Aston University

January 2014 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) Briefly describe the role of the **application model**, the **application program** and the **graphics system**, in the conceptual model of computer graphics.

Application model: holds the details of the objects to be rendered on screen.
Application program: mediates between the graphics system and the objects: e.g. defines response to user inputs, converts application model to commands driving the graphics system...
Graphics system: handles output and user input.

(3 marks)

b) Explain the role of the **GL_MODELVIEW matrix** and the **matrix stack** in OpenGL.

GL_MODELVIEW: the matrix stores the composite (1 mark) transformations applied to the objects prior to their projection. It can be used to move objects and / or the 'camera' (2 marks). Matrix stack: Used in animation – it is a stack for matrices, so the current modelview matrix can be stored (pushed) and restored (popped) (2 marks). The key role is to allow local transformations to be applied to a small set of objects, which might have other transformations applied first (2 marks).

(7 marks)

c) When referring to a 3D object we talk about its **geometric**, **topological** and **attribute** properties. In computer graphics terms, to what do these relate?

Geometric: location in space.
Topological: connectedness, wrt to other parts of the object / external objects.
Attribute: colour, material properties, sometimes information not related to visual.

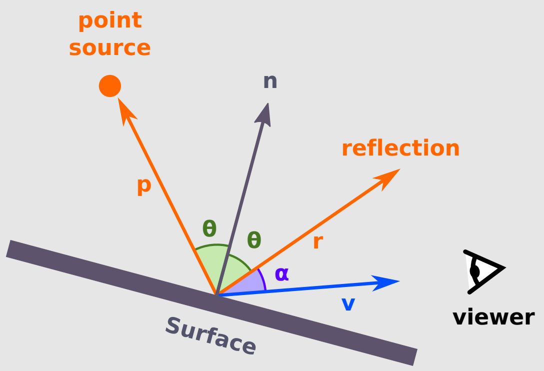
(3 marks)

d) What is the purpose of **display lists** in OpenGL?

Display lists allow us to store the precomputed geometry of an object (e.g. primitives) so that we don't need to rebuild the object (i.e. redo the computations) at every frame. This makes the rendering more efficient.

(3 marks)

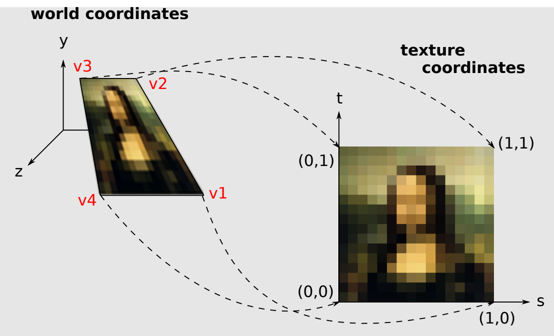
e) Explain, *using a sketch*, why **specular reflection** of light from an object depends on both the location of the light source and the viewer.



The sketch must include a description: The majority of reflection is about the reflection vector, but some 'leaks' away from the reflection vector because the surface is an imperfect mirror. The amount of specular intensity perceived decreases as the angle α between the viewer and the reflection vector increases. This angle depends both on the viewer's location and (through the reflection) on the light source (4 marks for diagram, 3 for explanation).

(6 marks)

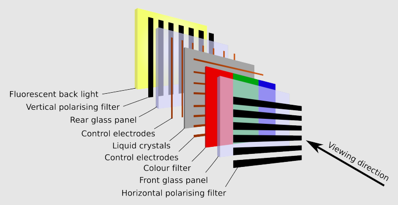
f) *Using a sketch and description*, show how **texture mapping** is achieved in practice.

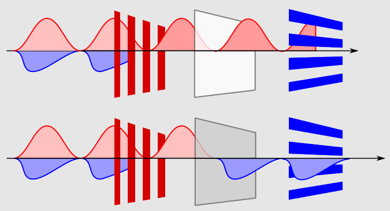


The mapping is the 'inverse mapping', from screen coordinates to texture coordinates – that is for each pixel of an object on the screen, we ask where is the relevant texture coordinate. The mapping is defined by matching (s,t) texture coordinates to (x,y,z) polygon vertices. (3 marks for the diagram, 3 for the description)

(6 marks)

g) Show, using a sketch with explanation, how a modern **liquid crystal display (LCD)** monitor works.





In a liquid crystal display, light (usually provided by a neon tube at the back of the display) needs to go through 2 orthogonal polarizing filters. By default, these filters don't let the light through. Liquid crystals, placed in between the fiters, can change the polarisation of light, and control how much light is let through. The change in polarisation is controlled by applying a voltage to the crystals, which in modern displays is achieved via a thin-film transistor (TFT) matrix (one transistor/capacitor per RGB subpixel). (3 marks for sketch + 4 marks for sensible description)

(7 marks)

h) Explain the difference between the **GL_FLAT** and **GL_SMOOTH** lighting options in OpenGL.

