# RI3004A  3D Graphics Rendering

## Discussion 2  (Answers)

For **Lecture 3** & **Lecture 4**

---

Please attempt the following questions before you go to your discussion class. Some of the questions may be quite open-ended and some may be even ambiguous. In those cases, you are encouraged to make your own (reasonable) assumptions.

(1) What is the use of the GLUT function `glutPostRedisplay()`?

The execution of the `glutPostRedisplay()` function tells GLUT to call the display callback function in the next cycle of the event loop. When do we want to call the `glutPostRedisplay()` function? When we explicitly want the rendered image to be updated.

(2) How does double buffering work? Why do we use it?

Double-buffering makes use of two color buffers—a front buffer and a back buffer. The back buffer is where all the rendering operations are applied to, and the front buffer is the one that is being displayed on the display device. The two buffers can be easily and quickly switched.

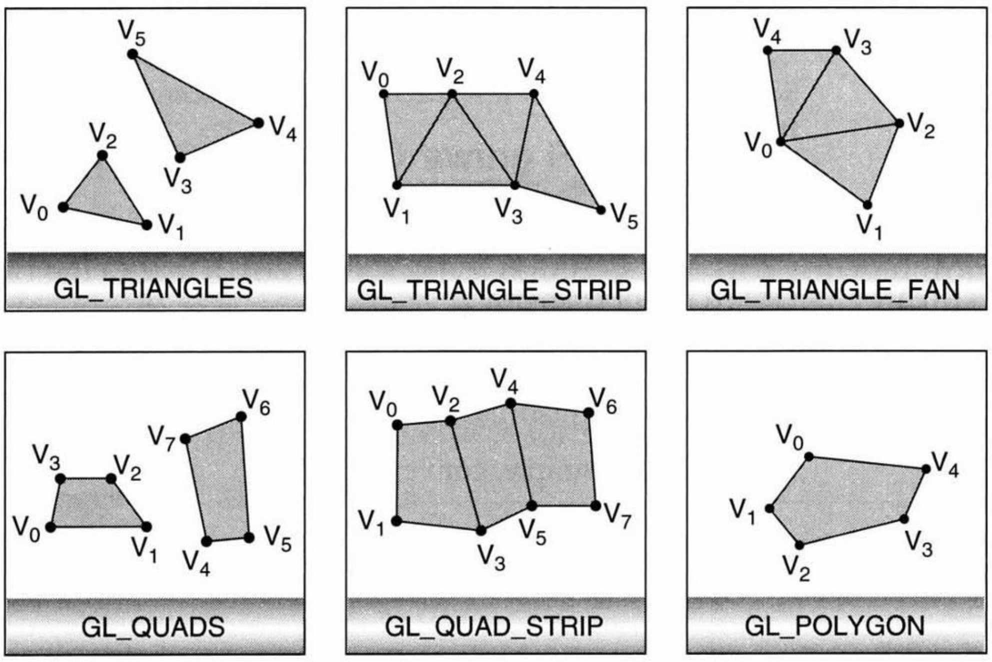
Double-buffering prevents partially rendered frame from being displayed on the display device. This reduces the flickering effect during animation.

(3) **(A)** What is an OpenGL viewport? **(B)** How do you specify one? **(C)** Can we have multiple viewports in a window? **(D)** Can a viewport be larger than the window? **(E)** If yes, what will happen? **(F)** When you use `glClear(GL_COLOR_BUFFER_BIT)`, are you clearing the entire window or just the viewport?

**(A)** An OpenGL viewport defines a rectangular region of the window in which OpenGL can draw.
**(B)** We specify one by using the function `glViewport( x, y, w, h )`, where `x` and `y` specify the window location (in pixels) of the lower left corner of the viewport, and `w` and `h` specify the width and height of the viewport in pixels.
**(C)** Yes. They can even overlap each other.
**(D)** Yes.
**(E)** In this case, what is supposed to be in the viewport but is outside the window region will not appear.
**(F)** It clears the entire window.

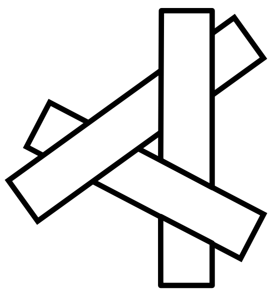
(4) OpenGL supports the `GL_TRIANGLES` primitive type. Why do you think that OpenGL also supports `GL_TRIANGLE_FAN` and `GL_TRIANGLE_STRIP`?

Usually, a surface is defined using many connected triangles (a triangle mesh). Each of these triangles **share vertices** with its neighbor triangles, and `GL_TRIANGLE_FAN` and `GL_TRIANGLE_STRIP` allow us to send much fewer vertices to the rendering pipeline to draw the surface. For example, to draw a disc made of 20 triangles, `GL_TRIANGLES` requires 20*3 = 60 vertices to be sent, whereas `GL_TRIANGLE_FAN` requires only 22 vertices to be sent. This saves system bus bandwidth and reduces the amount of vertex processing on the graphics hardware.


