Aston University

January 2012 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) Describe what is meant by a rendering pipeline.

The set of processes that translate (1 mark) the description of a "world" in terms of graphics primitives (1mark) into the image on the computer screen (1 mark).

(3 marks)

b) Describe the THREE primitive types typically used in OpenGL to define objects, and provide OpenGL code fragments to illustrate the creation in a display function of an example of each primitive.

Points:

```
glBegin(GL_POINTS);
  glVertex2i(21,23);
glEnd();
```

Lines:

```
glBegin(GL_LINES);
  glVertex2i(21,23);
  glVertex2i(24,25);
glEnd();
```

Polygons:

```
glBegin(GL_TRIANGLES);
  glVertex2i(21,23);
  glVertex2i(24,25);
  glVertex2i(27,25);
glEnd();
```

(2 marks for each fully correct description, 1 for a partial answer).

(6 marks)

c) Give THREE different examples of how a 2D texture could be generated.

Photograph, computer artist generated, or procedural methods (1 mark each).

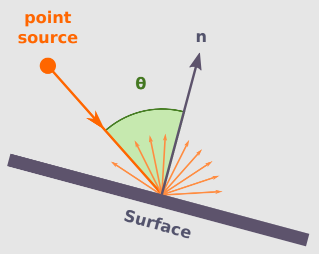
(3 marks)

d) What is the reason that most computer graphics applications use homogeneous coordinates?

Homogeneous coordinates allow all affine transformations (1 mark) to be written as transformation matrices (1 mark). This allows us to produce composite transformation matrix representing the compound effect of multiple transformations very simply using matrix multiplication (2 marks)

(4 marks)

e) Explain, using a sketch with description, why diffuse reflection of light from an object depends only on the location of the light source and not the viewer.



Some text to indicate that scattering is equal in all directions, so the only factor that is important is the amount of illumination the object gets from the given light source, which is a function of the angle between the light source and the object only (2 marks for a clear drawing, 2 marks for correct explanation).

(4 marks)

f) List the TWO main types of **projections** AND explain how they differ. For each type of projection, indicate an area of computer graphics where it is commonly used.

2 types of projections: perspective and parallel projections (2 marks).
***Perspective projection**: used whenever realism is required, .e.g 3D animation, video games... (1 mark)*
***Parallel projection**: preserve parallel lines and, often, lengths. More suitable for design (CAD, 3D modelling...) (1 mark)*

(4 marks)

g) Give the sequence of steps used in the **design** of a new 3D object.

The steps below ought to be mentioned (or similar, within sensible interpretation):

(a) Draw the object on paper (0.5 mark)

(b) Place the origin of the object (0.5 mark)

(c) Draw the x, y and z axes (0.5 mark)

(d) Indicate the size of the object (0.5 mark)

(e) Work out the vertices' coordinates given the origin, axes and object size (1 mark)

(f) Work out the faces (1 mark): the vertices must be passed in counter-clockwise order from the point of view of an external observer facing the face (1 mark)

(g) Work out the normals (1 mark)

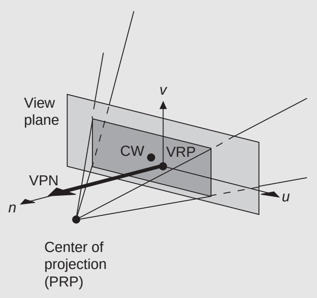
(6 marks)

h) Why are transformation matrices applied in the opposite order to that in which they appear in OpenGL code?

*In OpenGL the current (MODELVIEW) matrix, M, is a composite transformation matrix (1 mark) that represents all the transformations applied to that point. When a new transformation instruction appears in OpenGL this post-multiplies (right multiplies) the transformation matrix to produce the new one e.g. M ¡- M*S (2 marks) thus the last transformations in the code are applied first to the vertices (1 mark).*

(4 marks)

i) Show, using a sketch with explanation, the elements that are required to define the **perspective projection** of a 3D scene to a 2D plane.



