Computer Graphics

(Comp 4190.410)

Midterm Exam: October 30, 2013

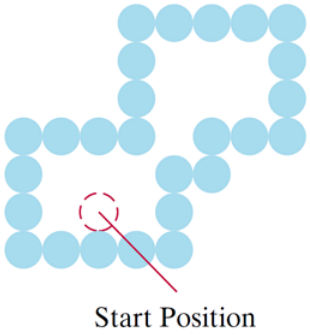
1. (20 points)

   (a) (4 points) What are the main tasks of the display processor?

   - Rasterization (Scan conversion of points, lines, polygons, characters, etc.)
   - Various geometric and viewing transformations
   - Interface with interactive input devices

   (b) (4 points) How many times the recursive function `boundaryFill4` will be called for the following example?

   - 25 times




```
void boundaryFill4 (int x, int y, int fillColor, int borderColor)
{
   int interiorColor;

   /* Set current color to fillColor, then perform following oprations. */
   getPixel (x, y, interiorColor);
   if ((interiorColor != borderColor) && (interiorColor != fillColor)) {
      setPixel (x, y);    // Set color of pixel to fillColor.
      boundaryFill4 (x + 1, y , fillColor, borderColor);
      boundaryFill4 (x - 1, y , fillColor, borderColor);
      boundaryFill4 (x , y + 1, fillColor, borderColor);
      boundaryFill4 (x , y - 1, fillColor, borderColor)
   }
}
```

   (c) (4 points) What are the main reasons for mapping the clipping window to a normalized viewport?

   - Efficiency in supporting multiple viewports on different devices
   - Simple clipping

   (d) (8 points) Is it possible to extend the Cohen-Sutherland line clipping algorithm to cubic Bézier curves which are monotone along the x- and y-axis directions? Explain why. Is it possible to extend the NLN algorithm? Explain why.

   - Yes. The Cohen-Sutherland algorithm is mainly based on the x- and y-monotoicity of line segments. When the two end points share at least one non-zero bit in their binary codes, the whole curve is guaranteed to be in the same region outside the clipping window.
   - No. These curves may not be monotone along arbitrary directions, whereas the regions of the NLN algorithm are bounded by lines $ax + by + c = 0$ with arbitrary slope, i.e., $a \neq 0$ and $b \neq 0$ in general.

2. (20 points) Consider a trackball of radius 1 with its center located at the origin $(0, 0, 0)$.

   (a) (5 points) When we rotate the trackball by moving a point at $\mathbf{a} \in S^2$ to a different location $\mathbf{b} \in S^2$, find the axis and angle of the 3D rotation.

   - Axis: $\mathbf{a} \times \mathbf{b} / \|\mathbf{a} \times \mathbf{b}\|$
   - Angle: $\arccos \langle \mathbf{a}, \mathbf{b} \rangle$

   (b) (5 points) We put a sensor to the housing of the trackball at the bottom location $(0, 0, -1)$ where the sensor can detect a 2D surface velocity of the rotating trackball. Given an angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$, what is the velocity the sensor at $(0, 0, -1)$ can detect?

   - $\omega \times (0, 0, -1) = (-\omega_y, -\omega_x, 0)$

   (c) (10 points) Given two sensors located at $\mathbf{p}_i = (x_i, y_i, z_i) \in S^2$, $i = 1, 2$, each detects a surface velocity $\mathbf{p}'_i = (x'_i, y'_i, z'_i)$ which is orthogonal to $\mathbf{p}_i$, i.e., $\langle \mathbf{p}_i, \mathbf{p}'_i \rangle = 0$. Using the relation $\mathbf{p}'_i = \omega \times \mathbf{p}_i$, formulate a matrix equation $A\omega = \mathbf{b}$, where $\mathbf{b} = (x'_1, y'_1, z'_1, x'_2, y'_2, z'_2)^T$, and discuss how to approximate the angular velocity ω from the overconstrained equation.

$$\begin{bmatrix} 0 & z_1 & -y_1 \\ -z_1 & 0 & x_1 \\ y_1 & -x_1 & 0 \\ 0 & z_2 & -y_2 \\ -z_2 & 0 & x_2 \\ y_2 & -x_2 & 0 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ z'_1 \\ x'_2 \\ y'_2 \\ z'_2 \end{bmatrix}$$

