TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2016

TRIMESTER 2

CGRA 151

INTRODUCTION TO COMPUTER GRAPHICS

Time Allowed: TWO HOURS

CLOSED BOOK

Permitted materials: Silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.

Printed foreign language–English dictionaries are permitted.

No other material is permitted.

Instructions: Attempt ALL questions.

The examination will be marked out of 120 marks.

Brief *Processing* documentation is provided on page 21, which may be detached from the question paper.

Answer in the appropriate boxes if possible — if you write your answer elsewhere, make it clear where your answer can be found.

There are spare pages for your working and your answers in this question paper. However, you may ask for additional paper if you need it.

********** WITH SOLUTIONS **********

Questions

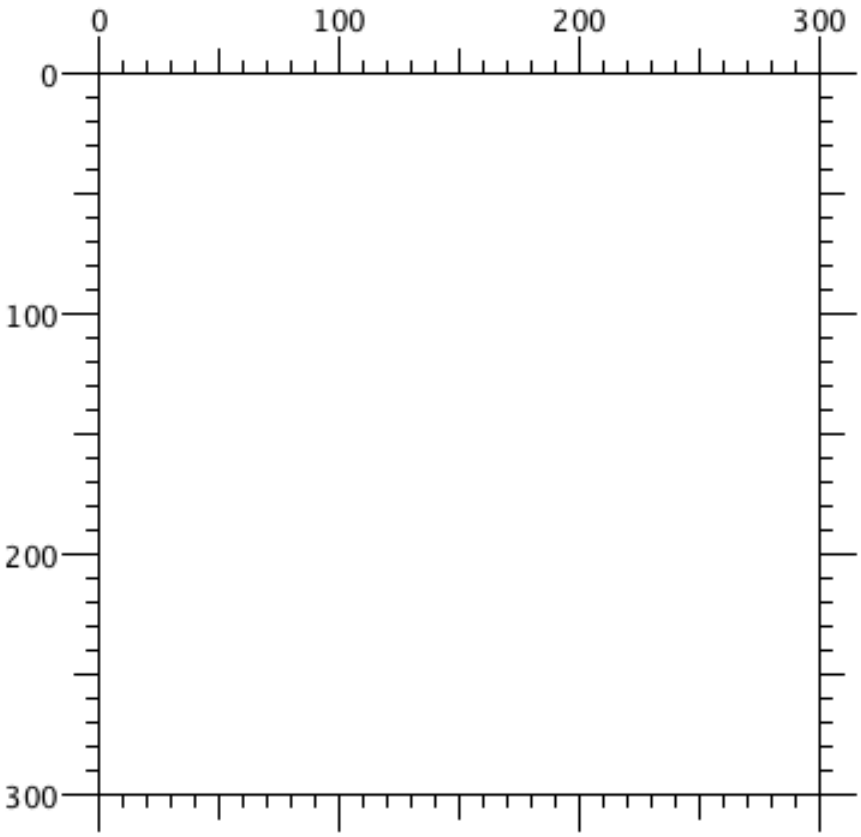
		Marks
1.	Programming in Processing	30
2.	Clipping Algorithms	30
3.	Colour, Display, and Human Vision	30
4.	Transformations in 2D and 3D	30
	Total	120

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Below is an extra copy of the template for Question 1(a), in case you make a mistake.