A diagram like this is required. Some text to say the View Reference Point anchors the projection plane (or locates the camera), the View Plane Normal orients it, and the View Up Vector defines the local vertical. The projection reference point determines the amount of perspective distortion (or zooming on a camera). (4 marks for the diagram being correct, one for each element, 2 marks for a coherent description).

(6 marks)

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

SECTION B

Answer any two of the four questions in this section.

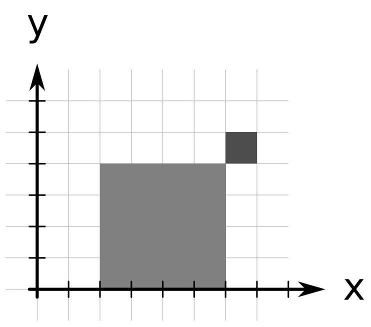
Some of the questions refer to OpenGL. An appendix giving the OpenGL API calls, can be found at the end of this paper.

2. a) Is OpenGL an **immediate** or a **retained** mode graphics package? State ONE benefit of an immediate mode graphics package AND ONE benefit of a retained mode package

Strictly OpenGL is an immediate mode graphics package (1 mark). Benefits of an immediate mode: more control, can be optimised for speed (1 mark). Benefits of retained mode: easier to make changes the application model because the scene graph (i.e. primitives) is retained in memory (1 mark).

(3 marks)

b) Write a **`renderScene()`** function which draws a 2D blue square of size 4 by 4 in the (x,y) plane, the lower left corner being positioned at (2,0), with a 1 by 1 green square positioned with lower left corner at (6,4). *Extra credit will be given for using good programming practices, such as employing functions to remove duplication in the code.*



(5 marks) for correct code drawing the square. They can use glVertex or the Vertex class from the labs.

```
// Draw a unit square
private void drawUnitSquare(void)
{
  glBegin(GL_POLYGON);
    glVertex2i(0,0);   // Or: new Vertex(0f,0f,0f).submit();
    glVertex2i(1,0);
    glVertex2i(1,1);
    glVertex2i(0,1);
  glEnd();
}
```

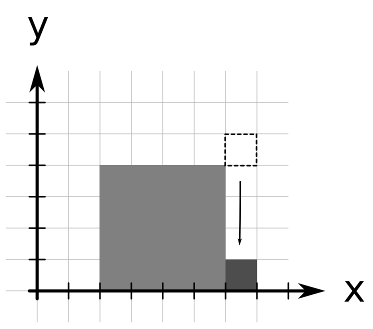
Colours can be specified using the float (0.0 to 1.0) or int (0 to 255) versions of the GL command. They can also use the Colour class from the labs. Use of scaling and translation (4 marks), push and pop (2 marks), colours (2 marks)

```
// Function to be called to determine what is displayed.
private void renderScene()
{
  glColor3f(0f,0f,1f);   // Blue
  glPushMatrix();
    glTranslatef(2f,0f,0f);
    glScalef(4f,4f,0f);
    drawSquare();
  glPopMatrix();

  glPushMatrix();
    glColor3f(0f,1f,0f); // Green
    glTranslatef(6f,4f,0f);
    drawSquare();
  glPopMatrix();
}
```

(13 marks)

c) We now want to make the smaller green square from Question 2.b) slide down the side of the larger blue square and stop when the bottoms of the 2 squares are aligned. Explain the changes required to the **renderScene** method and provide the code for the **updateScene** method.



*The **renderScene()** method becomes (2 marks):*

```
// Function to be called to determine what is displayed.
private void renderScene()
{
  glColor3f(0f,0f,1f);   // Blue
  glPushMatrix();
    glTranslatef(2f,0f,0f);
    glScalef(4f,4f,0f);
    drawSquare();
  glPopMatrix();

  glPushMatrix();
    glColor3f(0f,1f,0f); // Green
    glTranslatef(6f,offset,0f);
    drawSquare();
  glPopMatrix();
}
```

*This needs a class variable, **offset**, initialised to **4f**. (1 mark)*

*The **updateScene()** method would be:*

```
private void updateScene(void)
{
  if (offset>0) {
    offset -= 0.01f;  // Or other amount
  }
}
```

(5 marks)

(8 marks)

d) Indicate, with an explanation, which transformations are required to perform a **clockwise** 2D rotation by 45 degrees of a vertex A about another vertex P = (1,1).

