CS 3353 Extra-Credit Laboratory Problem Set

Extra Credit
Due Before End of Classes

These problems are to be done on an individual basis following the Trinity University Academic Integrity Policy. or Trinity University Honor Code.

Academic Integrity and Honor Code

All students are covered by a policy that prohibits dishonesty in academic work. The Academic Integrity Policy (AIP) covers all students who entered Trinity before the Fall of 2004. The Academic Honor Code covers all those who entered the Fall of 2004 or later. The Integrity Policy and the Code share many features: each asserts that the academic community is based on honesty and trust; each contains the same violations; each provides for a procedure to determine if a violation has occurred and what the punishment will be; each provides for an appeal process. The main difference is that the faculty implements the AIP while the Honor Code is implemented by the Academic Honor Council. Under the Academic Integrity Policy, the faculty member determines whether a violation has occurred as well as the punishment for the violation (if any) within certain guidelines. Under the Honor Code, a faculty member will (or a student may) report an alleged violation to the Academic Honor Council. It is the task of the Council to investigate, adjudicate, and assign a punishment within certain guidelines if a violation has been verified. Students who are under the Honor Code are required to pledge all written work that is submitted for a grade: On my honor, I have neither given nor received any unauthorized assistance on this work and heir signature. The pledge may be abbreviated pledged with a signature.

Laboratory problems should be submitted electronically (e-mail to cs3353@ariel.cs.trinity.edu) on or before the due date and should contain a problem write-up, source code to any programs and data sets used in solving the problem. The submitted files should be ASCII text files having Unix end-of-line characters (please convert all Windows and Mac text files to Unix format–I have found that Emacs seems to do a reasonable job of such conversions). If several files need to be submitted, put them in a directory having name *your-last-name-problem-set-number* and create a tar archive of this file system and attach it to your e-mail problem submission.