3. (20 points) Consider a perspective projection of 3D points $\mathbf{x}_i$ (from $\hat{\mathbf{v}}$) to 2D points $\mathbf{x}'_i$ (on $\hat{\mathbf{n}}$):

$$\hat{\mathbf{x}}'_i = \langle \hat{\mathbf{n}}, \hat{\mathbf{v}} \rangle \, \hat{\mathbf{x}}_i - \hat{\mathbf{v}} \, \langle \hat{\mathbf{n}}, \hat{\mathbf{x}}_i \rangle = P\hat{\mathbf{x}}_i, \quad \text{for i} = 1, 2.$$

   (a) (10 points) Explain why the projection $\mathbf{x}'_m$ of the midpoint $\mathbf{x}_m = (\mathbf{x}_1 + \mathbf{x}_2)/2$ is not the same as the midpoint $(\mathbf{x}'_1 + \mathbf{x}'_2)/2$ of the projections $\mathbf{x}'_1$ and $\mathbf{x}'_2$ onto the plane $\hat{\mathbf{n}}$.

   - Let $\hat{\mathbf{x}}_1 = [\mathbf{x}_1, 1]^t$ and $\hat{\mathbf{x}}_2 = [\mathbf{x}_2, 1]^t$, then
     $\hat{\mathbf{x}}'_1 = P\hat{\mathbf{x}}_1 = [w'_1\mathbf{x}'_1, w'_1]^t$ and $\hat{\mathbf{x}}'_2 = P\hat{\mathbf{x}}_2 = [w'_2\mathbf{x}'_2, w'_2]^t$, where $w'_1 \neq w'_2$ in general.
     Let $\hat{\mathbf{x}}_m = [\mathbf{x}_m, 1]^t = [(\mathbf{x}_1 + \mathbf{x}_2)/2, 1]^t = [\mathbf{x}_1 + \mathbf{x}_2, 2]^t$, then
     $\hat{\mathbf{x}}'_m = P\hat{\mathbf{x}}_m = P[\mathbf{x}_1 + \mathbf{x}_2, 2]^t = P\hat{\mathbf{x}}_1 + P\hat{\mathbf{x}}_2 = \hat{\mathbf{x}}'_1 + \hat{\mathbf{x}}'_2 = [w'_1\mathbf{x}'_1 + w'_2\mathbf{x}'_2, w'_1 + w'_2]^t$.
     Consequently, $\mathbf{x}'_m = (w'_1\mathbf{x}'_1 + w'_2\mathbf{x}'_2)/(w'_1 + w'_2) \neq (\mathbf{x}'_1 + \mathbf{x}'_2)/2$ if $w'_1 \neq w'_2$.

   (b) (10 points) Let $L(t) = (1 - t)\mathbf{x}'_1 + t\mathbf{x}'_2$, $0 \leq t \leq 1$, denote the line segment connecting the two projection points $\mathbf{x}'_1$ and $\mathbf{x}'_2$. What is the preimage of $L(t)$ (on the line segment $\overline{\mathbf{x}_1\mathbf{x}_2}$ connecting the two 3D points $\mathbf{x}_1$ and $\mathbf{x}_2$) for the above perspective projection?

   - The mapping from the given line segment $\overline{\mathbf{x}_1\mathbf{x}_2}$ to the projected line segment $\overline{\mathbf{x}'_1\mathbf{x}'_2}$ can be presented as a perspective transformation

$$\begin{bmatrix} w'_2 & 0 \\ w'_2 - w'_1 & w'_1 \end{bmatrix},$$

     which has an inverse mapping

$$\begin{bmatrix} w'_1 & 0 \\ w'_1 - w'_2 & w'_2 \end{bmatrix}$$

     Thus, the preimage of $L(t)$ is given as a point $(1 - \alpha(t))\mathbf{x}_1 + \alpha(t)\mathbf{x}_2$ on the line segment $\overline{\mathbf{x}_1\mathbf{x}_2}$, where

$$\alpha(t) = \frac{tw'_1}{t(w'_1 - w'_2) + w'_2}.$$

4. (20 points)

   (a) (10 points) Design a recursive bottom-up algorithm for constructing an LSS tree for an open polygonal chain $\mathcal{C}$ that connects a sequence of points $\mathbf{p}_i = (x_i, y_i, z_i)$, for $i = 0, \cdots, 2^k$, for some $k > 0$. To make life easy, you may split each subchain in the middle into two pieces with the same number of edges.