$V_0$ $V_1$ $V_2$ $V_3$ $V_4$ $V_5$
GL_TRIANGLES
$V_0$ $V_1$ $V_2$ $V_3$ $V_4$ $V_5$
GL_TRIANGLE_STRIP
$V_0$ $V_1$ $V_2$ $V_3$ $V_4$
GL_TRIANGLE_FAN
$V_0$ $V_1$ $V_2$ $V_3$ $V_4$ $V_5$ $V_6$ $V_7$
GL_QUADS
$V_0$ $V_1$ $V_2$ $V_3$ $V_4$ $V_5$ $V_6$ $V_7$
GL_QUAD_STRIP
$V_0$ $V_1$ $V_2$ $V_3$ $V_4$
GL_POLYGON

(5) Hidden surface removal is not necessary if we can sort the polygons in a back-to-front order and render these polygons in that order. Is it always possible that any set of polygons can be sorted in a back-to-front order? Show examples.

It is not always possible to be able to sort a set of polygons in a back-to-front order. The following diagram shows one example, in which the three polygons occlude one another cyclically. One way out of this problem is to split the original polygons.



(6) Devise a test to check whether a polygon in 3D space is planar.

First, we must ask how the polygon is specified or defined. Usually, the input is just the number of sides $N$ of the polygon, and the 3D coordinates of the vertices, $v_1$, $v_2$, ..., $v_N$. We assume that any three consecutive vertices are not collinear. To check that the polygon is planar, we find the normal vectors of the triangle $v_1$, $v_2$ and $v_3$, the triangle $v_3$, $v_4$ and $v_5$, and so on. The polygon is planar iff all the normal vectors are parallel.

(7) Devise a test to check whether a polygon on the $x$-$y$ plane is convex.

First, compute the normal vectors of the triangle $v_1$, $v_2$ and $v_3$, the triangle $v_2$, $v_3$ and $v_4$, and so on until triangle $v_{N-2}$, $v_{N-1}$, $v_N$. The polygon is convex iff all the normal vectors are in the same direction. Note that the normal vectors are 3D.

(8) Why does OpenGL (and many other graphics APIs) use homogeneous coordinates to represent points?

1. To have different representations to distinguish between points and directions (e.g. for specifying the "positions" of directional light sources).
2. To do translation using matrix multiplication, just like doing other linear transformations.
3. To allow perspective projection using matrix .multiplication (and a perspective division).

(9) Given a point $P$, whose coordinates are $\mathbf{P}$ expressed in the world coordinate frame, find the coordinates of $P$ expressed in the coordinate frame ($\mathbf{Q}$, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$), where $\mathbf{Q}$ is a point in the world coordinate frame, and $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ are unit vectors in the world coordinate frame. You may write your answer as multiplication of matrices and vectors.

$$\mathbf{P}' = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -q_x \\ 0 & 1 & 0 & -q_y \\ 0 & 0 & 1 & -q_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{P}$$

(10) For the following code fragment, write down the transformations that are actually applied to each vertex `vi`.

```
glMatrixMode( GL_MODELVIEW );
glLoadMatrixd( A );
glPushMatrix();
   glBegin( GL_POINTS ); glVertex3dv( v1 ); glEnd();
   glMultMatrixd( B );
   glPushMatrix();
      glMultMatrixd( C );
      glBegin( GL_POINTS ); glVertex3dv( v2 ); glEnd();
   glPopMatrix();
   glBegin( GL_POINTS ); glVertex3dv( v3 ); glEnd();
   glPushMatrix();
      glMultMatrixd( D );
      glPushMatrix();
         glMultMatrixd( E );
         glBegin( GL_POINTS ); glVertex3dv( v4 ); glEnd();
      glPopMatrix();
      glBegin( GL_POINTS ); glVertex3dv( v5 ); glEnd();
   glPopMatrix();
glPopMatrix();
glMultMatrixd( F );
glPushMatrix();
   glMultMatrixd( G );
glPopMatrix();
glBegin( GL_POINTS ); glVertex3dv( v6 ); glEnd();
```

A·v1
A·B·C·v2
A·B·v3
A·B·D·E·v4
A·B·D·v5
A·F·v6

(11) What is the perpendicular distance of the point $Q$ from the plane $ax + by + cz + d = 0$?

Let the coordinates of $Q$ be $[x_q,\ y_q,\ z_q]^T$. Then its distance from the plane is $\left|\dfrac{ax_q + by_q + cz_q + d}{\sqrt{a^2 + b^2 + c^2}}\right|$.

**——— End of Document ———**