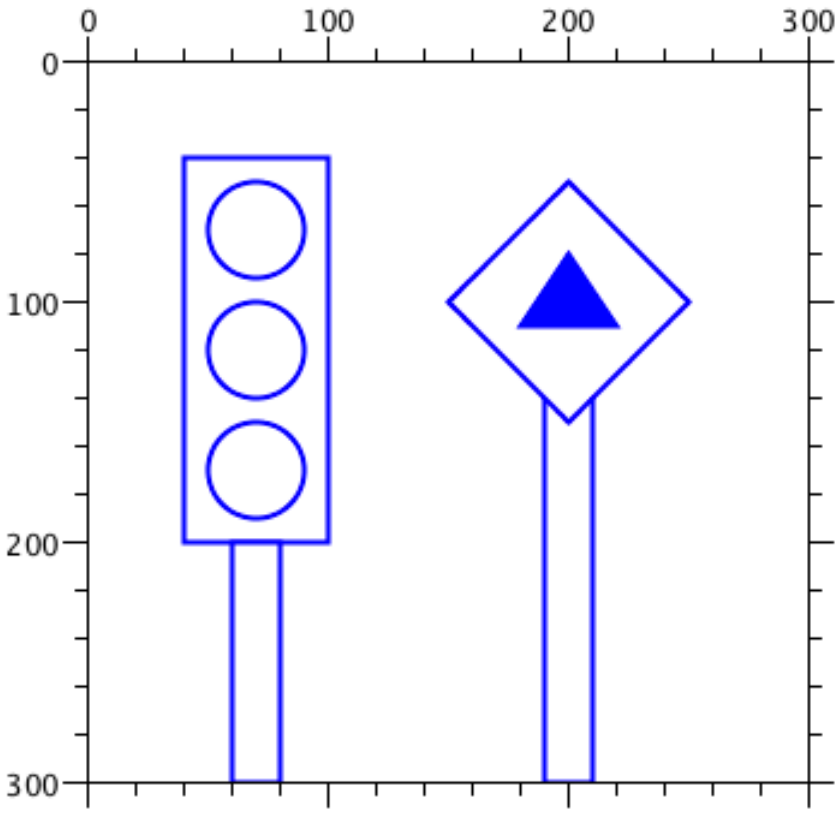
(spare copy of 300 × 300 Processing window)

Question 1. Programming in Processing **[30 marks]**

(a) **[10 marks]**

Sketch what the following code will draw in the Processing window.

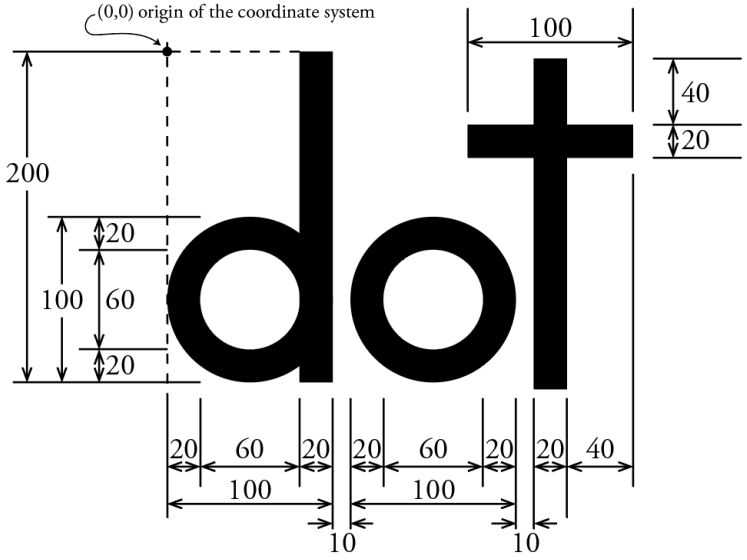
```
size( 300, 300 );
background( 255 );
stroke( 0 );
fill( 255 );
rect( 40,40, 60,160 );
rect( 60,200, 20,100 );
ellipse( 70, 70, 40,40 );
ellipse( 70,120, 40,40 );
ellipse( 70,170, 40,40 );
rect( 190,140, 20,160 );
beginShape();
  vertex( 200,150 );
  vertex( 150,100 );
  vertex( 200, 50 );
  vertex( 250,100 );
endShape( CLOSE );
fill( 0 );
triangle( 180,110, 220,110, 200,80 ) ;
```



(300 × 300 Processing window)

