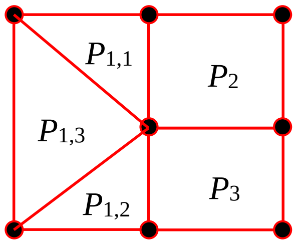
(1A) [4 marks] The diagram on the right shows 3 **quadrilaterals** (P_1, P_2 and P_3) from a mesh that approximates a **curved** surface. The vertex ***v*** **is shared only by P_2 and P_3**. Describe **two problems** that they may cause when they are being rendered with OpenGL. Assume OpenGL lighting computation and Gouraud shading are enabled.



(1) A crack may appear between P_1 and (P_2 and P_3).

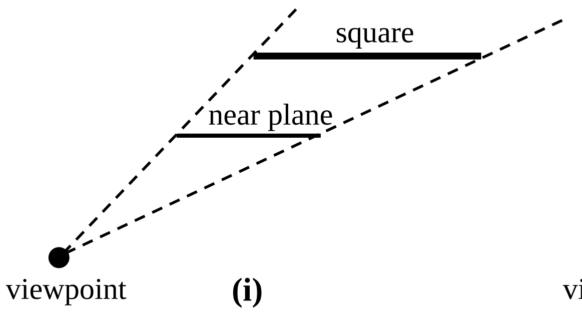
(2) The interpolated color in P_1 may look inconsistent from that of P_2 and P_3 along its edge adjacent with P_2 and P_3.

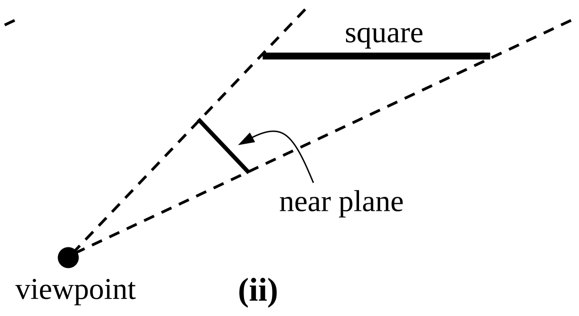
(1B) [4 marks] How would you **subdivide** quadrilateral P_1 to avoid the problems in Part A? Clearly **draw** your new polygon mesh in the space below. You must guarantee that each newly created polygon is **planar**. Furthermore, we assume that P_1 is sharing its left edge, bottom edge and top edge with some adjacent polygons, and we do not want those polygons to be modified.

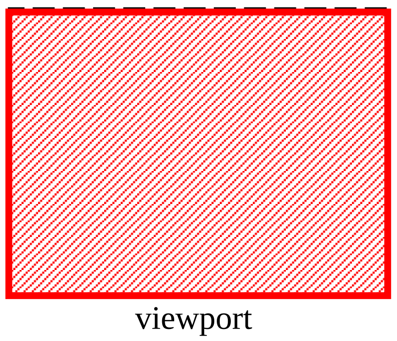


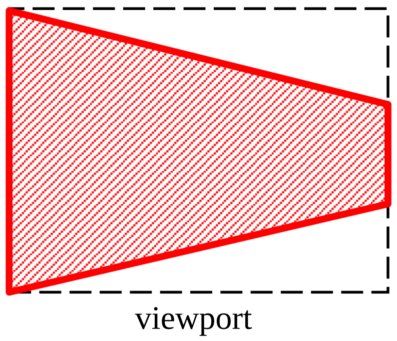
QUESTION 2 [15 MARKS]

(2A) [4 marks] Each of the following two diagrams shows a top view of a **view frustum** enclosing a **square** (only an edge of the square is seen from top view). The **near planes** in the two setups are positioned differently. The far planes are not shown. Given that the square is to be rendered to a **rectangular viewport**, **draw** and show how the square would appear in the viewport for each of the setups.









(2B) [6 marks] We want to capture an **environment cubemap** of a virtual scene from the **world** position (c_x, c_y, c_z). Write an OpenGL program fragment to set up the view to render the scene for the **+*x* side of the cubemap** (the side that faces in the +*x* direction of the world space). Use `gluLookAt()` and `glFrustum()`, and remember to set the correct OpenGL **matrix mode**. Assume the near and far plane distances are `nearDist` and `farDist` respectively, and the up-vector for the `gluLookAt()` function is (0, 1, 0).