GL_FLAT shades the whole polygon (i.e. decides on the effect of lighting) based on an average normal for that polygon (2 marks). GL_SMOOTH uses Gouraud shading (1 *mark) where the lighting is computed at each polygon vertex (1 marks) and this is interpolated over the polygons surface (1 mark).*

(5 marks)

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. You can assume that all the helper classes provided in the GraphicsLabs package are available (Colour, Vertex, Normal, ...)

2. a) We consider a 2D scene depicting a building with a lift on its side. Write a scene graph for this scene.

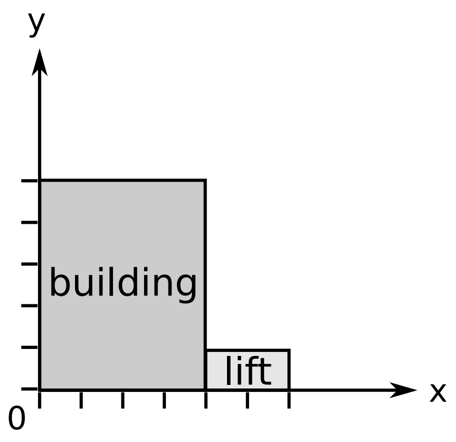


Figure 1: A building with a lift

```
+ Scene origin
|
+-- Building
    |
    +-- T() Lift
```

2 marks for correct scene graph + 1 mark for translating the lift (they might also include a scaling)

(3 marks)

b) Write a `drawUnitSquare()` method which draws a unit square in the x-y plane (z = 0), with its bottom left corner at the origin.

```
public void drawUnitSquare()
{
  Vertex v1 = new Vertex(0, 0, 0);
  Vertex v2 = new Vertex(1, 0, 0);
  Vertex v3 = new Vertex(1, 1, 0);
  Vertex v4 = new Vertex(0, 1, 0);

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

2 marks for correct vertices, 1 mark for correct square (counter-clockwise vertex order), 2 marks for begin/end block.

(5 marks)

c) Using the `drawUnitSquare()` method from question (b), write a `renderLift()` method which draws the lift as a blue rectangle with bottom left corner at the origin. You will assume that the lift is 2 units wide and 1 unit high.

```
public void renderLift()
{
  Colour.BLUE.submit();  // Or: glColor3f(0f, 0f, 1f);
  glScalef(2f, 1f, 1f);
  drawUnitSquare();
}
```

1 mark for colour, 1 mark for scaling and 1 mark for drawing the unit square.

(3 marks)

d) Using the `drawUnitSquare()` and the `renderLift()` methods, write a `renderScene()` method which draws the building with the lift on its side. The building should be drawn as a red rectangle of size 4 by 5, with its bottom left corner at the origin. The lift should be positioned about the building as shown on Figure 1.

```
public void renderScene()
{
  // Scale the building to size, making sure it doesn't
  // affect the lift
  glPushMatrix();
  {
    Colour.RED.submit(); // Or: glColor3f(1f, 0f, 0f);

    glScalef(4f, 5f, 1f);
    drawUnitSquare();
  }
  glPopMatrix();

  // Position the lift about the building
  glPushMatrix();
  {
    glTranslate(4f, 0f, 0f);
    renderLift();
  }
  glPopMatrix();
}
```

1 mark for colour, 1 for scaling, 1 for drawing the square, 2 for push/pop block (to prevent the scaling from applying to the translation further below), 1 for correct translation, 1 for renderLift(), 1 for second push/pop block.

(8 marks)

e) Explain how you would change the `renderScene()` method in order to animate the lift so it moves up to the top of the building. Write the appropriate `updateScene()` method and explain the role of any new variables you declare.

I would declare global variables to hold the y position of the lift (1 mark), initialised to 0 (1 mark), and another variable for its maximum limit (1 mark for correct maximum):

```
private float yLift = 0f;
private float yMax = 4;  // Height of building minus height
                         // of lift
```

I would update it in the `updateScene()` *so it increases, until it reaches the maximum value (the top of the building - the height of the lift):*

```
public void updateScene()
{
  if (yLift < yMax)
  {
    yLift += 0.1;   // Increment by a small amount
  }
}
```

(2 marks for correct test, 2 marks for incrementing yLift). I would finaly use `yLift` *to translate the lift in the* `renderScene()`*:*