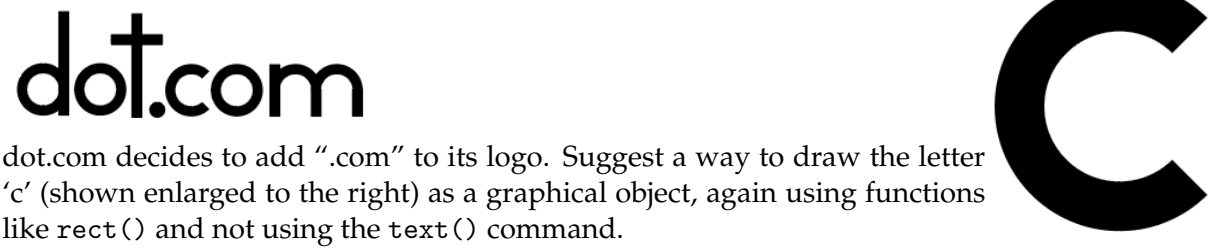
(b) **[15 marks]**

The company, dot.com, has designed a new logo, shown at right. It is black on a white background. Write a Processing method that will draw the logo as a graphical object (i.e., using functions like `rect()` and not using the `text()` command). The dimensions of the various components are shown below. The logo should have the origin, $(0,0)$, in the top left corner, as shown. The logo is of size 280×200. The white background is already drawn for you.




```
void drawDotLogo() {
  noStroke() ; // background rectangle has no border
  fill( 255 ) ;    // set colour to white for background rectangle
  rect( 0, 0, 280, 200 ) ; // draw the white background rectangle

  fill( 0 ) ; // black
  rect( 80, 0, 20, 200 ) ; // vertical bar of 'd'
  ellipse( 50, 150, 100, 100 ) ; // circle of 'd'
  ellipse( 160, 150, 100, 100 ) ; // circle of 'o'
  rect( 220, 0 , 20, 200 ) ; // vertical bar of 't'
  rect( 180, 40, 100, 20 ) ; // horizontal bar of 't'

  fill( 255 ) ; // white
  ellipse( 50, 150, 60, 60 ) ; // hole in 'd'
  ellipse( 160, 150, 60, 60 ) ; // hole in 'o'









}
```

(c) **[5 marks]**



dot.com decides to add ".com" to its logo. Suggest a way to draw the letter 'c' (shown enlarged to the right) as a graphical object, again using functions like `rect()` and not using the `text()` command.

```
// code, pseudo-code, or explanation of how to draw letter 'c'

  // method 1
  noStroke();
  fill( 0 ) ; // black
  ellipse( 50, 50, 100, 100 ) ; // circle of 'c'
  fill( 255 ) ; // white
  ellipse( 50, 50, 60, 60 ) ; // hole in 'c'
  triangle( 50, 50, 100, 100, 100, 0 ) ; // gap in right hand side of 'c'

  // method 2
  noStroke();
  fill(0);
  arc(50,50,100,100,PI/4.0,PI*7/4); // correct arc
  fill(255);
  ellipse(50,50,60,60); // take away the hole

  // method 3
  /* use a Bezier cubic curve with appropriate control points */

  // other methods are possible

// end of answer
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

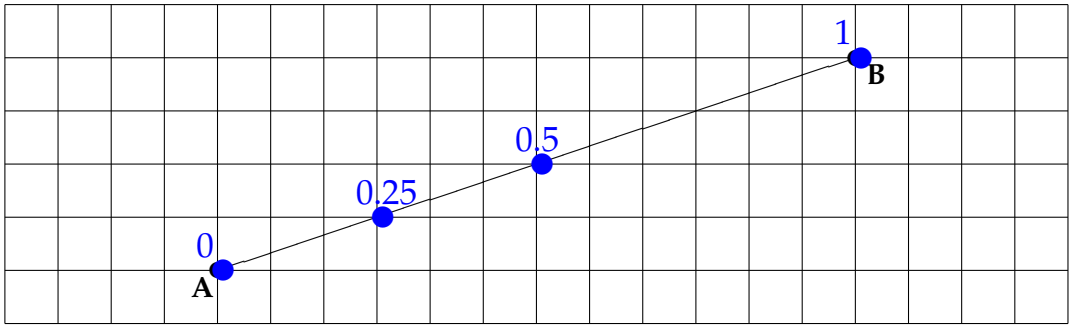
Question 2. Clipping Algorithms **[30 marks]**

This question is about clipping an arbitrary polygon against the edges of an axis-aligned box, such as a window on the screen.

(a) **[4 marks]** An edge of the polygon is specified by its end points $\mathbf{A} = (x_A, y_A)$ and $\mathbf{B} = (x_B, y_B)$, shown in the diagram below. The line between them can be represented parametrically as:

$$\mathbf{P}(t) = (1-t)\mathbf{A} + t\mathbf{B}.$$

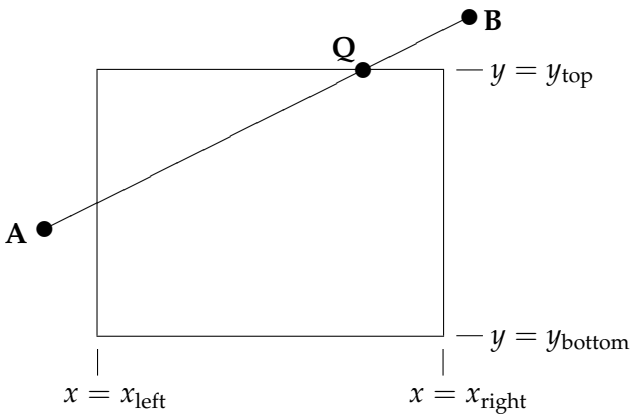
On the diagram below, show the points represented by $\mathbf{P}(0)$, $\mathbf{P}(0.25)$, $\mathbf{P}(0.5)$ and $\mathbf{P}(1)$. Label each of those four points with the value of the parameter t at that point.



(b) **[2 marks]** The edge between point **A** and point **B** is defined by the range $0 \leq t \leq 1$. Explain where points lie that have $t > 1$?

Points for which $t > 1$ lie on the infinite line defined by points **A** and **B**, on the far side of **B** from the line segment **AB**.

The axis-aligned clipping box is defined by four parameters: x_{left}, x_{right}, y_{top}, y_{bottom}.



(c) **[7 marks]** We want to find the intersection point $\mathbf{Q} = (x_Q, y_Q)$ where the polygon edge $\mathbf{P}(t)$ crosses the horizontal clipping edge at $y = y_{\text{top}}$. Give formulas for x_Q and y_Q that will allow you to calculate them from the known values (x_A, y_A, x_B, y_B, y_{top}).

$x_Q = x_A + \frac{y_{\text{top}} - y_A}{y_B - y_A}(x_B - x_A)$

$y_Q = y_{\text{top}}$

(d) **[7 marks]** Explain why and how we use bounding boxes to reduce the amount of computation required when clipping polygons.

Why

Bounding boxes are used so that we can calculate quickly whether the polygon lies completely outside the window or completely inside the window. If completely outside, we need not consider that polygon any further. If completely inside, we can draw the polygon with no need for further clipping. If the bounding box lies partially inside and partially outside, we need to call the polygon clipping algorithm before drawing. The polygon clipping algorithm is a lot more expensive to run than the simple bounding box calculation.

How

We compute the bounding box for the polygon and compare it against the bounding box for the window. Let the polygon have bounding box bbL, bbR, bbT, bbB and the window have bounding box wL, wR, wT, wB, where the initial letter represents Left, Right, Top and Bottom. The polygon is guaranteed to be outside the window if
$(bbL > wR) \vee (bbR < wL) \vee (bbT > wB) \vee (bbB < wT)$.
The polygon is guaranteed to be inside the window if
$(bbL > wL) \wedge (bbR < wR) \wedge (bbT > wT) \wedge (bbB < wB)$.

(e) **[10 marks]** We are given a polygon with N vertices, $\{(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_{N-1}, y_{N-1})\}$, and want to calculate its bounding box. Write a Processing method that takes, as input, the polygon vertices represented as two arrays (one array of x values and one array of y values) and that puts the bounding box of that polygon in the global variables `xL`, `xR`, `yT`, and `yB`.