Function usage:

```
gluLookAt(eyeX, eyeY, eyeZ, lookAtX, lookAtY, lookAtZ, upX, upY, upZ);
glFrustum(left, right, bottom, top, near, far);
```

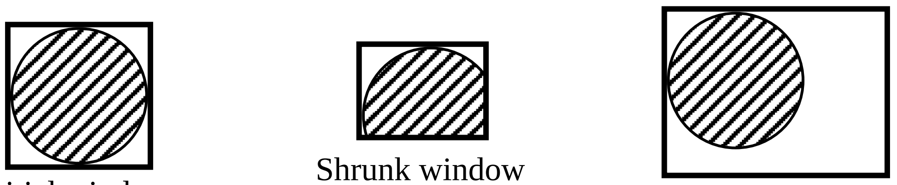
```
double cx, cy, cz;
double nearDist, farDist;  // both are positive values
...
glViewport( 0, 0, 512, 512 );
```

(more writing space next page)

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glFrustum( -nearDist, nearDist, -nearDist, nearDist,
           nearDist, farDist );

glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
gluLookAt( cx, cy, cz,  cx + 1, cy, cz, 0, 1, 0 );
```

(2C) [5 marks] Suppose in the **camera coordinate frame**, there is a disc in the *z* = 0 plane, **centered** at (100, 200, 0), and has a **radius** of 10. Initially, the disc fits exactly in a window of size 300 × 300. When the window is resized, you want the disc to appear the same size as before, even though it may be clipped; or if the window is too large, the disc always appears in the top-left corner (see diagram below). Complete the **reshape callback** function to set up the required **orthographic projection**. The **viewport** should always be the same size as the whole window. Function usage: `gluOrtho2D(left, right, bottom, top);`



```
void MyReshape( int w, int h ) {

    glViewport( 0, 0, w, h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();

    gluOrtho2D( 100 - 10,

                100 - 10 + 20 * (float)w/300,

                200 + 10 - 20 * (float)h/300,

                200 + 10                                     );

    glMatrixMode( GL_MODELVIEW );
}
```

QUESTION 3 [9 MARKS]

(3A) [4 marks] There are five triangles parallel to the $z = 0$ plane in the **camera** coordinate frame. Suppose the 5 triangles have the following colors and z coordinates:

Triangle	A	B	C	D	E
Color	Red	Green	Blue	Pink	White
z	–2	–3	–5	–7	–8

The 5 triangles are projected using the following OpenGL **orthographic** projection:

```
glOrtho( -100, 100, -100, 100, 0, 10 );
```

Assuming that all five triangles are inside the viewing volume, write in the following table the **z-buffer value** (z_{buf}) for the fragments of each triangle after it has been rasterized. The z-buffer value ranges from 0 to 1.

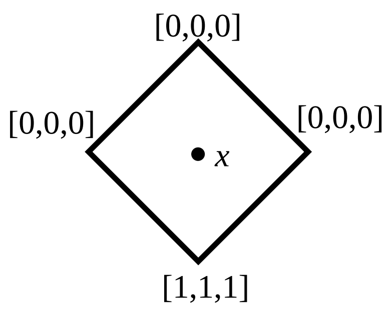
Triangle	A	B	C	D	E
z_{buf}	0.2	0.3	0.5	0.7	0.8

(3B) [5 marks] Suppose all the 5 triangles in Part A cover the pixel location (x_{pix}, y_{pix}). Given that the triangles are drawn in the order D, B, C, A and E, write in the following table the z-buffer value and the pixel color at location (x_{pix}, y_{pix}) in the framebuffer after each step.

Event	z_{buf} **at** (x_{pix}, y_{pix})	**Color at** (x_{pix}, y_{pix})
1. After clearing buffers	1.0	Black
2. After drawing Triangle D	0.7	Pink
3. After drawing Triangle B	0.3	Green
4. After drawing Triangle C	0.3	Green
5. After drawing Triangle A	0.2	Red
6. After drawing Triangle E	0.2	Red

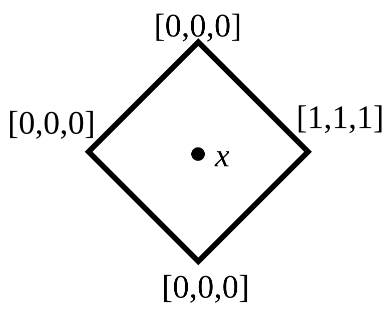
QUESTION 4 [10 MARKS]

(4A) [3 marks] A 3D square is projected on the screen as a 45°-rotated 2D square as shown in the diagram on the right. The four vertices have been assigned **RGB colors** as indicated in the diagram. Suppose the colors at the vertices are to be **bilinearly interpolated** over the interior of the square. Given that *x* is a pixel location at the center of the square, what would be the RGB color computed at *x*? Assume the rasterization is performed in **horizontal scan lines**.



RGB color at *x*: [0, 0, 0]

(4B) [3 marks] Suppose the square in Part A has been rotated 90° counterclockwise as shown in the diagram on the right. Given that everything is the same as in Part A, what would be the RGB color computed at *x*?



RGB color at *x*: [0.5, 0.5, 0.5]

(4C) [4 marks] Based on your results in Parts A and B, **(i)** what can you say about bilinear interpolation of vertex attributes over a quadrilateral? **(ii)** What undesirable effect can this cause? **(iii)** When will this effect be more observable?

(i) Bilinear interpolation of attributes over a quadrilateral is not rotational invariant.

(ii) This can cause undesirable effects because an interior point on the quadrilateral can be colored differently depending on the orientation of the quadrilateral.

(iii) This effect is made obvious when there is rotational animation of the quadrilateral.

QUESTION 5 [10 MARKS]

(5A) [2 marks] What is the use of **vertex normals**?

A vertex normal is used to specify the orientation of the surface at the vertex, and it is usually used in the lighting computation at the vertex.

(5B) [4 marks] Explain why **vertex normals** of triangles are **not suitable** for performing **back-face culling**.

(1) The 3 vertex normals of the triangle may be different from each other, so we may not know which one to use.
(2) It is possible that the angle between a vertex normal and the view vector is greater than 90 degrees, and yet the triangle's front face is still visible.

(5C) [4 marks] A triangle, whose three vertices have exactly the **same vertex normal**, is passed into OpenGL for rendering. The OpenGL **lighting computation** has been enabled and a **point light source** has been set up. The **material** at the three vertices are also the **same**. Are the computed colors at the three vertices always be the same? Explain why.

