```
            mapcolor(36,135,83,0);
            mapcolor(37,145,90,0);
            mapcolor(38,155,97,0);
            mapcolor(39,165,105,0);
            mapcolor(40,175,110,0);
            mapcolor(41,185,113,0);
            mapcolor(42,190,118,0);
            mapcolor(43,200,127,0);
            mapcolor(44,210,135,30);
            mapcolor(45,225,145,35);
            mapcolor(46,240,155,45);
            mapcolor(47,255,165,55);
            for (i=64;  i<128; i++) mapcolor(i,0,0,255);
            for (i=128; i<256; i++) mapcolor(i,255,0,0);
            mapcolor(851,0,150,0);          /* set up colors for instruction box */
            mapcolor(852,255,165,55);
            mapcolor(853,95,60,0);
            mapcolor(854,0,0,0);            /* color for indicator box background*/
        }
    }
```

```
/*  compute the compass heading in degrees of the input direction. */

#include "fogm.h"       /* fogm constants    */

float compass(direction)
double direction;
{
      float compassdir;

      compassdir = RTOD * direction;
      if (compassdir <= 90.0)
            compassdir = 90.0 - compassdir;
      else
            compassdir = 450.0 - compassdir;

      return(compassdir);
}
```

DISPLAY_TERRAIN

```
/*  Compute which polygons need to be drawn to display the terrain and
    output them in an order such that the polygons farthest from the viewer
    are drawn first and those closest are drawn last.

    Note: Eventhough this seems like a long routine, it is broken into 8
          independent cases based on the direction the camera is looking.
          If you understand one case the others are merely mirror images of the
          algorighm for other octants.      */

#include "fogm.h"
#include "math.h"
#include "gl.h"

display_terrain(vx, vy, vz, px, py, pz, fovy,
firstxgrid, firstzgrid, lastxgrid, lastzgrid)

Coord vx, vy, vz, px, py, pz;
int   fovy;
short firstxgrid, firstzgrid, lastxgrid, lastzgrid;

{
      extern float ground_plane[4][3];
      extern long gnd_plane_color;
      extern Object road[99][99];
      extern Object target[99][99];
      extern float savetriangle[99][99][2][3][3];
      extern long gridcolor[99][99];

      double lookdir;
      int threshold, count, startx, startz;
      short xgrid, zgrid;
      float tanval;
      float y;

      if (TV) viewport(0,474,0,474);
      else viewport(0,767,0,767);
      pushmatrix();

      color(SKYBLUE);
      clear();

      ortho2(0.0,1023.0,0.0,767.0);      /* outline the screen */
      color(BLACK);
      recti(0,0,1023,767);
      popmatrix();

      pushmatrix();
      perspective(fovy,1.0,0.0,19500.0);
      lookat(vx,vy,vz,px,py,pz,0.0);
```

```
/* determine the direction of the line of sight  */
lookdir = (double)atan2((float)(vz - pz), (float)(-(vx - px)));
if (lookdir < 0.0) lookdir += TWOPI;

/* lay down the ground plane */
color(gnd_plane_color);
polf(4, ground_plane);

/* put the grid objects through the geometry engine in an order
      based on the lookdir.  */
if (lookdir > SEVEN_QTR_PI)
{
     /* 8th OCTANT   */
     threshold = (int)(tan(lookdir+HALFPI) + 0.5);

     count = 0;
     startx = lastxgrid;
     startz = firstzgrid;
     while (startz <= lastzgrid) {
          zgrid = startz;
          xgrid = startx;

          while ((xgrid <= lastxgrid) && (zgrid <= lastzgrid)) {

               color(gridcolor[zgrid][xgrid]);
               polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
               polf(3,&savetriangle[zgrid][xgrid][1][0][0]);

               if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
               if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);
               /* check if tank should be drawn now */

               zgrid += 1;
               count += 1;

               if (count >= threshold) {
                    xgrid += 1;
                    count = 0;
               }
          }

          startx -= 1;
          count = 0;

          if (startx < firstxgrid) {
               startx = firstxgrid;
               startz += threshold;
          }
     }
}
else if ((lookdir > THREE_HALVES_PI) && (lookdir <= SEVEN_QTR_PI))
{
```

```
        /* 7th OCTANT   */
        tanval = tan(lookdir+HALFPI);
        if (tanval == 0.0)
              threshold = 1000;
        else
              threshold = (int)((1.0/tanval) + 0.5);

        count = 0;
        startx = lastxgrid;
        startz = firstzgrid;
        while (startx >= firstxgrid) {
              zgrid = startz;