```
float xL, xR ; // x-values of the left and right edges of the bounding box
float yT, yB ; // y-values of the top and bottom edges of the bounding box

void calculateBoundingBox( float[] x, float[] y, int N ) {
  // x is an array of the x-coordinates of the polygon's vertices
  // y is an array of the y-coordinates of the polygon's vertices
  // N is the number of vertices

  xL = x[0] ; xR = x[0] ;
  yT = y[0] ; yB = y[0] ;

  for( int i = 1 ; i < N ; i++ ) {
    if( x[i] < xL ) { xL = x[i] ; }
    if( x[i] > xR ) { xR = x[i] ; }
    if( y[i] < yT ) { yT = y[i] ; }
    if( y[i] > yB ) { yB = y[i] ; }
  }









}
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 3. Colour, Displays, and Human Vision **[30 marks]**

(a) **[6 marks]** Colour spaces

Pick **two** of the following three-dimensional colour spaces: *RGB*, *Lab*, *HSV*.

Explain what each of the three dimensions represents.

Your first choice of colour space is: *RGB*

[3 marks: 1 mark for each dimension]

First dimension: *R*: red component of colour

Second dimension: *G*: green component of colour

Third dimension: *B*: blue component of colour

Your second choice of colour space is: *Lab / HSV*

[3 marks: 1 mark for each dimension]

First dimension: *L*: Luminance (how bright it is) / *H*: Hue (the dominant colour)

Second dimension: *a*: red–green colour axis / *S*: Saturation (ranging from grey to fully saturated)

Third dimension: *b*: yellow–blue colour axis / *V*: Value (similar to luminance or brightness)

Printing in colour

(b) **[6 marks]** Explain why cyan, magenta and yellow are the three primary colours for printing.