```
...
glTranslate(4f, yLift, 0f);
renderLift();
...
```

2 marks for modified translation. 2 marks for overall explanation.

(11 marks)

Total marks for Question 2: 30 marks

3. a) Explain what **homogeneous coordinates** are AND why are they used in computer graphics.

Homogeneous coordinates extend the natural coordinates (x,y,z) with an additional dimension-less component w (1 mark), usually set to 1. Homogeneous coordinates allow transformations, in particular translations, to be expressed as matrices (1 mark) which makes it possible to combine several transformations into a single composite matrix (1 mark).

(3 marks)

b) Show how a simple 2D translation of (2,1) can be represented as a transformation matrix.

$$T(2,1) = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

(3 marks)

c) Apply the translation matrix from question (b) to the triangle (0,0), (1,1), (2,0) and give the resulting vertices. *Your answer should demonstrate how you get to the result.*

The new points are: (2,1), (3,2) and (4,1). Full workings required for the full mark (not just the result). 2 marks for workings, 1 mark for correct result.

(3 marks)

d) Give the transformation matrix for a non-uniform scaling of 1 in the x direction, and 3 in the y direction.

$$S(1,3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(3 marks)

e) Apply the scaling matrix from question d) to the vertices you computed in question c). *Your answer should demonstrate how you get to the result.*

The resulting points are: (2,3), (3,6) and (4,3). Again, workings required for the full mark.

(3 marks)

f) Compute the composite transformation matrix for the translation from (b) followed by the scaling from (d), and apply it to the original triange.

$$C = S(1,3)T(2,1) = \begin{bmatrix}1 & 0 & 0\\0 & 3 & 0\\0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}1 & 0 & 2\\0 & 1 & 1\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 2\\0 & 3 & 3\\0 & 0 & 1\end{bmatrix}$$

New vertices: (2,3), (3,6) and (4,3). 1 mark for correct order, 1 mark for result, 2 marks for vertices with workings.

(4 marks)

g) Give the 2 inverse transformation matrices for the translation from question (b) and the scaling from question (d).

$$T(2,1)^{-1} = T(-2,-1) = \begin{bmatrix}1 & 0 & -2\\0 & 1 & -1\\0 & 0 & 1\end{bmatrix}$$

$$S(1,3)^{-1} = S(1,\frac{1}{3}) = \begin{bmatrix}1 & 0 & 0\\0 & \frac{1}{3} & 0\\0 & 0 & 1\end{bmatrix}$$

(4 marks)

h) Using the results from question (g), compute the inverse transformation to the composite transformation you have developed in question (f). *Your answer should include the computations, not just the result.*

$$(ST)^{-1} = T^{-1}S^{-1} = \begin{bmatrix}1 & 0 & -2\\0 & 1 & -1\\0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}1 & 0 & 0\\0 & \frac{1}{3} & 0\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & -2\\0 & \frac{1}{3} & -1\\0 & 0 & 1\end{bmatrix}$$

(4 marks)

i) If you applied the inverse transformation you computed in question (h) to the vertices you computed in question f), which vertices would you obtain as a result?

We would get the original vertices back, i.e. (0,0), (1,1) and (2,0). 3 marks for correct answer, computations not required.

(3 marks)

Total marks for Question 3: 30 marks

4. a) What is meant by **visible surface determination**?

It is the determination of which faces of object are visible and which are occluded (2 marks) from the view reference point (1 mark).

(3 marks)

b) Explain the difference between **pixel** and **object** approaches to **visible surface determination**.

Pixel: search each pixel and find the object that is closest, overwriting the value that previously existed.
Object: compare all objects to find which is closest to the viewer, and then draw those that are visible. (4 marks)

(4 marks)

c) Give ONE example why a pixel based approach is often more appropriate and ONE example where an object based approach may be faster.

Pixel based is easier to implement because this can be done as part of the scan conversion routine. OR Less sensitive to objects interpenetrating etc – thus fewer special conditions. (2 marks) Object based approaches are faster when we are dealing with high resolution displays (many pixels) or small numbers of discrete objects. (1 mark)

(3 marks)

d) Describe, using pseudo-code AND textual description, how the **z-buffer algorithm** works.