              xgrid = startx;

              while ((xgrid >= firstxgrid) && (zgrid >= firstzgrid)) {

                    color(gridcolor[zgrid][xgrid]);
                    polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
                    polf(3,&savetriangle[zgrid][xgrid][1][0][0]);
                    if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
                    if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);

                    xgrid -= 1;
                    count += 1;

                    if (count >= threshold) {
                          zgrid -= 1;
                          count = 0;
                    }
              }

              startz += 1;
              count = 0;

              if (startz > lastzgrid) {
                    startz = lastzgrid;
                    startx -= threshold;
              }
        }
}
else if ((lookdir > FIVE_QTR_PI) && (lookdir <= THREE_HALVES_PI))
{
        /* 6th OCTANT   */
        tanval = -tan(lookdir+HALFPI);
        if (tanval == 0.0)
              threshold = 1000;
        else
              threshold = (int)((1.0/tanval) + 0.5);

        count = 0;
        startx = firstxgrid;
        startz = firstzgrid;
```

```
        while (startx <= lastxgrid) {
            zgrid = startz;
            xgrid = startx;

            while ((xgrid <= lastxgrid) && (zgrid >= firstzgrid)) {

                color(gridcolor[zgrid][xgrid]);
                polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
                polf(3,&savetriangle[zgrid][xgrid][1][0][0]);

                if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
                if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);
                xgrid += 1;
                count += 1;

                if (count >= threshold) {
                    zgrid -= 1;
                    count = 0;
                }
            }

            startz += 1;
            count = 0;

            if (startz > lastzgrid) {
                startz = lastzgrid;
                startx += threshold;
            }
        }
    }
    else if ((lookdir > PI) && (lookdir <= FIVE_QTR_PI))
    {
        /* 5th OCTANT   */
        threshold = (int)(-tan(lookdir+HALFPI) + 0.5);

        count = 0;
        startx = firstxgrid;
        startz = firstzgrid;
        while (startz <= lastzgrid) {
            zgrid = startz;
            xgrid = startx;

            while ((xgrid >= firstxgrid) && (zgrid <= lastzgrid)) {
                color(gridcolor[zgrid][xgrid]);
                polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
                polf(3,&savetriangle[zgrid][xgrid][1][0][0]);

                if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
                if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);
                zgrid += 1;
                count += 1;
```

```
                if (count >= threshold) {
                        xgrid -= 1;
                        count = 0;
                }
        }

        startx += 1;
        count = 0;

        if (startx > lastxgrid) {
                startx = lastxgrid;
                startz += threshold;
        }
    }
}
else if ((lookdir > THREE_QTR_PI) && (lookdir <= PI))
{
    /* 4th OCTANT   */
    threshold = (int)(tan(lookdir+HALFPI) + 0.5);

    count = 0;
    startx = firstxgrid;
    startz = lastzgrid;
    while (startz >= firstzgrid) {
        zgrid = startz;
        xgrid = startx;

        while ((xgrid >= firstxgrid) && (zgrid >= firstzgrid)) {

            color(gridcolor[zgrid][xgrid]);
            polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
            polf(3,&savetriangle[zgrid][xgrid][1][0][0]);
            if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
            if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);

            zgrid -= 1;
            count += 1;

            if (count >= threshold) {
                xgrid -= 1;
                count = 0;
            }
        }

        startx += 1;
        count = 0;

        if (startx > lastxgrid) {
            startx = lastxgrid;
            startz -= threshold;
        }
    }
```

```
}
else if ((lookdir > HALFPI) && (lookdir <= THREE_QTR_PI))
{
      /* 3rd OCTANT   */
      tanval = tan(lookdir+HALFPI);
      if (tanval == 0.0)
            threshold = 1000;
      else
            threshold = (int)((1.0/tanval) + 0.5);

      count = 0;
      startx = firstxgrid;
      startz = lastzgrid;
