# Aston University

January 2013 Examination Session

## CS2150: Computer Graphics

**Date: TBC**
**Time: TBC – TBC**

**Instructions to Candidates:**

- Answer the question from Section A and any TWO questions from Section B.
- Section A is worth 40 marks, each question from Section B is worth 30 marks.
- If you attempt more than two questions from Section B, your best two answers will be counted.

**Materials provided:**

- An Appendix summarising OpenGL commands and relevant elements from the GraphicsLab classes is provided at the end of this paper.

*Model answers to questions are provided in boxes.*

# SECTION A – Compulsory question

1. a) Define what is meant by the term **computer graphics**.

(3 marks)

*Computer graphics is the conversion of a description to an image, via a process typically referred to as the rendering pipeline (3 marks).*

b) What is meant by the term **homogeneous coordinates**? Explain why they are used in computer graphics?

(3 marks)

*Homegeneous coordinates introduce an additional coordinate "w" to the spatial coordinates (1 mark). This allows translations to be expressed as transformation matrices (1 mark). All transformations can be then be multiplied to form a single composite matrix (1 mark).*

c) What role does the **scene graph** play in computer graphics? Sketch a complete scene graph to represent the Sun, the orbiting Earth and the Moon that orbits the Earth.

(6 marks)

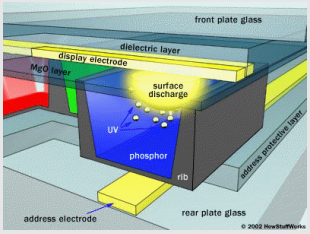
*A scene graph is an abstract representation of the organization / structure of the world model employed in computer graphics, and typically includes geometry, transformation and attribute elements (2 marks). The scene graph I want is going to look something like:*

```
Origin
 |
 +-- [T] Sun
      |
      +-- [R][T] Earth
           |
           +-- [R][T] Moon
```

*If the students get all transformations and objects hierarchy: 4 marks, just objects hierarchy: 2 marks.*

d) Show, *using a sketch with explanation*, the way a **plasma display** works.

(8 marks)




*(4 marks for diagram) A plasma display is made up of a matrix of plasma subpixels, or cells (1 mark). The plasma is activated by applying an electric current via electrodes on either side of the cell (1 mark). Light is then emitted at the plasma's native colour (red, green or blue) (1 mark). The composite colour is made from varying the brightness of the RGB cells (1 mark).*

e) Explain the roles of the **GL_PROJECTION** and **GL_MODELVIEW** matrices in the OpenGL rendering pipeline.

(5 marks)

***GL_MODELVIEW**: the matrix stores the composite transformations (1 mark) applied to the objects prior to their projection. It can be used to move objects and / or the 'camera' (2 marks).*
***GL_PROJECTION**: the matrix that defines the 3D-2D projection (1 mark), which acts like the lens of the camera, determining the perspective effect, and allowing us to 'zoom in' (1 mark).*

f) Describe, *using a sketch with explanation*, the method used to apply **textures** to objects in OpenGL.

(5 marks)

*(2 marks for sketch) Textures are mapped to objects by linking object vertices to the corresponding texture coordinates (1 mark). This is often called the inverse mapping, from screen space (x,y) to texture space (s,t) (1 mark). In effect, for each position in screen space, we compute the corresponding world space coordinate, and from this derive the texture coordinate to obtain the colour (R,G, B) values (1 mark).*

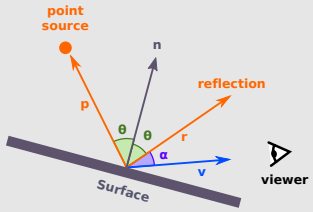
g) Describe what is meant by **back-face culling** and explain how it works. Why is back-face culling very frequently used in visible surface determination?

(4 marks)

*Back-face culling involves the removal of faces of objects that point away from the viewer (1 mark). It is very simple to implement: in canonical view coordinates, discard faces which normal has a negative z-component (2 marks). It is used because visible surface determination is quite complex and this simple preprocessing halves the number of surfaces that need to be considered in visible surface determination. (1 mark)*

h) Show, *using a sketch with explanation*, why the intensity of **specular reflection** depends on the position of both the viewer and the light source, relative to the surface being lit.

