CSCI 550 (Fall 2016)

Assignment #1

Handout: Wednesday, August 31, 2016
Due: 11:59pm, Wednesday, Sept. 28 2016

All assignments will be submitted through Canvas.
The non-programming part can be in any format such as Microsoft Word, PDF, plain text or scanned hand-writing.
The programming part should include: (1) all source code files; (2) the executable file (.exe); and (3) a README file containing necessary instructions, known bugs, and any other notes you would like me to read.

Part 1: non-programming problems (50 pts)

1. What is a "client-server" relationship? Give a non-graphics example of a common client-server relationship.

2. What is the difference between calling glutPostRedisplay and calling the display function directly? Which one is preferred? Explain why.

3. Suppose an RGB raster system is to be designed using a 20-in by 15-in screen with a resolution of 100 pixels per inch in each direction. If we want to store 24 bits per pixel in the frame buffer:

   (a) How much storage (in bytes) do we need for the frame buffer?

   (b) How many distinct colors can be represented using this frame buffer? How many different colors can be displayed in one frame buffer (i.e. in one image frame)?

   (c) If a 120 Hz frame refreshing rate is required, what is the minimum memory access speed (bytes per second) for the frame buffer?

4. Show two different ways of defining a cylinder surface using GL_TRIANGLES and GL_TRIANGLE_STRIP primitives, respectively (show your solution in segments of OpenGL commands, but do not write a full program).

5. Draw the graphical output if the following points were given, in the that order, to the primitives (1) GL_LINES, (2) GL_LINE_STRIPs, (3) GL_TRIANGLES, and (4) GL_TRIANGLE_STRIP:

   \[ P_1 = (0, 0); P_2 = (0, 1); P_3 = (1, 1); P_4 = (1, 0); P_5 = (2, 0); P_6 = (2, 1) \]
Part 2: Programming problem: Interactive Bezier Curve Drawing (100 pts)

Bezier curve is an important modeling tool in curve/surface design. It is defined by a sequence of connected line segments called control polyline, as shown in Figure 1(a). To draw a Bezier curve, a subdivision algorithm can be used to recursively refine the control polyline to generate progressive linear approximations to the curve.

Let $P^0_i$ ($i = 0 \cdots n - 1$) be the original vertices of the control polyline. The following procedure generates one subdivision:

1. Define $P^1_i$ ($i = 1 \cdots n - 1$) as the mid-points of all line segments in the control polyline, i.e. $P^1_i = (P^0_{i-1} + P^0_i)/2$, ($i = 1 \cdots n - 1$).

2. Similarly, define $P^2_i$ ($i = 2 \cdots n - 1$) as the mid-points of all new line segments formed by $P^1_i$, i.e. $P^2_i = (P^1_{i-1} + P^1_i)/2$, ($i = 2 \cdots n - 1$).

3. Continue doing the above for each newly formed polyline, i.e. $P^k_i = (P^{k-1}_{i-1} + P^{k-1}_i)/2$, where $k = 1 \cdots n - 1$, and $i = k \cdots n - 1$. When $k = n - 1$, there is only one point left, the process is then complete.

After this subdivision, the original control polyline is divided into two separate control polylines: $P^i_i$ ($i = 0 \cdots n - 1$), and $P^i_{n-1}$ ($i = n - 1 \cdots 0$), as shown in Figure 1(b). Each of these two polylines represents half of the curve. But they are now closer to the curve than the original polyline. We call the original polyline the level-0 approximation, and the two refined polylines the level-1 approximation. Naturally, the above procedure can be applied to each of the level-1 polylines to generate four new (more refined) polylines for level-2 approximation to the curve. The process can continue for the level-2 polylines and all subsequent higher level polylines to generate level-3, level-4, ... polylines.

In this project, you will write an interactive program for the definition and drawing of Bezier curve. Your program should be able to:

![Figure 1: The Bezier curve subdivision algorithm](image)
1. Interactively define the initial control polyline, i.e. user will use the mouse to click points on the screen to define the vertices of the polyline, and the program should draw the current polyline every time a vertex is defined. You are also required to use the motion callback function to generate “rubberbanding” effect for polyline definition.

2. Receiving a character from the keyboard as a keyboard event. The character invokes the following actions:

   • “q” or “Q”: exit the program.
   • a number \(i, (i = 1 \cdot \cdot \cdot 9)\): draw the level-\(i\) polylines generated from the above procedure.
   • “+” or “-”: increase or decrease the current level of approximation by 1, and draw the new polylines.
   • “p” or “P”: toggle the drawing of the original (level-0) control polyline. By default, the level-0 polyline is always drawn along with the curve (actually the current approximation of the curve). In this state, user can always add a vertex to the end of the original polyline by clicking on the screen (the polyline and curve should be redrawn immediately). By press “p” or “P”, the original polyline will disappear from the screen, only the curve is drawn now. In this state, clicking on the screen will not generate new vertex. Pressing “p” or “P” again will return the program to the original state (the original polyline should be drawn again).

3. Generate a menu for all of the above keyboard functionalities.