      while (startx <= lastxgrid) {
            zgrid = startz;
            xgrid = startx;

            while ((xgrid <= lastxgrid) && (zgrid <= lastzgrid)) {

                  color(gridcolor[zgrid][xgrid]);
                  polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
                  polf(3,&savetriangle[zgrid][xgrid][1][0][0]);

                  if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
                  if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);
                  xgrid += 1;
                  count += 1;

                  if (count >= threshold) {
                        zgrid += 1;
                        count = 0;
                  }
            }

            startz -= 1;
            count = 0;

            if (startz < firstzgrid) {
                  startz = firstzgrid;
                  startx += threshold;
            }
      }
}
else if ((lookdir > QTR_PI) && (lookdir <= HALFPI))

{
      /* 2nd OCTANT   */
      tanval = -(tan(lookdir+HALFPI));
      if (tanval == 0.0)
            threshold = 1000;
      else
            threshold = (int)((1.0/tanval) + 0.5);
```

```
        count = 0;
        startx = lastxgrid;
        startz = lastzgrid;
        while (startx >= firstxgrid) {
                zgrid = startz;
                xgrid = startx;

                while ((zgrid <= lastzgrid) && (xgrid >= firstxgrid)) {

                        color(gridcolor[zgrid][xgrid]);
                        polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
                        polf(3,&savetriangle[zgrid][xgrid][1][0][0]);

                        if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
                        if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);
                        xgrid -= 1;
                        count += 1;

                        if (count >= threshold) {
                                zgrid += 1;
                                count = 0;
                        }
                }

                startz -= 1;
                count = 0;

                if (startz < firstzgrid) {
                        startz = firstzgrid;
                        startx -= threshold;
                }
        }
}
else if ((lookdir >= 0.0) && (lookdir <= QTR_PI))
{
        /* 1st OCTANT   */
        threshold = (int)(-tan(lookdir+HALFPI) + 0.5);

        count = 0;
        startx = lastxgrid;
        startz = lastzgrid;
        while (startz >= firstzgrid) {
                zgrid = startz;
                xgrid = startx;

                while ((xgrid <= lastxgrid) && (zgrid >= firstzgrid)) {

                        color(gridcolor[zgrid][xgrid]);
                        polf(3,&savetriangle[zgrid][xgrid][0][0][0]);
                        polf(3,&savetriangle[zgrid][xgrid][1][0][0]);

                        if (road[zgrid][xgrid] != 0) callobj(road[zgrid][xgrid]);
```

```
                    if (target[xgrid][zgrid] != 0) callobj(target[xgrid][zgrid]);
                    zgrid -= 1;
                    count += 1;

                    if (count >= threshold) {
                        xgrid += 1;
                        count = 0;
                    }
                }

                startx -= 1;
                count = 0;

                if (startx < firstxgrid) {
                    startx = firstxgrid;
                    startz -= threshold;
                }
            }
        }
        popmatrix();
}
```

DIST_TO_LOS

```
#include "gl.h"
#include "math.h"
float dist_to_los(vx,vy,vz,px,py,pz,point)
/* compute the distance from the point "point" to the line of sight */

Coord vx,vy,vz,px,py,pz;
float point[3];

{
      float a,b,c;  /* direction numbers of line of sight */
      float d,e,f;
      float dist;

      a = (float)(px - vx);
      b = (float)(py - vy);
      c = (float)(pz - vz);

      d = point[0] - (float)vx;
      e = point[1] - (float)vy;
      f = point[2] - (float)vz;

      dist = sqrt((up_i(e*c - f*b,2) + up_i(f*a - d*c,2) + up_i(d*b - e*a,2))/
         (up_i(a,2) + up_i(b,2) + up_i(c,2)));

      return(dist);
}
```

```
#include "gl.h"
#include "math.h"
#include "stdio.h"
#include "fogm.h"

#define X   0
#define Y   1
#define Z   2

#define DIAGONAL   0
#define HORIZONTAL 1
#define VERTICAL   2

#define LOWER      0
#define UPPER      1

#define NONE       0
#define INTERSECT  1
#define PROPER     2


do_boundary(bound_type, which_triangle, xgrid, zgrid,
bound_start, bound_end, left_start,