(6 marks)




*Needs to include a brief explanation that the light is reflected about a reflection vector – the angle between this and the view angle determines the brightness the viewer sees. In general the specular exponent determines how concentrated about the reflection vector the light is i.e. how shiny the object appears (3 marks for sketch, 3 for explanation).*

***Total marks for Question 1:*** ***40 marks***

# SECTION B

## Answer any two of the four questions in this section.

*Some of the questions refer to OpenGL. An appendix giving the OpenGL API calls, can be found at the end of this paper.*

2. a) How do we define objects for display in OpenGL? In which 3D coordinate system would these objects usually be defined?

(4 marks)

*Objects are defined as a series of polygons (typically quadrilaterals or triangles) which are defined in the 3D world by a series of vertices with x, y and z coordinates (3 marks). The coordinate system used is the "world coordinate system" (1 mark).*

b) Show how to create a Java method, using OpenGL, to draw a unit square aligned with the x and y axes, with the lower left corner at (0.0, 0.0, 0.0). *You do not need to write a complete program, just the method for the rendering of the square. You may assume that all supporting functionality used in the graphics labs is available.*

(8 marks)

```
private void drawSquare()
{
  Vertex v1 = new Vertex(0.0f, 0.0f,  0.0f);
  Vertex v2 = new Vertex(1.0f, 0.0f,  0.0f);
  Vertex v3 = new Vertex(1.0f, 1.0f,  0.0f);
  Vertex v4 = new Vertex(0.0f, 1.0f,  0.0f);

  GL11.glBegin(GL11.GL_POLYGON);
  {
      v1.submit();
      v2.submit();
      v3.submit();
      v4.submit();
  }
  GL11.glEnd();
}
```

*This can be answered in several ways; my example shows the solution as a separate method, which is best, but the code inside the method could be given alone for full marks. Correct vertices (2 marks), correct usage (1 mark), correct use of glBegin / glEnd (2 marks), correct order of vertices (2 marks), overall correct structure / syntax (1 mark).*

c) Show the code needed in the **`initScene()`** method to setup a white light source positioned at (0, 0, 8). The light is assumed to be fully diffused (i.e. you can ignore the other components).

(6 marks)

*In the **`initScene()`** method, we need to add:*

```
// Define the light properties
float diffuse0[]  = { 1.0f,  1.0f, 1.0f, 1.0f};
float position0[] = { 0.0f, 0.0f, 8.0f, 1.0f};

// Apply these properties
GL11.glLight(GL11.GL_LIGHT0,GL11.GL_DIFFUSE,
  FloatBuffer.wrap(diffuse0));
GL11.glLight(GL11.GL_LIGHT0,GL11.GL_POSITION,
  FloatBuffer.wrap(position0));

// Enable lighting and switch on the light
GL11.glEnable(GL11.GL_LIGHT0);
GL11.glEnable(GL11.GL_LIGHTING);
```

*This is a minimal requirement I will be quite lenient about the exact syntax, but the elements must be here (6 marks). The light colour and position should be set correctly.*

d) To enable the square from (b) to react to light, what do you need to define in its rendering method? Indicate what OpenGL code should be included to that effect.

(3 marks)

*The normal must be defined for the square to react to light (1 mark). The following statement needs to be added after **`GL11.glBeginGL11.GL_POLYGON`**:*

```
new Normal(v1.toVector(), v2.toVector(), v3.toVector(),
  v4.toVector()).submit();
```

*2 marks for code (if the normal was defined in (b), give the marks here).*

e) Show how the **`renderScene()`** method needs to be updated for the square from (b) to reflect blue diffuse light. We assume that the square has equal reflection properties on both sides and only reflects diffuse light (i.e. you can ignore the other light components).

(3 marks)

*In the renderScene method we need to defined the diffuse properties:*

```
float diffuse[]  = {0.0f, 0.0f, 1.0f, 1.0f};
```

*and apply them before drawing the square:*

```
GL11.glMaterial(GL11.GL_FRONT_AND_BACK,GL11.GL_DIFFUSE,
  FloatBuffer.wrap(diffuse));
```

*2 marks for correct elements, 1 mark for correct values.*

f) Describe BOTH the **GL_FLAT** and **GL_SMOOTH** lighting models used in OpenGL.