Not the same. Because the three vertices have different positions, therefore their vector L to the light source are different, and their vector V to the viewer are also different. These will produce different values of N.L and R.V in the Phong Illumination Equation.

QUESTION 6 [13 MARKS]

(6A) [4 marks] In Whitted Ray Tracing, suppose a **shadow ray** $\boldsymbol{P}(t) = \boldsymbol{O} + t\boldsymbol{D}$ (where $\boldsymbol{D}$ is a unit vector) is being shot from the surface point $\boldsymbol{O}$ towards the only light source at position $\boldsymbol{S}$. If there is an intersection between the ray and the scene at a value of $t > epsilon$, can we be sure that the surface point should be in shadow? **Why**? Assume all objects are **opaque**.

No. Because $t > epsilon$ includes intersections that are beyond/behind the light source.

(6B) [2 marks] When finding the intersection between a ray and a triangle using the **barycentric coordinates** approach, how can you tell that the ray is **parallel or almost parallel** to the plane of the triangle?

When the value of β or γ or t is huge.

(6C) [7 marks] Suppose there is an **opaque** sphere in an **enclosed** environment. All the other surfaces of the environment are **opaque** and have materials that have **diffuse and specular** components (k_d, k_r, and k_{rg} are greater than 0). However, the sphere's material has **only diffuse** component ($k_r = 0$ and $k_{rg} = 0$). There are **two point light sources** in the scene. Assuming we want to render a 100x100 pixels image of the scene using Whitted Ray Tracing, with **one level of recursion**, what would be the **total number of rays** that have to be shot? Assume the camera is within the environment but outside the sphere, and the **sphere occupies 3000 pixels** in the rendered image. **Show your workings clearly**.

If all objects are opaque, then there is no need to spawn refraction rays. If the scene is enclosed, then all primary rays and reflection rays will hit some object.

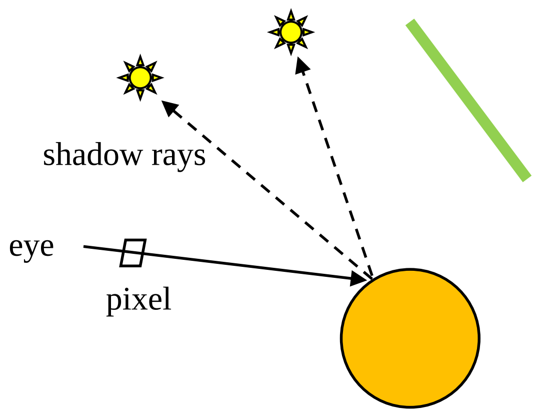
Rays shot per pixel on sphere = 1 primary ray + 2 shadow rays = 3

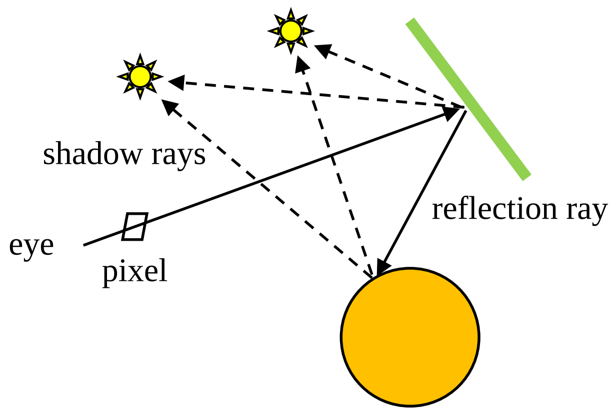
Total number of rays shot for pixels on sphere = 3000 * 3 = 9,000

Rays shot per pixel not on sphere = 1 primary ray + 1 reflected ray + 4 shadow rays = 6

Total number of rays shot for pixels not on sphere = (100*100 - 3000) * 6 = 42,000

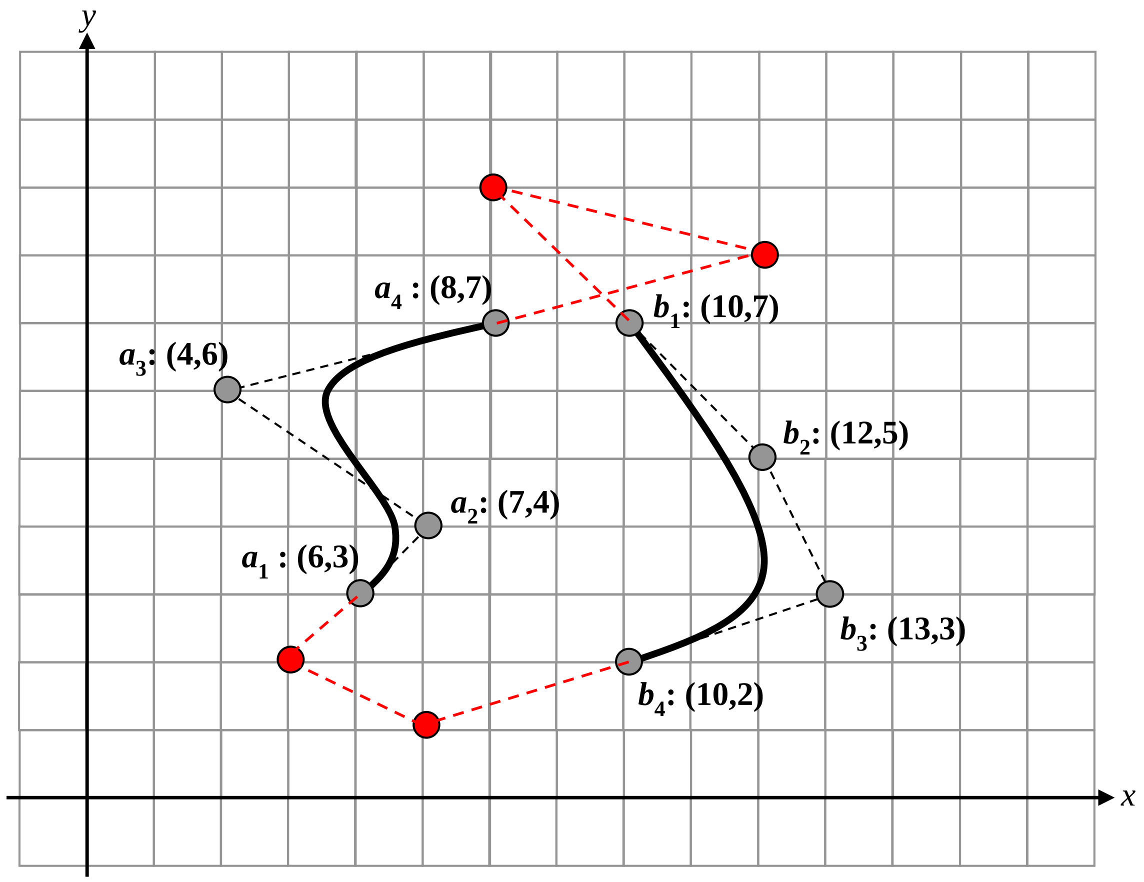
Total rays shot = 9,000 + 42,000 = 51,000





QUESTION 7 [10 MARKS]

(7) [10 Marks] The diagram below shows two **cubic Bezier curve segments** with control points $a_1\, a_2\, a_3\, a_4$ and $b_1\, b_2\, b_3\, b_4$. Create **two more** cubic Bezier curve segments to join the given curves in order to maintain ***C*¹** continuity. Namely, create two curve segments with control points $c_1\, c_2\, c_3\, c_4$ and $d_1\, d_2\, d_3\, d_4$ such that the first curve segment connects a_4 to b_1 and the second one connects b_4 to a_1. Draw your **new points** accurately in the diagram in the correct locations, together with the broken lines indicating the **control polygons**. State the **coordinates** of the new points below the diagram.



Coordinates of the new control points:

c_1: (8, 7)

c_2: (12, 8)

c_3: (8, 9)

c_4: (10, 7)

d_1: (10, 2)

d_2: (7, 1)

d_3: (5, 2)

d_4: (6, 3)

(blank page)

——— END OF PAPER ———