left_end, right_start, right_end, start_corner_flag,
end_corner_flag, poly1, vertex_cnt)

int bound_type, which_triangle, xgrid, zgrid;

float bound_start[3], bound_end[3], left_start[3], left_end[3],
right_start[3], right_end[3];

int *start_corner_flag, *end_corner_flag;

float poly1[10][3];

int *vertex_cnt;


{
      int test_index, cnt, index;

      float bound_right[3], bound_left[3], bound_start_edge[3],
      bound_end_edge[3];

      float vertex_array[10][3];
      float road_poly[10][3];
      float grid_poly[10][3];

      int intersect_cnt;
```

```
int intersect_type, decending_sort;

float upper_bound, lower_bound;


float gnd_level();

int in_this_poly();

intersect_cnt = -1;

/* compute the verticies of the road segment currently
  being worked on */
for (index = 0; index < 3; ++index) {
      road_poly[0][index] = left_start[index];
      road_poly[1][index] = left_end[index];
      road_poly[2][index] = right_end[index];
      road_poly[3][index] = right_start[index];
}

/* compute the verticies of the grid triangle associated with
  this boundary */
grid_poly[0][X] = (float)(xgrid*FT_100M);
grid_poly[0][Z] = (float)((zgrid+1)*FT_100M);
grid_poly[1][X] = (float)((xgrid+1)*FT_100M);
grid_poly[1][Z] = (float)(zgrid*FT_100M);
if (which_triangle == UPPER) {
      grid_poly[2][X] = (float)((xgrid+1)*FT_100M);
      grid_poly[2][Z] = (float)((zgrid+1)*FT_100M);
}
else {
      grid_poly[2][X] = (float)(xgrid*FT_100M);
      grid_poly[2][Z] = (float)(zgrid*FT_100M);
}
if (bound_type == HORIZONTAL) {
      test_index = X;
}
else if (bound_type == VERTICAL) {
      test_index = Z;
}
else if (bound_type == DIAGONAL) {
      test_index = Z;
}

if (bound_start[test_index] < bound_end[test_index]) {
      lower_bound = bound_start[test_index];
      upper_bound = bound_end[test_index];
}
else {
      lower_bound = bound_end[test_index];
      upper_bound = bound_start[test_index];
}
```

```
/* determine points of intersection between left and right sides
  of the road and the boundary                */

line_intersect2(bound_start, bound_end, right_start, right_end,
bound_right, &intersect_type);
if (intersect_type == PROPER) {

      /* intersection lies on road line segment, add intersection
          to array */
      intersect_cnt += 1;
      vertex_array[intersect_cnt][X] = bound_right[X];
      vertex_array[intersect_cnt][Z] = bound_right[Z];
      vertex_array[intersect_cnt][Y] = gnd_level(bound_right[X],
      -bound_right[Z]);
}
else if ((intersect_type == INTERSECT) &&
   (in_this_poly(grid_poly, 3, right_start)) &&
   (bound_right[test_index] > lower_bound) &&
   (bound_right[test_index] < upper_bound)) {

      /* intersection point is beyond the bound of the road's right
            line segment, but the right start point is inside the polygon so
            add the road's right start point to the vertex array */

      intersect_cnt += 1;
      vertex_array[intersect_cnt][X] = right_start[X];
      vertex_array[intersect_cnt][Z] = right_start[Z];
      vertex_array[intersect_cnt][Y] = gnd_level(right_start[X],
      -right_start[Z]);
}
else if ((intersect_type == INTERSECT) &&
   (in_this_poly(grid_poly, 3, right_end)) &&
   (bound_right[test_index] > lower_bound) &&
   (bound_right[test_index] < upper_bound)) {

      /* intersection point is beyond the bound of the road's right
            line segment, but the right end point is inside the polygon so
            add the road's right end point to the vertex array */

      intersect_cnt += 1;
      vertex_array[intersect_cnt][X] = right_end[X];
      vertex_array[intersect_cnt][Z] = right_end[Z];
      vertex_array[intersect_cnt][Y] = gnd_level(right_end[X],
      -right_end[Z]);
}
line_intersect2(bound_start, bound_end, left_start, left_end,
bound_left, &intersect_type);
if (intersect_type == PROPER) {
      /* intersection lies on road line segment, add intersection
            to array */
      intersect_cnt += 1;
      vertex_array[intersect_cnt][X] = bound_left[X];
```

```
        vertex_array[intersect_cnt][Z] = bound_left[Z];
        vertex_array[intersect_cnt][Y] = gnd_level(bound_left[X],
        -bound_left[Z]);
}
else if ((intersect_type == INTERSECT) &&
    (in_this_poly(grid_poly, 3, left_start)) &&
    (bound_left[test_index] > lower_bound) &&
    (bound_left[test_index] < upper_bound)) {

        /* intersection point is beyond the bound of the road's left
             line segment, but the left start point is inside the polygon so