(6 marks)

*GL_FLAT shades all faces (polygons) of an object with the same colour, as determined by the average of the normals of all the vertices that make up the face, or the single normal if only one is given, using the basic OpenGL lighting / materials model (3 marks). The GL_SMOOTH model uses Gouraud shading, where the lighting is first computed, using the basic OpenGL lighting / materials model, at each vertex, and this is then interpolated smoothly across the surface producing a smoother appearance to the object (3 marks).*

***Total marks for Question 2:*** ***30 marks***

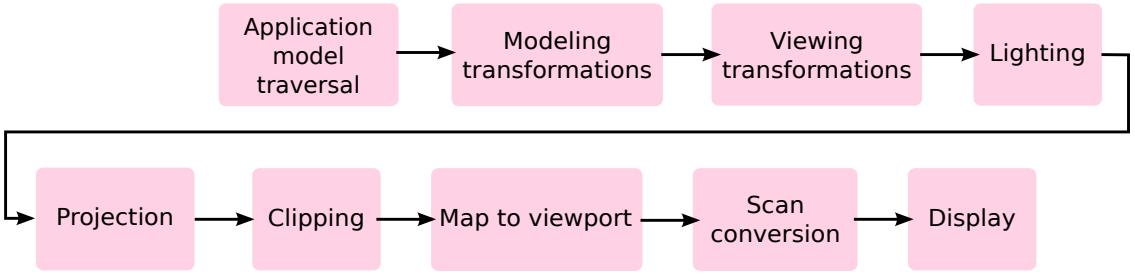
---

3. a) What is meant by a **rendering pipeline**?

(3 marks)

*The rendering pipeline is the complete process of the conversion of geometrical primitives to screen pixels. It includes the specification / traversal of the 3D world model through to the setting of the RGB values of the screen pixels (3 marks).*

b) Briefly describe each of the components in the **rendering pipeline** with **Gouraud shading**.



(19 marks)

*(a) Application model traversal: extract relevant objects for the part of the scene visible based on the view volume / world model (2 marks),*

*(b) Modeling transformations: movement, and location of objects using transformations such as scaling, rotation and translation (3 marks)*

*(c) Viewing transformations: position, aim and orientation of the camera (2 marks)*

*(d) lighting: compute the colour of the objects given the light and materials using the simplified lighting model used in computer graphics that considers ambient, diffuse, specular and emissive lighting (3 marks)*

*(e) projection: projection to canonical view volume using transformations to define the viewer position and viewing characteristics (3 marks),*

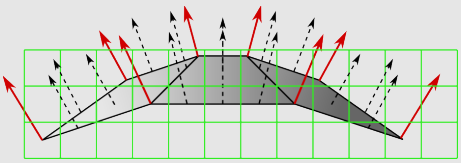
*(f) clipping: remove primitives lying outside the view volumne (2 mark),*

*(g) map to viewport: scale the canonical view volume (and all objects along with it) to actual viewport size (2 mark),*

*(h) scan conversion: from objects to pixels, typically based on the z-buffer algorithm that also computes visibility (2 marks)*

c) Explain, *with a simple sketch*, how **Phong shading** works.

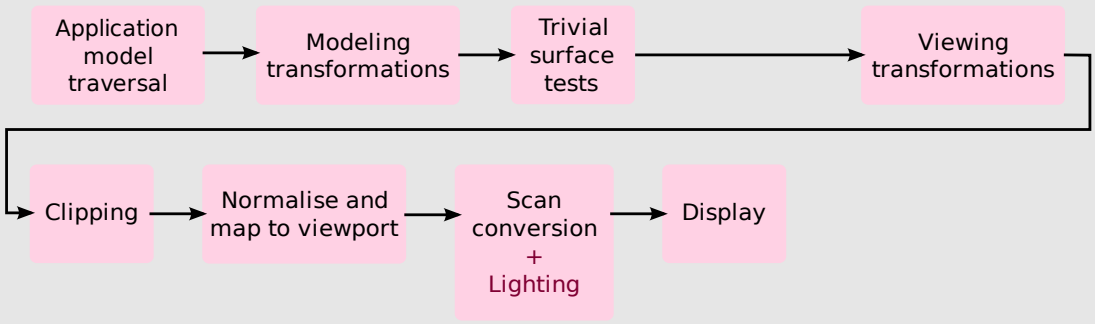
(4 marks)



*(2 marks for sketch) In Phong shading, the colour at a given pixel is computed using the approximate normal to the visible polygon at that pixel. The approximate normal is obtained by interpolating between the known polygon vertex normals. (2 marks)*

d) Explain WHY and HOW the **rendering pipeling** from question (b) needs to change when using **Phong shading**.

