

Given Name:_____ Family Name:_____

Student Number:_____ Signature:_____

UNIVERSITY COLLEGE DUBLIN
Faculty of Science

COMP 3003 (Visual Computing - Graphics)
Instructor: Hamish Carr

Sample Exam Paper
December 2004

Duration: 105 minutes

No aids allowed

This examination paper consists of **13** pages and **12** questions. Please bring any discrepancy to the attention of an invigilator. The number in brackets at the start of each question is the number of points the question is worth.

Answer all questions marked MANDATORY and 4 questions marked CHOICE. Since the exam has a total of 70 marks, you should spend approximately 1.5 minutes per mark.

For instructor's use only:

	Score
1 (7)	
2 (7)	
3 (7)	
4 (7)	
5 (7)	
6 (7)	
Subtotal	

	Score
7 (7)	
8 (7)	
9 (7)	
10 (7)	
11 (7)	
12 (7)	
Subtotal	

Total (70)	
------------	--

1. Coordinate Systems (MANDATORY: 7 marks total)

- (a) [2] What are
- homogeneous coordinates*
- ?

Answer: *Homogeneous coordinates* are coordinates in $d+1$ dimensions that represent points in d dimensions in a form that allows projection and translation to be performed by matrix multiplication.

- (b) [2] Show how to convert 3-D
- homogeneous coordinates*
- to 3-D
- cartesian coordinates*
- .

Answer: The homogeneous coordinates (x, y, z, w) refer to the point $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$

- (c) [3] Vertices in coordinate system B can be converted to coordinate system A by matrix multiplication in homogeneous coordinates. Let B be defined by basis vectors
- $(1, 2, 2)_A$
- ,
- $(-4, 1, 1)_A$
- , and
- $(0, 3, -3)_A$
- with origin of
- $(2, -3, 4)_A$
- . Give the homogeneous matrix for converting vertices from coordinate system A to coordinate system B.

Answer: To convert from cartesian basis B to cartesian basis A (assuming a common origin), we simply multiply by the matrix containing the vectors of B in A-coordinates:

$$\begin{bmatrix} 1 & -4 & 0 \\ 2 & 1 & 3 \\ 2 & 1 & -3 \end{bmatrix}$$

Expressed in homogeneous coordinates, this array looks like this:

$$B = \begin{bmatrix} 1 & -4 & 0 & 0 \\ 2 & 1 & 3 & 0 \\ 2 & 1 & -3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

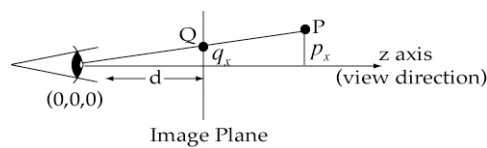
After performing the change of basis, we have to change the origin by translation, multiplying by the following homogeneous matrix:

$$T = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since we perform T *after* B, the final matrix is:

$$T = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -4 & 0 & 0 \\ 2 & 1 & 3 & 0 \\ 2 & 1 & -3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -4 & 0 & 2 \\ 2 & 1 & 3 & -3 \\ 2 & 1 & -3 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Perspective Projection (MANDATORY: 7 marks total)

(a) [1] What is a *vanishing point*?**Answer:** A *vanishing point* for a set of parallel lines in space is the point in the image to which the perspective projections of those lines converge.(b) [1] What is *foreshortening*?**Answer:** *Foreshortening* is the reduction with distance of lengths perpendicular to the direction of view when using perspective projection.(c) [5] Assume that you are standing at the origin, looking down the z-axis with an image plane at a distance d in front of you. Show how to construct the perspective projection matrix that maps vertices in world coordinates to the image plane.**Answer:** In the diagram shown, we are computing the position Q in the image plane of a point P in the world.

Because we have two similar triangles, we know that:

$$\frac{q_x}{d} = \frac{p_x}{p_z}$$

$$q_x = p_x \frac{d}{p_z}$$

and similarly for q_y , while $q_z = d$. Then:

$$(q_x, q_y, q_z) = \left(p_x \frac{d}{p_z}, p_y \frac{d}{p_z}, d \right)$$

$$(q_x, q_y, q_z) = \left(p_x \frac{d}{p_z}, p_y \frac{d}{p_z}, p_z \frac{d}{p_z} \right)$$

$$(q_x, q_y, q_z, q_w) = \left(p_x, p_y, p_z, \frac{p_z}{d} \right)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

as shown in the class slides.