   - At the leaf level, the LSS is a line segment connecting two adjacent points $\mathbf{p}_{i-1}$ and $\mathbf{p}_i$, for $i = 1, \cdots, 2^k$. At this leaf level, there is no approximation error and the thickness of the corresponding LSS volume is zero: $\epsilon = 0$.
   - At an intermediate level, assume that the left-child LSS is the sweeping of a sphere of radius ϵ_i along the line segment $\overline{\mathbf{p}_{(i-1)*2^h}\mathbf{p}_{i*2^h}}$ and the right -child LSS is the sweeping of a sphere of radius $\epsilon_{(i+1)}$ along the line segment $\overline{\mathbf{p}_{i*2^h}\mathbf{p}_{(i+1)*2^h}}$, for $i = 1, \cdots, 2^{(k-h)}$.
   - The parent LSS is the sweeping of a sphere of radius $d_i + \max(\epsilon_i, \epsilon_{(i+1)})$ along a line segment $\overline{\mathbf{p}_{(i-1)*2^h}\mathbf{p}_{(i+1)*2^h}}$, where d_i is the minimum distance between the point $\mathbf{p}_{i*2^h}$ and the line segment $\overline{\mathbf{p}_{(i-1)*2^h}\mathbf{p}_{(i+1)*2^h}}$.
   - The minimum distance d_i can be computed using the inner product between two vectors $\mathbf{a} = \mathbf{p}_{i*2^h} - \mathbf{p}_{(i-1)*2^h}$ and $\mathbf{b} = \mathbf{p}_{(i+1)*2^h} - \mathbf{p}_{(i-1)*2^h}$. (i) If $\langle \mathbf{a}, \mathbf{b} \rangle < 0$, $d_i = \|\mathbf{a}\|$, (ii) else if $\langle \mathbf{a}, \mathbf{b} \rangle > \|\mathbf{a}\|\|\mathbf{b}\|$, $d_i = \|\mathbf{p}_{(i+1)*2^h} - \mathbf{p}_{i*2^h}\|$, and (iii) otherwise, $d_i = \|\mathbf{a} \times \mathbf{b}\|/\|\mathbf{b}\|$.

   (b) (10 points) Design a recursive view frustum culling algorithm for the polygonal chain $\mathcal{C}$ using the LSS tree constructed in (a).

   - Given an LSS node with two end points $\mathbf{p}_k = (x_k, y_k, z_k)$, for $k = 1, 2$, and the view frustum bounded by six planes $L_i : a_i x + b_i y + c_i z + d_i = 0$, for $i = 1, \cdots, 6$, with a unit outward normal direction $(a_i, b_i, c_i) \in S^2$, we test the signed distance of the two points $\mathbf{p}_k$ against each bounding plane L_i.
   - For a plane L_i, if the two signed distances are both larger than the thickness ϵ of the LSS bounding volume, i.e., $a_i x_k + b_i y_k + c_i z_k + d_i - \epsilon > 0$, for $k = 1, 2$, the LSS volume is completely outside the view frustum and should be culled away.
   - Else if the two signed distances are both smaller than $-\epsilon$ for all the six planes L_i, i.e., $a_i x_k + b_i y_k + c_i z_k + d_i + \epsilon < 0$, for all $k = 1, 2$ and all $i = 1, \cdots, 6$, the LSS volume is completely contained in the view frustum.
   - Otherwise, we go to the next lower level of the LSS tree hierarchy and repeat the same procedure recursively.
   - Finally, at the end of the recursion, we will end up with the leaf level of the LSS tree, where we clip the line segment at the leaf node against the view frustum.

5. (20 points) Fill in the blanks in the following OpenGL program segments taken from HW #3-3.