1 mark for correct transformation + 1 mark for explanation:

1. *translation T(-1,-1) to move the scene so that P coincides with the world origin*
2. *rotation R(-45), a negative angle corresponds to a clockwise rotation*
3. *translation T(1, 1) to move the scene back to its original position*

(6 marks)

Total marks for Question 2: 30 marks

3. a) List THREE **shading models** for determining the appearance of polygon

meshes AND describe briefly the way each works.

Flat shading: *a single light intensity value is computed for each polygon, based on the angle between the polygon's normal and the light source*
Gouraud shading: *the light intensity is computed at each vertex and interpolated in between*
Phong shading: *the surface normals are interpolated at each pixel and the light intensity is computed based on the pixel normal*

(6 marks)

b) Using the camera analogy, explain how **viewing** is controlled in OpenGL **and** provide a simple code statement showing how the camera would be set up using the relevant function from the GLU library.

A given view of a scene is defined by 3 elements (1 mark each):

(a) the position of the camera,

(b) the point the camera is aiming at,

(c) the up direction of the camera.

In OpenGL, this can be set using the **`gluLookAt`** *command (1 mark), which takes 9 parameters (3 sets of triplets) corresponding to 2 points (position and target) and a vector (aligned with the 'up' direction) (1 marks for correctly explaining the parameters, including the order):*

```
    gluLookAt( xpos, ypos, zpos,
               xaim, yaim, zaim,
               xup,  yup,  zup );
```

(1 mark for code)

(6 marks)

c) Give THREE types of **lighting** used in OpenGL AND explain their physical interpretation.

1 mark for each model + 1 mark for light aspect + 2 marks for reflection/intensity:
ambient light: *Light which is reflected in all directions a great many times (1 mark) and has the same intensity everywhere (1 mark). Provides a "minimum" amount of constant light to the scene (1 mark).*

diffuse light: *light which is scattered equally in all directions (1 mark) and has varying intensity depending on the angle between the lit surface normal and the light source (1 mark). Is responsible for the depth effect (1 mark).*
specular light: *light which is reflected in a limited direction (1 mark) and which intensity decreases as the angle between the reflected ray and the viewer increases (1 marks). Provides the "shiny" spot on objects (1 mark).*

(6 marks)

d) Given a fully specified light source, indicate which other THREE pieces of information affect the perceived intensity of a lit object.

Material properties, polygon normals (i.e. orientation – for diffuse and specular) and viewpoint (for specular). 2 marks each.

(6 marks)

e) Briefly describe TWO methods that can be used to speed up the z-buffer algorithm for visible surface determination, and explain how they achieve this speed up.

Back face culling *removes all faces on objects pointing away from the viewer (1 mark) by checking the sign of the z-component of the face normal in view reference coordinates (1 mark) which halves the number of faces that need to be considered in the algorithm (1 mark).*
Pre-sorting the faces *in distance from the viewer (1 mark). Order the faces using the z-coordinate in the view reference system and process from the front backwards (1 mark) so that the lighting calculations are only ever carried out for the faces that are finally found to be visible (1 mark).*

(6 marks)

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

---

4. a) Show how a simple 2D translation of (2,-2) can be represented as a transformation matrix. Apply this transformation matrix to the line (0,3), (4,4),

and give the resulting vertices in homogeneous coordinates.

The matrix is: $\begin{bmatrix}1 & 0 & 2\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}$ *(2 marks). The new points are: (2,1,1), (6,2,1) (1 mark) but I want to see that they have used the matrix for full marks (i.e.* $\begin{bmatrix}1 & 0 & 2\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\3\\1\end{bmatrix} = \begin{bmatrix}2\\1\\1\end{bmatrix}$ *) (2 marks)*

(5 marks)

b) Using a transformation matrix, show how the line resulting from Question 4.a), can be scaled by a non-uniform scaling of 2 in the x-direction, and 0.5 in the y-direction.