3. The OpenGL (Projective) Pipeline (MANDATORY: 7 marks total)

- (a) [3] What matrices does OpenGL keep track of for you, and what do they do?

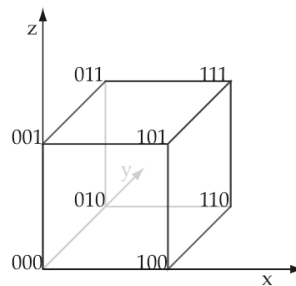
Answer: The *modelview* matrix combines the transformations that describe the position of the model in world coordinates and the position of the world in *view* or *camera* coordinates. It transforms objects from the model to camera coordinates.

The *projection* matrix defines how the projection from view coordinates to the image plane.

The *texture* matrix transforms the texture coordinates of the vertices.

- (b) [4] Give OpenGL code for drawing an unshaded untextured red cube centred at the origin. You may make your cube any size and orientation that is convenient to you. You may not use
- `glutSolidCube()`
- or
- `glutWireCube()`
- .

Answer: Since the cube is unshaded and untextured, we do not specify lighting, materials, normals, or texture coordinates. Sketch the cube and use a translation to move it into place. OpenGL expects triangles to specified in CCW order looking out, so once we have done the $x = 0$ face, the $x = 1$ face is easy: just change the x-coordinate to 1 and reverse the order of the vertices. For an exam answer, don't worry too much about the CCW ordering, and remember that quads are easier to work with for the cube. If you are running out of time on this type of question, make sure you have the first face specified correctly.



```
void drawCube()
{ /* drawCube() */
  glPushMatrix();      glTranslatef(-0.5, -0.5, -0.5);
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_TRIANGLES);          /* you can also use quads */
  glVertex3i(0, 0, 0); glVertex3i(0, 0, 1); glVertex3i(0, 1, 1); /* x = 0 */
  glVertex3i(0, 0, 0); glVertex3i(0, 1, 1); glVertex3i(0, 1, 0);
  glVertex3i(1, 0, 0); glVertex3i(1, 1, 1); glVertex3i(1, 0, 1); /* x = 1 */
  glVertex3i(1, 0, 0); glVertex3i(1, 1, 0); glVertex3i(1, 1, 1);
  glVertex3i(0, 0, 0); glVertex3i(1, 0, 0); glVertex3i(1, 0, 1); /* y = 0 */
  glVertex3i(0, 0, 0); glVertex3i(1, 0, 1); glVertex3i(0, 0, 1);
  glVertex3i(0, 1, 0); glVertex3i(1, 1, 1); glVertex3i(1, 1, 0); /* y = 1 */
  glVertex3i(0, 1, 0); glVertex3i(0, 1, 1); glVertex3i(1, 1, 1);
  glVertex3i(0, 0, 0); glVertex3i(0, 1, 0); glVertex3i(1, 1, 0); /* z = 0 */
  glVertex3i(0, 0, 0); glVertex3i(1, 1, 0); glVertex3i(1, 0, 0);
  glVertex3i(0, 0, 1); glVertex3i(1, 1, 1); glVertex3i(0, 1, 1); /* z = 1 */
  glVertex3i(0, 0, 1); glVertex3i(1, 0, 1); glVertex3i(1, 1, 1);
  glEnd(); glPopMatrix();
} /* drawCube() */
```

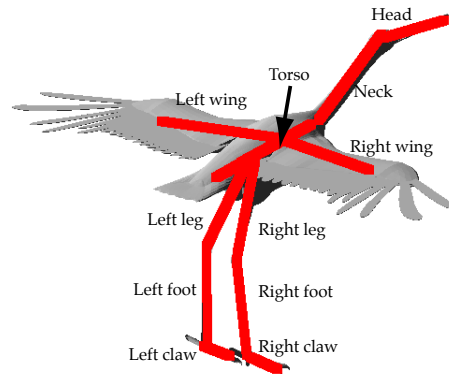
4. Animation (MANDATORY: 7 marks total)

(a) [1] What is a *bone*?**Answer:** A *bone* is an independently transformed part of an articulated model.