```
void zBuffer () {
   int pz; /* Polygons z at pixel (x,y) */
   for (y = ymin; y <= YMAX; y++) {
      for (x = xmin; x <= XMAX; x++) {
         WritePixel(x,y,BACKGROUND_VALUE);
         WriteZ(x,y,0);
      }
   }
   for (each polygon) {
      for (each pixel in the polygons projection) {
         pz = polygons z value at (x,y);
         if (pz >= ReadZ(x,y)) {
            WritePixel(x,y,polygon colour);
            WriteZ(x,y,pz);
         }
      }
   }
}
```

The z-buffer algorithm relies on 2 buffers: one to store pixel colours (framebuffer) and one to store the closest object's depth at each pixel (z-buffer). The z-buffer values are initialised to 0 (or whatever the lowest z-value in the canonical view volume is). At each pixel, polygons intersecting with that pixel are processed in sequence. Whenever a polygon is found which z value is greater than that stored in the z buffer, the pixel colour is set to the polygon colour, and the z buffer value at that pixel is set to the polygon's z value. At the end of the algorithm, the pixel will have the colour of the closest polygon. 10 marks for clear explanation and/or correct pseudo-code.

(10 marks)

e) Give TWO methods that can be used to speed up the z-buffer algorithm AND briefly explain what they do.

1. Spatial partitioning: split the area of interest into smaller regions where fewer objects need to be compared. (2 marks)
2. Back face culling: scan convert only those faces pointing toward the viewer. (2 marks).

(4 marks)

f) Give 3 data structures used to model polygon meshes and explain briefly how they differ.

1. Explicit representation: each polygon is defined as a set of vertices. Duplication of vertices and edges.
2. Pointers to vertex list: vertices are stored in a list, polygons store pointers to their respective vertices. Duplication of edges.
3. Pointers to edge list: vertices and edges are both stored in a list. Edges point to their end vertices, polygons point to their edges. No duplication.
1 mark for each data structure and 3 marks for descriptions.

(6 marks)

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

5. a) Define what is meant by the **RGB colour model** AND explain how a colour is defined in this model.

RGB stands for Red, Green, Blue, which decomposes light into these three colour chanels (2 marks). Colour is defined as a combination of RGB values specifying the intensity in each chanel. (2 marks) Bright colours correspond to high values, while dark colours have low values (additive model). (1 mark)

(5 marks)

b) List, with a description and a real-life example for each, THREE possible interactions of light with a physical object.

Three possible interactions (2 marks each) would be:material properties 1. Absorption – the light energy is absorbed by the object, and this is then lost to the scene, e.g. a black object
2. Reflection – the light energy is entirely bounced back, along the reflection vector from where it came, e.g. a mirror
3. Scattering – the light is reflected in many directions (equally and randomly), e.g. a matte surface.

(6 marks)

c) List, with a description for each, the FOUR **material properties** that may be set in OpenGL AND indicate how each affects the **shading** of an object.

Possible setting are (4 needed, 2 marks each): GL_AMBIENT: the reflection of the material to ambient light – this will typically be used for the basic 'background lighting'.
GL_DIFFUSE: the diffuse reflection (scattered light), used to represent matte objects.
GL_SPECULAR: the reflection for an imperfect mirror – this is the shiny spots on quite reflective objects (but not mirrors).
GL_SHININESS: this is the specular exponent, which determines just how shiny the object looks – high values look more metallic / polished.
GL_EMISSION: allows an object to "reflect" more light than it receives, giving it a glowing appearance.

(8 marks)

d) Describe, using brief code examples, how you would set up **lighting** in OpenGL (typically in the `initScene()` method).

In OpenGL lights need to be set up first (some detail needed, if some steps / details are missed marks can still be given, one for each line – 7 marks):

```
// global ambient light level
float globalAmb[] = {0.2f,  0.2f,  0.2f, 1.0f};
GL11.glLightModel(GL11.GL_LIGHT_MODEL_AMBIENT,FloatBuffer.wrap(globalAmb)

// Light source
float diffuse0[]  = { 0.6f,  0.6f, 0.6f, 1.0f};
float ambient0[]  = { 0.1f,  0.1f, 0.1f, 1.0f};
float pos0[] = { 0.0f, 10.0f, 5.0f, 1.0f};

GL11.glLight(GL11.GL_LIGHT0, GL11.GL_AMBIENT, FloatBuffer.wrap(ambient0))
GL11.glLight(GL11.GL_LIGHT0, GL11.GL_DIFFUSE, FloatBuffer.wrap(diffuse0))
GL11.glLight(GL11.GL_LIGHT0, GL11.GL_SPECULAR, FloatBuffer.wrap(diffuse0)
GL11.glLight(GL11.GL_LIGHT0, GL11.GL_POSITION, FloatBuffer.wrap(pos0));

// Enable light, lighting and normalisation
GL11.glEnable(GL11.GL_LIGHT0);
GL11.glEnable(GL11.GL_LIGHTING);
GL11.glEnable(GL11.GL_NORMALIZE);
```

(7 marks)

e) Give ONE example of a physical effect that is not represented in the OpenGL lighting model. Discuss how important this is for visual realism.

Many are possible but I would expect to see shadows here. They are rather important to visual realism, so they are a big loss, but can be approximated using OpenGL methods, such a stencil buffers. Reflections and refrations are also missing in OpenGL.

(4 marks)

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