Cyan, Magenta, Yellow (CMY) is a subtractive colour space. That is, each ink subtracts light from white. To do this effectively, each ink should subtract one third of the visible spectrum, and reflect the other two thirds. Cyan subtracts the red part of the spectrum. Yellow subtracts the blue part. Magenta subtracts the green part. Combining any two of the primaries allows us to remove two-thirds of the spectrum. Combining all three removes the entire spectrum, seen as black. In theory, we can reproduce any colour for humans because the human visual system has three types of colour receptor, sensitive to the long, medium and short wavelengths. These are roughly correlated with the red, green and blue parts of the spectrum. In practice, we cannot actually reproduce all colours because the correlation is only rough and because the inks are not perfect absorbers of light.

(c) **[2 marks]** Explain why we also use black ink.

Two reasons: (1) The combination of CMY does not produce black, but rather a muddy grey-brown. (2) Using black ink reduces the amount of the other three inks needed and reduces the overall amount of ink needed, thereby reducing the maximum amount of wetting that needs to be applied to the paper.

(d) **[10 marks]** Display resolution

DispGrey, a new display manufacturer, wishes to make a new tablet computer with a high resolution greyscale screen to be used for reading online books. Use your knowledge of the human retina to **calculate** the screen resolution, in pixels per inch, that will make the screen resolution match the resolving ability of the human eye.

You may find it useful to know that there is a maximum of 150,000 cones per square millimetre in the human retina and that there are exactly 25.4 millimetres in an inch.

State any and all assumptions that you make in your calculation (e.g., you will need to make an assumption about the distance from the eye to the display)

Assume that the screen is $b = 400$mm from the viewer's eye and that the eye is $d = 20$mm in diameter. Assume that the minimum spot size detectible by the human eye is captured by a cluster of 3–4 cones in the fovea. The width of a single cone cell, w, can be calculated from what we are told: $w = \frac{1}{\sqrt{150{,}000}} = 0.0026$mm. That cluster width, c, is double this, so $c \approx 0.005$mm. By similar triangles, the pixel size, a, is related to b in the same proportion as the minimum receptor size, c, is related to d. The pixel size is therefore $a = \frac{bc}{d} = \frac{400 \times 0.005}{20} = 0.1$mm. We get pixels per inch, p, by inverting this and multiplying by the number of millimetres per inch: $p = \frac{1}{0.1} \times 25.4 \approx 250$ppi.
Depending on the assumptions, this number can vary quite considerably.

Human vision

(e) **[3 marks]** What difference would a human notice if the screen had a resolution significantly **lower** than that calculated in part (d)? Justify your answer.

The human would notice the individual pixels on the screen because a resolution significantly below that calculated in part (d) has pixels sufficiently large that a human can distinguish individual pixels. The screen will have a blocky ('pixelated') look.

(f) **[3 marks]** What difference would a human notice if the screen had a resolution significantly **higher** than that calculated in part (d)? Justify your answer.

The human would notice no difference at all, because the display in part (d) is at a resolution where a human cannot perceive individual pixels, so increasing the resolution will have no visible effect.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 4. Transformations in 2D and 3D **[30 marks]**

(a) **[4 marks]** Why are homogeneous coordinates used in transformations in computer graphics?

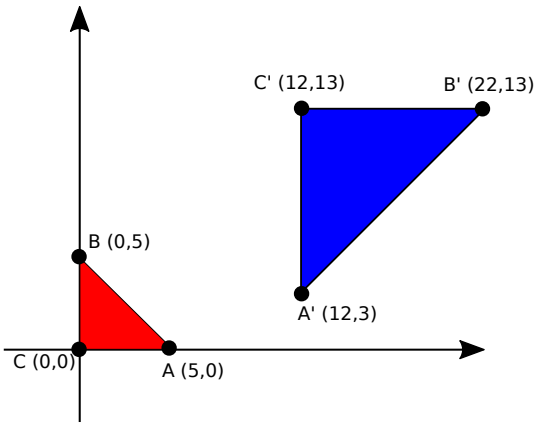
We use matrices to represent transformations in computer graphics [1 mark].
With homogeneous coordinates, we can represent transformations as matrices that are not possible to represent as matrices in non-homogeneous coordinates [3 marks].