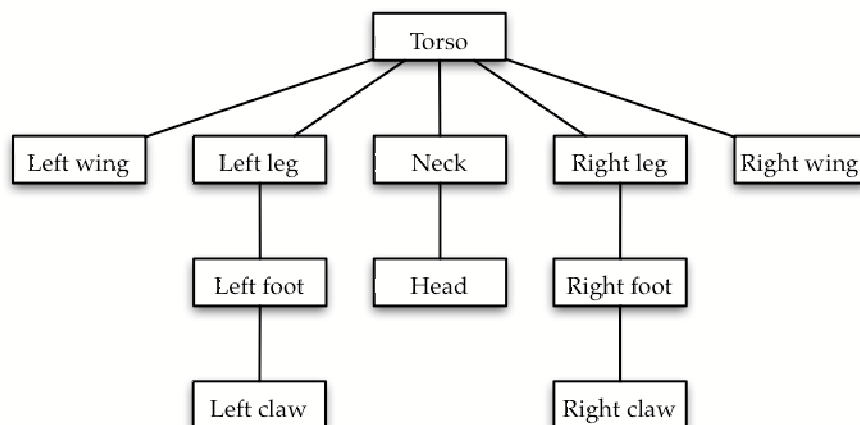
(b) [1] How do OpenGL's matrix stacks help animate characters?

Answer: By providing a simple hierarchical method for specifying the transformation matrices on each bone.

(c) [5] For the model shown of a stork, sketch a suitable bone hierarchy for animating it.



Answer: The easiest way to approach this kind of question is to sketch the animation skeleton directly on the image, as shown (you may need to print in colour or look at the PDF online to see the skeleton here). You need not get the same set of bones shown here. To draw the hierarchy, pick a master bone, then draw a graph showing the hierarchy. In this case, the torso is the most sensible master bone because the centre of gravity is probably inside it.



5. Colour, Lighting and Shading (MANDATORY: 7 marks total)

(a) [3] What is the *Phong lighting model*?

Answer: The *Phong lighting model* approximates the light coming from a surface by decomposing it into *specular*, *diffuse*, *ambient* and *emissive* components. The *specular* component gives the *shiny* or *gloss* reflection from a light source, while the *diffuse* component gives the *matte* reflection from a light source, and the *ambient* component gives the contribution from light in the scene (i.e. directionless light). The *emissive* component is not reflective: it is the amount of light emitted directly from the surface.

(b) [3] Describe how the surface normal vector is used for lighting calculations.

Answer: Both specular and diffuse components of light depend on the cosine of the angle between the normal vector and another vector. For diffuse reflection, the second vector is the vector from the surface to the light source. For specular reflection, the second vector is the average of the vector to the light source and the vector to the eye.

In either case, the cosine of the angle is computed from the dot product of the vectors.

(c) [1] Give RGB coordinates for the colour yellow. (Hint: where is yellow in the rainbow?)

Answer: Since yellow is between red and green, it should combine them: (1, 1, 0) will do nicely.

6. Textures (MANDATORY: 7 marks total)

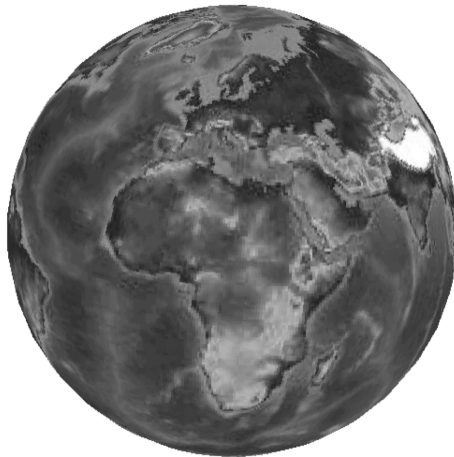
- (a) [3] What is the difference between the implicit, explicit and parametric forms of a line? Give examples.