(4 marks)




*In Phong shading, the lighting is pixel-dependent (2 marks) and thus needs to be computed at a later stage in the pipeline, typically as part of scan conversion (2 marks). A graph isn't expected for a full mark.*

***Total marks for Question 3:*** ***30 marks***

---

4. a) Explain the role, when implementing computer graphics, of a graphics **Application Programming Interface** (API) such as OpenGL.

(4 marks)

*An API is meant to provide a fixed abstraction and in the context of graphics the key role is to provide a hardware abstraction layer, so that applications can use the fixed interface while the underlying hardware evolves but still employs the OpenGL API for its instruction (4 marks).*

b) Why are **composite transformation matrices** very important in computer graphics?

(5 marks)

*Composite transformations allow us to group together transformations so that a single matrix can be used to represent the combined effect of a large number of transformations (2 marks). This allows for greater efficiency because the composite matrix can be used to multiply the vertices of the objects once, to achieve a series of combined transformations (1 marks) such as often occur, for example in the viewing part of the rendering pipeline (1 mark). Also, the inverse composite transformation can be easily computed without having to invert each transformation separately (1*

*mark).*

c) Show how a triangle given by the vertices (0,0), (0,1) and (1,1) could be uniformly **scaled** by a factor of 4, and THEN **translated** by the vector (4,2), using the **composite transformation matrix** to compute the resulting coordinates of the vertices.

(9 marks)

*Scaling matrix:* $S = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ *(2 marks), translation matrix:* $T = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$ *(2 marks). Composite matrix:* $C = T \times S = \begin{bmatrix} 4 & 0 & 4 \\ 0 & 4 & 2 \\ 0 & 0 & 1 \end{bmatrix}$ *(3 marks). The new points are: (4,2), (4,6), (8,6), (1 mark) but I want to see that they have used the matrix for full marks (i.e.* $\begin{bmatrix} 4 & 0 & 4 \\ 0 & 4 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$ *) (2 marks).*

d) Show the **inverse composite transformation matrix** of the composite matrix developed in part (c) above.

(6 marks)

*The key here is knowing that* $(TS)^{-1} = S^{-1}T^{-1}$*. So the inverse composite matrice are:* $\begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/4 & 0 & -1 \\ 0 & 1/4 & -1/2 \\ 0 & 0 & 1 \end{bmatrix}$ *(2 marks each for correct matrices, 2 marks for correct inverse composite).*

e) Explain the concept of the **matrix stack** in OpenGL AND discuss its relationship with **composite transformation matrices** and **scene graphs**.

(6 marks)

*OpenGL only has a single "current" composite transformation matrix, the MODELVIEW matrix (1 mark). It is however possible to save the current MODELVIEW matrix by pushing (1 mark) it onto a matrix stack (1 mark) and restoring it at a later point by popping (1 mark) it off the stack, discarding any transformations applied since the push. This makes it possible to localise*

*transformations, so that they only apply to a group of objects (1 mark for mentioning localisation/discarding), and allows to have hierarchies of objects in a manner similar to that described by the scene graph (each child of and object would have its transformations localised inside a push/pop block, within the parent's push/pop block.) (1 mark)*

***Total marks for Question 4:*** ***30 marks***

---

5. a) Describe the 2 types of **projections** available in OpenGL and give an example of a suitable application for each type.

(6 marks)

*2 marks for listing each type of projection.*
***Perspective projection****: the projectors converge to the centre of projection (the eye). Foreshortening is preserved (1 mark) and thus this type of projection is more suited to application where realism is important, such as movies and games (1 mark).*
***Parallel projection****: the projectors are parallel to one another (no convergence). Parallel projections preserve parallelism and lengths (1 mark) and are thus better suited to Computer Aided Design applications (1 mark).*

b) Define what a **vanishing point** is and draw a **2-point perspective projection** of a cube.

(4 marks)

*In a perspective projection, parallel edges of an object which are not also parallel to the projection plane will meet, when extended, at a unique point in the distance, called vanishing point (e.g. train tracks joining at the horizon). (2 marks + 2 marks for sketch.)*

c) Define what is meant by **visible surface determination**.