(b) **[2 marks]** Give two examples of transformations that particularly benefit from the use of homogeneous coordinates.

translation [1 mark], *projection* [1 mark].

(c) **[8 marks]**

Describe the transformations required to transform object ABC to object $A'B'C'$. You may use any or all of *translate*, *rotate*, *shear* and *scale*, specifying them with the correct parameters, in the correct order.



scale(2, 2) *or* scale(2)
rotate(–PI/2) *or* rotate(−90°) (i.e., rotation of 90° clockwise)
translate(12, 13)

Since the scale is uniform, it does not matter if we scale or rotate first.
2 marks for the correct order of the correct transforms.
2 mark for the correct parameters in each of the three correct transforms.

(d) **[6 marks]** Give the 3×3 matrices for each of the following three 2D transformations: *scale*$(2,3)$ (scaling by a factor of 2 in the x-dimension and a factor of 3 in the y-dimension), *rotate*$(60°)$ (rotation of $60°$ anticlockwise), *translate*$(4,5)$ (translation by a distance of 4 in the x-dimension and a distance of 5 in the y-dimension). You may leave expressions like $\cos(60°)$ and $\sin(60°)$ in your answer

scale$(2,3)$: $\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

rotate$(60°)$: $\begin{bmatrix} \cos(60°) & -\sin(60°) & 0 \\ \sin(60°) & \cos(60°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

translate$(4,5)$: $\begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix}$

2 marks for each matrix.

(e) **[8 marks]** We want to project the 3D world to a 2D plane. We set the eye to be at the origin, $(0,0,0)$, and the plane to which we project is parallel to the xy-plane at distance $z = d$. A *projection* matrix to do this is defined as:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

We have a line in 3D with end points at $\mathbf{A} = (5,5,25)$ and $\mathbf{B} = (10,12,20)$.

Set $d = 10$ and, using the projection matrix, find the 2D coordinates of the projected points.

e.g., $\begin{bmatrix} 5 \\ 5 \\ 25 \end{bmatrix} \xrightarrow{\text{homog.}} \begin{bmatrix} 5 \\ 5 \\ 25 \\ 1 \end{bmatrix} \xrightarrow{\text{project}} \begin{bmatrix} 5 \\ 5 \\ 25 \\ 2.5 \end{bmatrix} \xrightarrow{\text{de-homog.}} \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix} \xrightarrow{\text{to 2D}} \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

$A' = (2,2)$
By similar method:
$B' = (5,6)$
4 marks for using the correct method
2 marks for correct answer for point **A**
2 marks for correct answer for point **B**

(f) **[2 marks]** What is the length of the projected line in 2D?

length = $\sqrt{(5-2)^2 + (6-2)^2} = 5$
2 marks for getting the length right
Will still get marks if calculate correct length from incorrect answer in previous box.

Brief Processing documentation

You may detach this page from the question paper.

```
void point( float x, float y )
// draw a point at location (x,y)

void line( float x1, float y1, float x2, float y2 )
// draw a line segment from location (x1,y1) to location (x2,y2)

void rect( float left, float top, float width, float height )
// draw a rectangle with its top left corner at the indicated position
// of size width times height

void ellipse( float centerX, float centerY, float width, float height )
// draw an ellipse centred at the indicated position
// of size width times height

void triangle( float x1, float y1, float x2, float y2, float x3, float y3 )
// draw a triangle with vertices at the three points
// (x1,y1), (x2,y2), (x3, y3)

void beginShape()
// start a shape which comprises a sequence of polygon vertices

void vertex( float x, float y )
// specify a vertex in the current shape

void endShape( mode )
// end and draw a shape. If the mode is "CLOSE" then the final point
// will be connected to the initial point. If the mode is missing then
// the shape will be left open.

void bezier( float x1, float y1, float x2, float y2,
             float x3, float y3, float x4, float y4 )
// draw a Bezier cubic curve with the four control points
// (x1,y1), (x2,y2), (x3, y3), (x4, y4)

void background( float grey )
void stroke( float grey )
void fill( float grey )
// set the colour of the background, stroke or fill to be
// a grey value between 0 (black) and 255 (white)

void noStroke()
void noFill()
// turn off drawing strokes around objects and filling objects, respectively
```

****