Scaling matrix: $\begin{bmatrix}2 & 0 & 0\\0 & 0.5 & 0\\0 & 0 & 1\end{bmatrix}$. *(2 marks). New vertices (4,0.5,1), (12,1,1) (2 marks with working, 1 mark without).*

(4 marks)

c) Show the composite transformation matrix for the translation in (a) followed by the scaling in (b), and show, with workings, the inverse of this composite transformation matrix.

Composite matrix: $\begin{bmatrix}2 & 0 & 0\\0 & 0.5 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 2\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}2 & 0 & 4\\0 & 0.5 & -1\\0 & 0 & 1\end{bmatrix}$ *(3 marks).*

For inverse use $(AB)^{-1} = B^{-1} \times A^{-1}$ *(1 mark) and is:*

$$\begin{bmatrix}1 & 0 & -2\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0.5 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0.5 & 0 & -2\\0 & 2 & 2\\0 & 0 & 1\end{bmatrix}$$

(3 marks – one for each correct inverse).

(7 marks)

d) Explain the role of the **matrix stack** in OpenGL AND explain how it could be used to construct hierarchical models of the objects we wish to display.

Matrix stack: 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, inside the push – pop pair (2 marks). Localising the effect of transformations in a controllable manner means that hierarchical objects can be constructed with transformations being applied to any given level depending on the locations of the push-pop pairings (2 marks). (A sketch could also be used to show this – basically a scene graph.)

(6 marks)

e) Consider a scene where a pulsing (i.e. the size getting larger and smaller) minor star orbits a larger stable star located at a given position in the virtual universe. Draw a scene graph for the scene, including any necessary transformations (no actual values are required for the transformations, but you should explain briefly the purpose of each transformation).

Pulsing could be mimicked using variable scaling (1 mark), while rotation about a fixed point requires both rotation and translation. (2 mark) Overall correct scene graph (2 mark).

```
scene origin
 +-- T[origin of large star]
     S[size of large star]
     Large star
      +-- Ry[angle about large star]
          T[to small star]
          S[variable size]
          Small star
```

(5 marks)

f) Show, using any suitable notation how the scene graph from Question 4.e) would be implemented in OpenGL. In the transformation calls, do not specify values but put a suitable comment instead:

```
glRotatef(/* something in some way */);
```

You can assume a `drawSphere()` method is available to draw a unit sphere at (0,0,0).

The scene graph from the previous question would translate to something like this:

```
glPushMatrix();
  glTranslatef(/* final star x,y,z coordinates */);
  glScalef(/* x,y,z size of the large central star */);
  drawSphere();
  glPushMatrix();
    glRotatef(/* amount for a given time, about correct axis
    glTranslatef(/* the smaller outer star to the correct
                    distance from the inner star */)
    glScalef(/* by a variable factor to mimic the pulsing */)
    drawSphere();
  glPopMatrix();
glPopMatrix();
```

Good use of push/popMatrix (1 mark) and correct implementation of the scene graph (2 marks).

(3 marks)

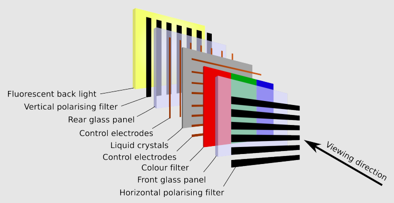
Total marks for Question 4: ***30 marks***

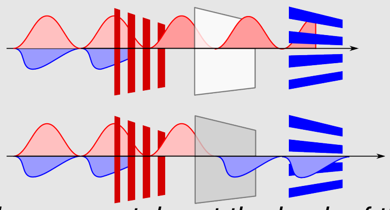
5. a) What is the main difference between outputting a colour image to a display device and a hardcopy device?

On a display device light is used to create the colour, requiring an additive (RGB) colour model (1 mark). On hardcopy devices inks are used which use a subtractive colour model, CMYK (1 mark) because inks change reflection / absorption, not emission (1 mark).