A Paint Program

The course web page class-files link contains a template program, newpaint2.c, which is shown below:

```
/* This program illustrates the use of the glut library for
interfacing with a window system */


#define NULL 0
#define LINE 1
#define RECTANGLE 2
#define TRIANGLE  3
#define POINTS 4
#define TEXT 5

#include <GL/glut.h>

void mouse(int, int, int, int);
void key(unsigned char, int, int);
void display(void);
void drawSquare(int, int);
void myReshape(GLsizei, GLsizei);

void myinit(void);

void screen_box(int, int, int);
void right_menu(int);
void middle_menu(int);
void color_menu(int);
void pixel_menu(int);
void fill_menu(int);
int pick(int, int);

/* globals */

GLsizei wh = 500, ww = 500; /* initial window size */
GLfloat size = 3.0;   /* half side length of square */
int draw_mode = 0; /* drawing mode */
int rx, ry; /*raster position*/

GLfloat r = 1.0, g = 1.0, b = 1.0; /* drawing color */
int fill = 0; /* fill flag */

void drawSquare(int x, int y)
{

        y=wh-y;
        glColor3ub( (char) random()%256, (char) random()%256, (char) random()%256);
        glBegin(GL_POLYGON);
                glVertex2f(x+size, y+size);
                glVertex2f(x-size, y+size);
                glVertex2f(x-size, y-size);
                glVertex2f(x+size, y-size);
        glEnd();
}


/* rehaping routine called whenever window is resized
or moved */

void myReshape(GLsizei w, GLsizei h)
{

/* adjust clipping box */

        glMatrixMode(GL_PROJECTION);
```

```
        glLoadIdentity();
        glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

/* adjust viewport and  clear */

        glViewport(0,0,w,h);
    glClearColor (0.8, 0.8, 0.8, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        display();
        glFlush();

/* set global size for use by drawing routine */

        ww = w;
        wh = h;
}

void myinit(void)
{

glViewport(0,0,ww,wh);


/* Pick 2D clipping window to match size of X window
This choice avoids having to scale object coordinates
each time window is resized */

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, (GLdouble) ww , 0.0, (GLdouble) wh , -1.0, 1.0);

/* set clear color to black and clear window */

        glClearColor (0.8, 0.8, 0.8, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
}



void mouse(int btn, int state, int x, int y)
{
    static int count;
    int where;
    static int xp[2],yp[2];
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
    {
       glPushAttrib(GL_ALL_ATTRIB_BITS);

       where = pick(x,y);
       glColor3f(r, g, b);
       if(where != 0)
       {
          count = 0;
          draw_mode = where;
       }
       else switch(draw_mode)
       {
         case(LINE):
          if(count==0)
          {
              count++;
              xp[0] = x;
              yp[0] = y;
          }
          else
```

```
          {
              glBegin(GL_LINES);
                 glVertex2i(x,wh-y);
                 glVertex2i(xp[0],wh-yp[0]);
              glEnd();
              draw_mode=0;
              count=0;
          }
          break;
        case(RECTANGLE):
          if(count == 0)
          {
              count++;
              xp[0] = x;
              yp[0] = y;
          }
          else
          {
              if(fill) glBegin(GL_POLYGON);
              else glBegin(GL_LINE_LOOP);
                 glVertex2i(x,wh-y);
                 glVertex2i(x,wh-yp[0]);
                 glVertex2i(xp[0],wh-yp[0]);
                 glVertex2i(xp[0],wh-y);
              glEnd();
              draw_mode=0;
              count=0;
          }
          break;
        case (TRIANGLE):
          switch(count)
          {
            case(0):
              count++;
              xp[0] = x;
              yp[0] = y;
              break;
            case(1):
              count++;
              xp[1] = x;
              yp[1] = y;
              break;
            case(2):
              if(fill) glBegin(GL_POLYGON);
              else glBegin(GL_LINE_LOOP);
                 glVertex2i(xp[0],wh-yp[0]);
                 glVertex2i(xp[1],wh-yp[1]);
                 glVertex2i(x,wh-y);
              glEnd();
              draw_mode=0;
              count=0;
          }
          break;
        case(POINTS):
          {
             drawSquare(x,y);
             count++;
          }
  break;
case(TEXT):
  {
 rx=x;
 ry=wh-y;
 glRasterPos2i(rx,ry);
 count=0;
  }
       }
```

```
       glPopAttrib();
       glFlush();
     }
}

int pick(int x, int y)
{
    y = wh - y;
    if(y < wh-ww/10) return 0;
    else if(x < ww/10) return LINE;
    else if(x < ww/5) return RECTANGLE;
    else if(x < 3*ww/10) return TRIANGLE;
    else if(x < 2*ww/5) return POINTS;
