Quiz #4 (CSE4190.410)

November 11, 2013 (Monday)

Name: ______________ **Dept:** ______________ **ID No:** ______________

1. (5 points) Consider a perspective transformation P of 3D points $\mathbf{x}_i$ to 3D points $\mathbf{x}'_i = P\mathbf{x}_i$, for $i = 1, 2, 3$. Explain why the image $\mathbf{x}'_m = P\mathbf{x}_m$ of the center $\mathbf{x}_m = (\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3)/3$ is not the same as the center $(\mathbf{x}'_1 + \mathbf{x}'_2 + \mathbf{x}'_3)/3$ of the three points $\mathbf{x}'_1$, $\mathbf{x}'_2$, $\mathbf{x}'_3$.

   - Let $\hat{\mathbf{x}}_i = [\mathbf{x}_i, 1]^t$, then $\hat{\mathbf{x}}'_i = P\hat{\mathbf{x}}_i = [w'_i\mathbf{x}'_i, w'_i]^t$, for $i = 1, 2, 3$.
     $w'_i \neq w'_j$ in general, for $i \neq j$
     Let $\hat{\mathbf{x}}_m = [\mathbf{x}_m, 1]^t = [(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_2)/3, 1]^t = [\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3, 3]^t$, then
     $\hat{\mathbf{x}}'_m = P\hat{\mathbf{x}}_m = P[\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3, 3]^t = P\hat{\mathbf{x}}_1 + P\hat{\mathbf{x}}_2 + P\hat{\mathbf{x}}_3 = \hat{\mathbf{x}}'_1 + \hat{\mathbf{x}}'_2 + \hat{\mathbf{x}}'_2$
     $= [w'_1\mathbf{x}'_1 + w'_2\mathbf{x}'_2 + w'_3\mathbf{x}'_3, w'_1 + w'_2 + w'_3]^t$.
     Consequently, $\mathbf{x}'_m = (w'_1\mathbf{x}'_1 + w'_2\mathbf{x}'_2 + w'_3\mathbf{x}'_3)/(w'_1 + w'_2 + w'_3) \neq (\mathbf{x}'_1 + \mathbf{x}'_2 + \mathbf{x}'_3)/3$ in general.

2. (15 points)

   (a) (5 points) Design a recursive bottom-up algorithm for constructing a sphere 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. We may assume a procedure that can compute the minimum-enclosing sphere of two spheres.

   - At the leaf level, the bounding volume 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: $\epsilon = 0$.
   - At each parent of the leaf level, the bounding sphere can be determined as the minimum enclosing sphere for the triangle of three vertices $\mathbf{p}_{2i-2}, \mathbf{p}_{2i-1}, \mathbf{p}_{2i}$, for $i = 1, \cdots, 2^{(k-1)}$.
   - At other intermediate levels, the bounding sphere can be constructed to be the minimum enclosing sphere for two bounding spheres of the two child nodes.

   (b) (10 points) Design a recursive top-down algorithm for constructing an LSS tree for a set of disconnected line segments $\overline{\mathbf{a}_i\mathbf{b}_i}$, for $i = 0, \cdots, 2^k$, for some $k > 0$. We may assume a procedure that can compute the minimum-enclosing LSS bounding volume for an arbitrary set of discrete points.

   - At the root level, the LSS is the minimum-enclosing LSS that bounds the whole set of points $\mathbf{a}_i, \mathbf{b}_i$, for $i = 0, \cdots, 2^k$.
   - At an intermediate level, sort the line segments along the direction $\vec{\mathbf{a}_i\mathbf{b}_i}$ using the midpoints $(\mathbf{a}_i + \mathbf{b}_i)/2$ of the line segments, and divide the edge set into two groups and compute the minimum-enclosing LSS for each group.
   - Repeat the same procedure recursively until we reach the leaf level where we end up with only one line segment.

3. (10 points) Fill in the blanks in the following OpenGL program segments taken from HW #3-4.

```
const int INIT_SIZE = 800;
int width = INIT_SIZE, height = INIT_SIZE;
int angle = 0;
enum {FRONT, UP, LEFT, ROTATE};

void SelectViewport(int view, bool clear)
{
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();	(+1)
	glOrtho(-width, width, -height, height, -100000, 100000);

	if (view == FRONT)
		gluLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0);
	else if (view == UP)
		gluLookAt(0, 1, 0, 0, 0, 0, 0, 0, -1);
	else if (view == LEFT)
		gluLookAt(1, 0, 0, 0, 0, 0, 0, 1, 0);
	else
	{
		gluLookAt(1, 1, 1, 0, 0, 0, -1, 1, -1);	(+2)
		glRotatef((float) angle, 0, 1, 0);
	}

	int w = width / 2;
	int h = height / 2;
	int x = (view == LEFT || view == ROTATE)? w : 0;	(+1)
	int y = (view == UP || view == ROTATE)? h : 0;
	glViewport(x, y, w, h);	(+1)

	if (clear)
	{
		glScissor(x, y, w, h);
		glClearColor(view < 2? 0.9f : 1, view % 2 == 0? 0.9f : 1, view > 0 && view < 3? 0.9f : 1, 1);
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);	(+1)
	}
}

void displayCallback()
{
	glEnable(GL_SCISSOR_TEST);
	for (int view = 0; view < 4; view++)
	{
		SelectViewport(view, true);	(+1)
		DrawGrid();
		DrawCurves();
		if (view != ROTATE)
			DrawControlPoints();
	}
	glDisable(GL_SCISSOR_TEST);
```

```
		glDepthMask(false);
		MinimumDistance();
		glDepthMask(true);

		glutSwapBuffers();
}

double ToLocalX(int x)
{
		return 4*x - width;
}

double ToLocalY(int y)
{
		return height - 4*y
}

void motionCallback(int x, int y)
{
		if (selectedView == FRONT)
		{
				controlPoints[selectedCurve][selectedPoint][0] = ToLocalX(x);
				controlPoints[selectedCurve][selectedPoint][1] = ToLocalY(y - height/2);
		}
		else if (selectedView == UP)
		{
				controlPoints[selectedCurve][selectedPoint][0] = ToLocalX(x);
				controlPoints[selectedCurve][selectedPoint][2] = -ToLocalY(y);
		}
		else if (selectedView == LEFT)
		{
				controlPoints[selectedCurve][selectedPoint][2] = -ToLocalX(x - width/2);		(+1)
				controlPoints[selectedCurve][selectedPoint][1] = ToLocalY(y - height/2);
		}
}

void idleCallback()
{
		if (rotate)
		{
				angle = (angle + 1) % 360;		(+1)
				glutPostRedisplay();		(+1)
		}
		Sleep(20);
}
```