(3 marks)

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

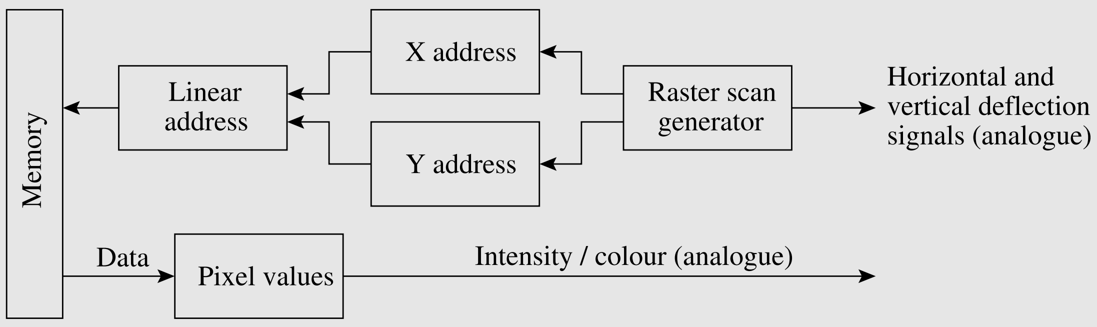




In a liquid crystal display, light (usually provided by a neon tube at the back of the display) needs to go through 2 orthogonal polarizing filters. By default, these filters don't let the light through. Liquid crystals, placed in between the fiters, can change the polarisation of light, and control 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)

c) Show, using a sketch with explanation, the architecture of the video controller that drives the input signal for an analogue display device and describe its operation.


Memory
Linear address
X address
Y address
Raster scan generator
Horizontal and vertical deflection signals (analogue)
Data
Pixel values
Intensity / colour (analogue)

The raster scan generator sends the correct offsets to the CRT deflection plates (or TFT matrix), and produces a memory address, which is converted to address space, and then used to retrieve the corresponding pixels RGB values. Appropriate signals are then sent to set the RGB levels in the output device. (4 marks for sketch, 2 for description)

(6 marks)

d) Show, using a sketch with explanation, the architecture of a typical computer with a dedicated display processor (GPU) and discuss TWO advantages of this architecture over a system without a dedicated display processor.

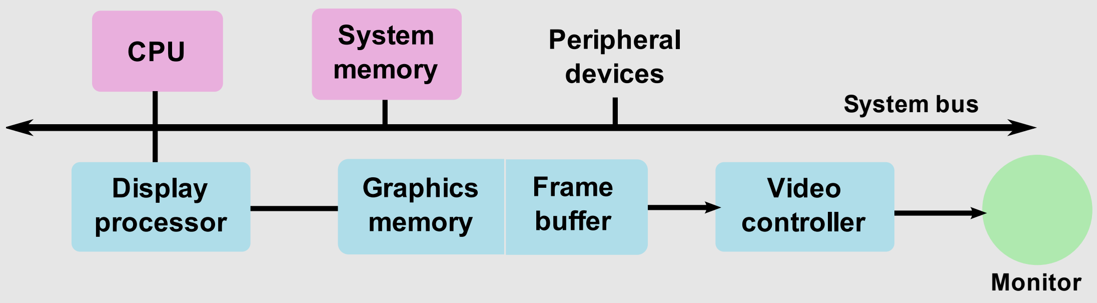


Diagram (6 marks). The advantage of a dedicated display processor is that it can access video memory without recourse to the system bus, which is often a bottle-neck, and can remove all graphics computations from the main CPU (2 marks).

(8 marks)

e) Explain how the **pixel-precision** and **object-precision** algorithms for visible surface determination work. Explain why the pixel-precision algorithm is usually more efficient and give an example of a particular situation where object-precision would be more efficient.

Pixel-precision: *trace a ray from the viewer (projection centre) through each pixel of the display and determine which object is hit first (closest). Assign the colour of the object to the pixel. (2 marks)*
Object-precision: *determine the parts of each object which are not occluded (by itself or other objects). Draw these parts in the object's colour. (2 marks)*
Pixel-precision is usually faster, as object-precision involves more complex steps (1 mark). In the case where the resolution of the display changed often, however, object-precision would only require the display step (the determined visible surface being still valid) while pixel-precision would need a complete rerun of the algorithm (given the new set of pixels). (1 mark)

(6 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