else if(x < ww/2) return TEXT;
    else return 0;
}

void screen_box(int x, int y, int s )
{
    glBegin(GL_QUADS);
      glVertex2i(x, y);
      glVertex2i(x+s, y);
      glVertex2i(x+s, y+s);
      glVertex2i(x, y+s);
    glEnd();
}

void right_menu(int id)
{
   if(id == 1) exit(0);
   else display();
}

void middle_menu(int id)
{

}

void color_menu(int id)
{
   if(id == 1) {r = 1.0; g = 0.0; b = 0.0;}
   else if(id == 2) {r = 0.0; g = 1.0; b = 0.0;}
   else if(id == 3) {r = 0.0; g = 0.0; b = 1.0;}
   else if(id == 4) {r = 0.0; g = 1.0; b = 1.0;}
   else if(id == 5) {r = 1.0; g = 0.0; b = 1.0;}
   else if(id == 6) {r = 1.0; g = 1.0; b = 0.0;}
   else if(id == 7) {r = 1.0; g = 1.0; b = 1.0;}
   else if(id == 8) {r = 0.0; g = 0.0; b = 0.0;}
}


void pixel_menu(int id)
{
   if (id == 1) size = 2 * size;
   else if (size > 1) size = size/2;
}

void fill_menu(int id)
{
   if (id == 1) fill = 1;
   else fill = 0;
}

void key(unsigned char k, int xx, int yy)
{
   if(draw_mode!=TEXT) return;
glColor3f(0.0,0.0,0.0);
   glRasterPos2i(rx,ry);
```

```
   glutBitmapCharacter(GLUT_BITMAP_9_BY_15, k);
/*glutStrokeCharacter(GLUT_STROKE_ROMAN,i); */
   rx+=glutBitmapWidth(GLUT_BITMAP_9_BY_15,k);

}

void display(void)
{
int shift=0;
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glClearColor (0.8, 0.8, 0.8, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    screen_box(0,wh-ww/10,ww/10);
    glColor3f(1.0, 0.0, 0.0);
    screen_box(ww/10,wh-ww/10,ww/10);
    glColor3f(0.0, 1.0, 0.0);
    screen_box(ww/5,wh-ww/10,ww/10);
    glColor3f(0.0, 0.0, 1.0);
    screen_box(3*ww/10,wh-ww/10,ww/10);
    glColor3f(1.0, 1.0, 0.0);
    screen_box(2*ww/5,wh-ww/10,ww/10);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
       glVertex2i(wh/40,wh-ww/20);
       glVertex2i(wh/40+ww/20,wh-ww/20);
    glEnd();
    glBegin(GL_TRIANGLES);
       glVertex2i(ww/5+ww/40,wh-ww/10+ww/40);
       glVertex2i(ww/5+ww/20,wh-ww/40);
       glVertex2i(ww/5+3*ww/40,wh-ww/10+ww/40);
    glEnd();
    glPointSize(3.0);
    glBegin(GL_POINTS);
       glVertex2i(3*ww/10+ww/20, wh-ww/20);
    glEnd();
glRasterPos2i(2*ww/5,wh-ww/20);
glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'A');
shift=glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'A');
glRasterPos2i(2*ww/5+shift,wh-ww/20);
glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'B');
shift+=glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'B');
glRasterPos2i(2*ww/5+shift,wh-ww/20);
glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'C');
    glFlush();
    glPopAttrib();
}


int main(int argc, char** argv)
{
    int c_menu, p_menu, f_menu;

    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("square");
    glutDisplayFunc(display);
    c_menu = glutCreateMenu(color_menu);
    glutAddMenuEntry("Red",1);
    glutAddMenuEntry("Green",2);
    glutAddMenuEntry("Blue",3);
    glutAddMenuEntry("Cyan",4);
    glutAddMenuEntry("Magenta",5);
    glutAddMenuEntry("Yellow",6);
    glutAddMenuEntry("White",7);
    glutAddMenuEntry("Black",8);
    p_menu = glutCreateMenu(pixel_menu);
    glutAddMenuEntry("increase pixel size", 1);
```

```
    glutAddMenuEntry("decrease pixel size", 2);
    f_menu = glutCreateMenu(fill_menu);
    glutAddMenuEntry("fill on", 1);
    glutAddMenuEntry("fill off", 2);
    glutCreateMenu(right_menu);
    glutAddMenuEntry("quit",1);
    glutAddMenuEntry("clear",2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutCreateMenu(middle_menu);
    glutAddSubMenu("Colors", c_menu);
    glutAddSubMenu("Pixel Size", p_menu);
    glutAddSubMenu("Fill", f_menu);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);
    myinit ();
    glutReshapeFunc (myReshape);
glutKeyboardFunc(key);
    glutMouseFunc (mouse);
    glutMainLoop();

}
```

Extending A Paint Program

Extend the interactive paint program, described in Chapter 3 of our text, *Interactive COMPUTER GRAPHICS, A Top-Down Approach with OpenGL* by attempting to remove as many problems with the program as possible an by a new drawing tool to the tool palette. When you turn-in your program you should identify what problems in newpaint2.c you have fixed.

The new drawing tool should allow drawing of a *poly-line*. That is, in the poly-line drawing mode a line is drawn from one mouse click to the next mouse click, rather than having to enter mouse clicks for each endpoint of each line as the current line drawing mode requires. You will need to define an interface convention to indicate that you have completed drawing all the lines to be included in the poly-line drawing.

The new drawing tool should be included as a new tool rather than modifying the existing line drawing tool.

Problem Set 2 Solution [HTML] [PS] [PDF]