Answer: The *explicit* form gives y (and z) in terms of x , e.g. $y = 4x - 2$.

The *implicit* form gives x and y (and z) as weighted terms in a sum, e.g. $-4x - y = 2$.

The *parametric* form gives x and y (and z) in terms of a variable t called the *parameter*, e.g. $(x, y) = (t, 4t - 2)$

- (b) [4] The image of the world shown below has been drawn with no lighting, but with a texture on the surface of the earth. Assume that all of the relevant code will be in a single function: what additional OpenGL code is required so that the textured world is properly shaded assuming that the sun is the light source?



Answer: Here is the code. On an exam, you will *NOT* be expected to remember the correct function or constant names, but you should remember the *necessary* steps: turning on lighting, turning on the light, setting the light's position and colour, setting the material properties of the surface and setting the texture to modulate rather than replace.

```
void drawEarth()
{ /* drawEarth() */
  GLfloat sun_position[4] = { 1.0, 0.0, 0.0, 0.0};
  GLfloat white_colour[3] = { 1.0, 1.0, 1.0};
  glEnable(GL_LIGHTING); /* turn on lighting */
  glEnable(GL_LIGHT0); /* turn on the light */
  glLightfv(GL_LIGHT0, GL_DIFFUSE, white_colour);
  glLightfv(GL_LIGHT0, GL_POSITION, sun_position);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, white_colour);
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
  drawTexturedEarth(); /* the existing code */
} /* drawEarth() */
```

7. GUIs and GLUT (CHOICE: 7 marks total)

(a) [2] What is the *event loop*?

Answer: The *event loop* is the core processing loop of a GUI program. Instead of the program dictating the order in which things happen, it waits for the user to interact with the computer in the form of *events* such as mouse movement and clicking, key presses, &c. This event processing is usually coded as a simple while loop, hence the name *event loop*.

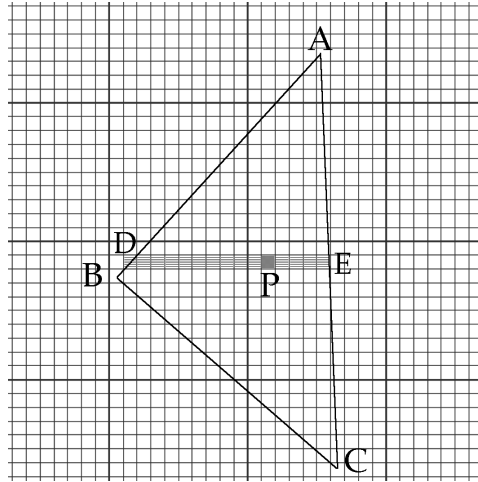
(b) [5] Give code for a simple GLUT program that accepts keyboard input and renders an image. You need not handle mouse input or handle any OpenGL functionality: i.e. you do not have to set matrices, lights, vertices, &c. You should restrict yourself to setting up a window, setting appropriate functions, and setting the program running.

Answer:

```
int main(int argc, char **argv)
{ /* main() */
  glutInit(&argc, argv);          /* initialize GLUT      */
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(640, 480); /* set window size     */
  glutCreateWindow("3003");      /* set window title    */
  glutDisplayFunc(display);      /* set display callback */
  glutKeyboardFunc(keydown);     /* set keystroke callback */
  glutMainLoop();                /* start the event loop */
  return 0;                       /* return 0 when done  */
} /* main() */
```

8. Rasterization and Interpolation (CHOICE: 7 marks total)

- (a) [4] To fill in the triangle shown, explain how to determine which pixels should be set.



Answer: Either of the following solutions is acceptable:

Linewise Scan Sort the vertices A, B, and C so that A has the highest y-value and C the lowest y-value. For integer y-values between A_y and B_y , compute the x-coordinates D_x and E_x where the row of pixels intersects lines AB and AC. Then write a loop that sets all pixels on row D_y with x-coordinates between D_x and E_x .

Half-Plane Test Find the minimum and maximum x- and y- coordinates of A, B, C. For every pixel with x- and y-coordinates in those ranges, use the half-plane test to determine whether the pixel is inside or outside the triangle. The half-plane test checks the pixel against each edge of the triangle: if a pixel is inside with respect to each edge, it is inside the triangle.

