RI3004A 3D Graphics Rendering

Discussion 1 (Answers)

For **Lecture 1** & **Lecture 2**

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) To be able to display realistic images, our display devices need to be able to produce every frequency in the visible light spectrum. True or false? Why? What are the advantages and disadvantages?

False.

The human retina has three types of cones (color-sensitive cells), and they are sensitive to 3 different and overlapping parts of the visible light spectrum. The ‘red’ cone is sensitive to frequencies around the red frequency, and likewise for the ‘green’ and ‘blue’ cones. When light enters the eye, at each location on the retina, only 3 values (tristimulus values) are sent to the brain, and the relative proportions of the three values define the sensation of a color. It is possible that a single-frequency light and a multiple-frequency light can excite the three cones in the same way. In this case, the eye can see the same color and cannot distinguish between the two. For example, a mixture of pure red light and pure green light can excite the cones in the same way as a pure yellow light does, and produce the same sensation of “yellow”.

The advantage is that we do not need to build display devices that are able to produce every frequency in the visible spectrum. Another advantage is that we can see colors that cannot be produced by any single-frequency light, for example, purple color. The disadvantage is that many different combinations of many different frequencies can produce the same color and we cannot distinguish between them (but, is this really a disadvantage?).

However, note that the combinations of the red, green, and blue primaries cannot produce all the colors that can be sensed by human eyes. This is too advanced to discuss.

(2) Each pixel in a frame-buffer has 8 bits for each of the R, G and B channels. How many different colors can each pixel represent? Is this enough? On some systems, each pixel has only 8 bits (for all R, G, and B combined). How would you allocate the bits to the R, G and B primaries?

$2^{8+8+8} = 2^{24}$ = 16,777,216 different colors.

Sometimes this is not enough. For example, only 256 shades of gray can be represented, and if we have these 256 shades displayed across the screen, we can still see the banding effects.

Most systems allocate 3:3:2 for R:G:B because our eyes are actually less sensitive to the changes in the blue component.

(3) Referring to Lecture 1 Slide 35. If an imaginary image plane is *d* unit distance in front of the pinhole camera, what are the coordinates of the projection (on the imaginary image plane) of the 3D point (x, y, z)? If the camera's center of projection is not located at the origin, and the camera is pointed in an arbitrary position, the calculation of the projection becomes very messy. How would you make it less messy?

$x_p = d\,x / z, \quad y_p = d\,y / z, \quad z_p = d.$

The camera has its own coordinate frame, where the coordinates of the image point are expressed in. The world coordinate frame is where the 3D point's coordinates are expressed in. In the example on the slide, the two coordinate frames coincide exactly. This makes it easy to calculate the coordinates of the projection of the 3D point. If the camera's center of projection is not located at the origin, and it is pointed in an arbitrary position, that means the two coordinate frames are not aligned. To make projection calculation easy, we would first do a 3D rotation and translation to align the two coordinate frames. In other words, we first do a 3D transformation to express the coordinates of the 3D point with respect to the camera coordinate frame.

(4) Why do we need a primitive assembly stage in the rendering pipeline architecture?

Each primitive are defined by its vertices only. These vertices are processed independently in the vertex processor without regards to the type of the primitive that they define. At the primitive assembly stage, we must recall the primitive type and collect its processed vertices, so that we can redefine the primitive for clipping and rasterization.

(5) What is a GLUT display callback function?

A GLUT display callback function is a user-defined function registered using `glutDisplayFunc()`. This user-defined function will be executed whenever GLUT determines that the window should be refreshed.

(6) Which of the two following program fragments is more efficient? Why? Can the same optimization be done for the case of GL_POLYGON?

A	**B**

`double v[3*N][3];`
`...`
`for ( int i = 0; i < 3*N; i+=3 )`
`{`
`   glBegin(GL_TRIANGLE);`
`      glVertex3dv( v[i] );`
`      glVertex3dv( v[i+1] );`
`      glVertex3dv( v[i+2] );`
`   glEnd();`
`}`
	`double v[3*N][3];`
	`...`
	`glBegin(GL_TRIANGLE);`
	`   for ( int i = 0; i < 3*N; i+=3 )`
	`   {`
	`      glVertex3dv( v[i] );`
	`      glVertex3dv( v[i+1] );`
	`      glVertex3dv( v[i+2] );`
	`   }`
	`glEnd();`

B is more efficient. The executions of multiple glBegin() and glEnd() add extra execution overhead, which may be a concern for applications that strives for very high frame rates.

The same optimization cannot be used for GL_POLYGON because each glEnd() is used to indicate the end of the list of vertices of the polygon, and this indirectly specify the number of vertices the polygon has.

(7) Assume we have the following OpenGL function calls:

```
glViewport( u, v, w, h );
gluOrtho2D( x_min, x_max, y_min, y_max );
```

Find the mathematical equations that map a point (x, y) that lies within the clipping rectangle to a point (x_s, y_s) that lies within the viewport.

We can solve this problem separately in the x and y directions. The transformation is linear, that is $x_s = ax + b$, $y_s = cy + d$. We must maintain proportions, so that x_s in the same relative position in the viewport as x is in the window, hence

$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{x_s - u}{w} \quad \Rightarrow \quad x_s = u + w\frac{x - x_{min}}{x_{max} - x_{min}}$$

Likewise $$y_s = v + h\frac{y - y_{min}}{y_{max} - y_{min}}$$

(8) What is hidden-surface removal? When is it not necessary?

Hidden surface removal is the determination of what part of the 3D object should appear in the rendered image and what part should not appear because it is occluded.

Hidden surface removal is not necessary when there is only one single planar object in the space, or when the primitives are already sent to the rendering pipeline in a back to front order (painter's algorithm).

——— End of Document ———