(3 marks)

*Visible surface determination is the act of defining which surfaces are visible, from the view point, often prior or as part of the rasterisation step (3 marks).*

d) Discuss TWO methods, excluding back-face culling, that can be used to speed up visible surface determination.

(8 marks)

*There are many options. Id expect them to describe spatial partitioning, which divides the world (or sometimes screen) space into regions, which can then be treated separately, which is particularly effective on parallel hardware, but also helps in a serial approach since the number of object / pixel comparisons in object or hybrid approaches is reduced (4 marks). Id also expect them to discuss pre-sorting (to scan convert the polygons from the closets first to minimize the number of lighting evaluations required) or bounding extents (to allow easy tests for object overlap based on the bounding extent and a mini-max test which is very fast) with discussion 4 marks for either.*

e) Give the purpose of the **z-buffer algorithm** and explain, *using pseudo-code and a description*, how the algorithm works.

(9 marks)

*The z-buffer algorithm performs both scan conversion (rasterisation) and visible surface determination. (1 mark) The algorith works by rasterising each polygon, overriding pixel colour if the depth (z) value of the polygon at the pixel is greater than the current one. (1 marks + 7 marks for listing)*

```
void zBuffer ()
{
  int pz; // Polygon's z at pixel (x,y)

  // Fill all pixels with background colour
  for (y = ymin; y <= YMAX; y++) {
    for (x = xmin; x <= XMAX; x++) {
      WritePixel(x,y,BACKGROUND\_VALUE);
      WriteZ(x,y,0);
    }
  }

  // Overwrite pixel colour with polygon's if closer
  for (each polygon) {
    for (each pixel in the polygon's projection) {
      pz = polygon's z value at (x,y);
      if (pz >= ReadZ(x,y)) {
        WritePixel(x,y,polygon colour);
        WriteZ(x,y,pz);
      }
    }
  }
}
```

***Total marks for Question 5:*** ***30 marks***

# Appendix

## OpenGL commands (GL11 class)

```
static void glBegin(int mode)
static void glBindTexture(int target, int texture)
static void glColor3f(float red, float green, float blue)
static void glDisable(int flag)
static void glEnable(int flag)
static void glEnd()
static void glFlush()
static void glFrustum(double left, double right,
                      double bottom, double top,
                      double zNear, double zFar)
static void glLight(int light, int pname, java.nio.FloatBuffer params)
static void glLineWidth ( float width )
static void glLoadIdentity ()
static void glMaterial(int face, int pname, java.nio.IntBuffer params)
static void glMatrixMode(int mode)
static void glNormal3f(float nx, float ny, float nz)
static void glPointSize(float size)
static void glPolygonMode(int face, int mode)
static void glPopAttrib()
static void glPopMatrix()
static void glPushAttrib(int flag)
static void glPushMatrix()
static void glRotatef(float angle, float x, float y, float z)
static void glScalef(float x, float y, float z)
static void glShadeModel(int mode)
static void glTexCoord2f(float s, float t)
static void glTranslatef(float x, float y, float z)
static void glVertex3f(float x, float y, float z)
static void gluLookAt ( float eyeX, float eyeY, float eyeZ,
                        float centerX, float centerY, float centerZ,
                        float upX, float upY, float upZ )
```

## OpenGL enumerated types (GL11 class)

```
face:      GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
light:     GL_LIGHTi
mname:     GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_SHININESS, GL_EMISSION
mode:      GL_POINTS, GL_LINES, GL_TRIANGLES, GL_QUADS, GL_POLYGON
mmode:     GL_MODELVIEW, GL_PROJECTION
pname:     GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION
polymode:  GL_POINT, GL_LINE, GL_FILL
shademode: GL_FLAT, GL_SMOOTH
flag:      GL_LIGHTING, GL_TEXTURE_2D, GL_LIGHTING_BIT
```

## GraphicsLab package

```
new Vertex( float x, float y, float z )  // instantiates a new vertex
v.submit()                              // submits the vertex v for drawing
v.toVector()                             // converts a vertex v to a Vector
new Normal( Vector, Vector, Vector ) // instantiates a Normal object
                                        // based on the 3 argument vectors
n.submit()                                      // submits the normal n
```

END OF EXAMINATION PAPER