- (b) [3] Assume that point A in the picture above is red, point B is blue and point C is green. Explain how to compute the colour of pixel P.

Answer: Either of the following solutions is acceptable:

Linewise Interpolation Perform linear interpolation along AB to find the colour at D, and along AC to find the colour at E. Then perform linear interpolation along DE to find the colour of the pixel P. You may find it easiest to draw directly on the figure as shown.

Barycentric Interpolation Either as part of the half-plane test or separately, compute the barycentric coordinates of P. The barycentric coordinates are then used as weights to compute the colour of P directly from the colours of A, B, C. Multiply the barycentric coordinates by the colours at the vertices and add them together.

9. Blending & Compositing (CHOICE: 7 marks total)

(a) [3] What is the *frame buffer* and what is it used for?

Answer: The *frame buffer* is an array in memory (usually on the video card) where the image being displayed is computed and / or stored. It is used to fill in individual pixels and to perform a variety of operations such as accumulation, stereo imaging, alpha and depth tests, &c.

(b) [4] Describe how to draw translucent objects in OpenGL.

Answer: Draw the non-translucent objects in the scene first. Then turn on blending with `glEnable(GL_BLEND)`. Pixels set by triangles will now combine the value already in the frame buffer with the value set by the triangle. To control how much of each is used, set the blending function with `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)` so that the *alpha* component of an object's colour is the percentage of the new (source) image combined with (100% - alpha) of the old image. Make sure that translucent objects are rendered from back to front.

10. Optimization (CHOICE: 7 marks total)

- (a) [2] What is the difference between transform rate and fill rate? Which is more important for textured objects?

Answer:

The *transform rate* is the rate at which the video card can apply geometric transformations to vertices.

The *fill rate* is the rate at which the video card can compute and set pixel values.

For textured objects, the fill rate is usually more important than the transform rate as there are often many pixels per triangle.

- (b) [3] Describe three ways of increasing the number of triangles your OpenGL card can render in a given amount of time.

Answer:

Use *display lists* to cache geometry on the video card and reduce the amount of traffic on the video bus.

Use *triangle strips* or *triangle fans* to reuse vertices between adjacent triangles.

Use *vertex arrays* to pass all the vertices to the video card once, then refer to vertices by indices in the array.

- (c) [2] What is the difference between *clipping* and *culling*?

Answer:

Clipping chops off parts of the geometry that lie outside the field of view.

Culling discards parts of the geometry that are not visible because they are behind other parts of the object.

11. Image Analysis (CHOICE: 7 marks total)

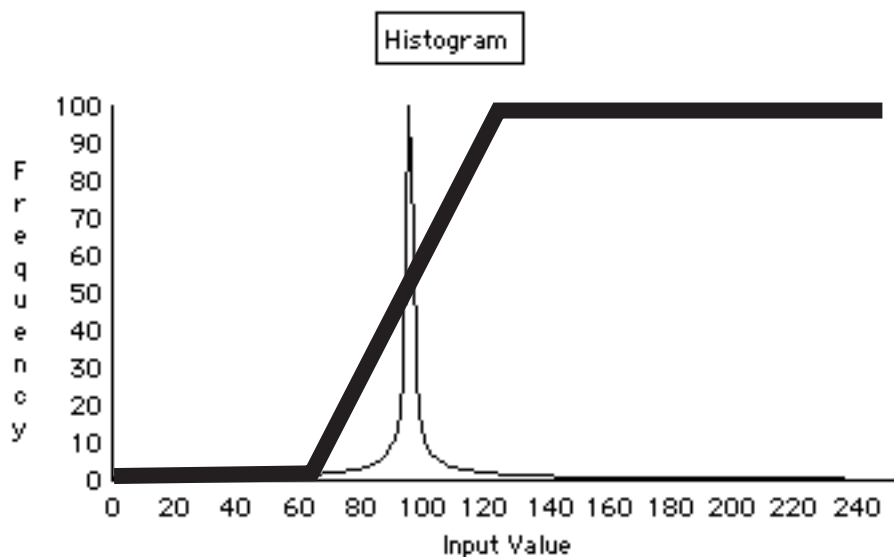
(a) [3] What is a *filter mask* and how is it applied to an image. Show a small example.

Answer: A filter mask is a small array used to define how a pixel is combined with its neighbours. The centre of the filter is superimposed on a pixel in the input image, and the pixels multiplied with the values in the filter. The sum of these multiplications is then used as the value of the corresponding pixel in the output image. For example:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? & ? & ? \\ ? & -2 & 1 & 6 & 1 & ? \\ ? & 1 & 6 & 1 & -2 & ? \\ ? & 6 & 1 & -2 & 0 & ? \\ ? & 1 & -2 & 0 & 0 & ? \\ ? & ? & ? & ? & ? & ? \end{bmatrix}$$

Note that the question marks ('?') indicate that we don't know what value to put there because the filter flops off the edge of the input image.

(b) [4] Suppose that you have the following histogram distribution. Sketch a suitable transfer function to improve the contrast in the image.



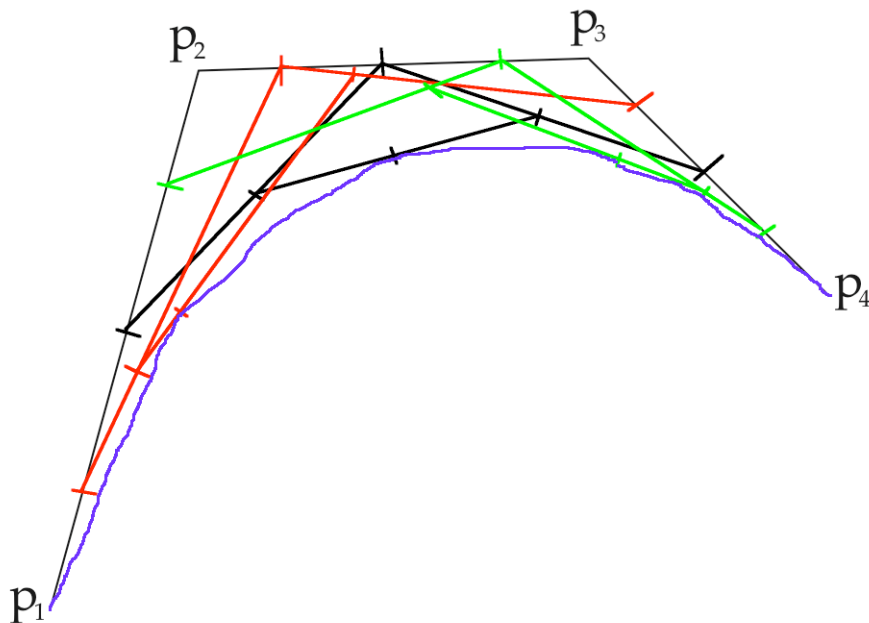
Answer: In this case, the simplest solution is to draw the transfer function directly on top of the histogram, as shown.

12. Curves and Surfaces (CHOICE: 7 marks total)

- (a) [2] What is
- C^1
- continuity and why is it important?

Answer: C^1 continuity means that the first derivative of the function is continuous. It is important because this is one way of expressing the idea of *smoothness*.

- (b) [5] In the following image, use the de Casteljau algorithm to find the points on the Bzier curve at parameter values of
- $t = 0.25$
- ,
- $t = 0.5$
- and
- $t = 0.75$
- . Sketch the Bzier curve that passes through these points.



Answer: You will probably want to print this in colour or look at it online. Since $t = 0.5$ is easiest, you should probably do it first (shown in black). Find the three halfway points along the edges and draw two black lines between them as shown. Then find the halfway points on these edges and draw another black line. Finally, take the halfway point: it is on the Bzier curve.

For $t = 0.25$, do the same, but take the point one quarter of the way along each edge (shown in red). For $t = 0.75$, use the point three-quarters of the way along each edge (shown in green).

Finally, draw a (hopefully) smooth curve from p_1 to p_4 , passing through these points.

End of examination

Total pages: 13

Total marks: 70