             add the road's left start point to the vertex array */

        intersect_cnt += 1;
        vertex_array[intersect_cnt][X] = left_start[X];
        vertex_array[intersect_cnt][Z] = left_start[Z];
        vertex_array[intersect_cnt][Y] = gnd_level(left_start[X],
        -left_start[Z]);
}
else if ((intersect_type == INTERSECT) &&
    (in_this_poly(grid_poly, 3, left_end)) &&
    (bound_left[test_index] > lower_bound) &&
    (bound_left[test_index] < upper_bound)) {

        /* intersection point is beyond the bound of the road's left
               line segment, but the left end point is inside the polygon so
               add the road's left end point to the vertex array */

        intersect_cnt += 1;
        vertex_array[intersect_cnt][X] = left_end[X];
        vertex_array[intersect_cnt][Z] = left_end[Z];
        vertex_array[intersect_cnt][Y] = gnd_level(left_end[X],
        -left_end[Z]);
}

/* if either of the bound's end points fall within the bounds of the
   road, add them to the array*/
if ((!*start_corner_flag) && (in_this_poly(road_poly, 4, bound_start))) {
        /* put in start bound point */
        *start_corner_flag = TRUE;
        intersect_cnt += 1;
        vertex_array[intersect_cnt][X] = bound_start[X];
        vertex_array[intersect_cnt][Z] = bound_start[Z];
        vertex_array[intersect_cnt][Y] = bound_start[Y];
}
if ((!*end_corner_flag) && (in_this_poly(road_poly, 4, bound_end))) {
        /* put in end bound point */
        *end_corner_flag = TRUE;
        intersect_cnt += 1;
        vertex_array[intersect_cnt][X] = bound_end[X];
        vertex_array[intersect_cnt][Z] = bound_end[Z];
        vertex_array[intersect_cnt][Y] = bound_end[Y];
```

```
    }
    /* determine the point of intersection between the start and end
      bound of the road and the grid boundary */
    line_intersect2(bound_start, bound_end, left_start, right_start,
    bound_start_edge, &intersect_type);
    if (intersect_type == PROPER) {
        /* intersection lies on road line segment, add intersection
            to array */
        intersect_cnt += 1;
        vertex_array[intersect_cnt][X] = bound_start_edge[X];
        vertex_array[intersect_cnt][Z] = bound_start_edge[Z];
        vertex_array[intersect_cnt][Y] = gnd_level(bound_start_edge[X],
        -bound_start_edge[Z]);
    }
    line_intersect2(bound_start, bound_end, left_end, right_end,
    bound_end_edge, &intersect_type);
    if (intersect_type == PROPER) {
        /* intersection lies on road line segment, add intersection
            to array */
        intersect_cnt += 1;
        vertex_array[intersect_cnt][X] = bound_end_edge[X];
        vertex_array[intersect_cnt][Z] = bound_end_edge[Z];
        vertex_array[intersect_cnt][Y] = gnd_level(bound_end_edge[X],
        -bound_end_edge[Z]);
    }
    /* put the points from the vertex_array into the poly1 array in
       the proper order */
    decending_sort = (bound_start[test_index] != lower_bound);
    sort_array(vertex_array, intersect_cnt, decending_sort, test_index);

    for (cnt = 0; cnt <= intersect_cnt; ++cnt) {
        *vertex_cnt += 1;
        poly1[*vertex_cnt][X] = vertex_array[cnt][X];
        poly1[*vertex_cnt][Y] = vertex_array[cnt][Y];
        poly1[*vertex_cnt][Z] = -vertex_array[cnt][Z];
    }
}
```

EDIT_INDBOX

```
/* update the control settings of the indicator box */
#include "fogm.h"
#include "gl.h"

edit_indbox(indbox, speedtag, headingtag, elevtag, altmsltag,
zoomtag, tilttag, pantag, desigtag, speed, compassdir,
vx, vy, vz, pan, tilt, zoom, designate)

Object indbox;

Tag speedtag, headingtag, elevtag, altmsltag, zoomtag, tilttag, pantag,
desigtag;

float speed, compassdir;

Coord vx, vy, vz;

double pan, tilt;

int designate;

int zoom;
{
      char chspeed[5], chheading[5], chelev[5], chaltmsl[5];
      float gnd_level();
      float zoomtic, pantic, tilttic;

      sprintf(chspeed,"%4.0f",speed);          /* convert speed to string  */
      sprintf(chheading,"%3.0f",compassdir);    /* convert heading to str   */
      sprintf(chelev,"%4.0f",vy - gnd_level(vx,vz)); /* convert elev AGL to str */
      sprintf(chaltmsl,"%4.0f",vy);            /* convert alt MSL to str  */

      /* compute new location for zoom, pan, and tilt indicators          */
      zoomtic = zoom *   -0.2766  + 222.128;
      tilttic = tilt *  721.92682 + 365.0;
      pantic  = pan  * -721.92682 + 435.0;

      editobj(indbox);              /* update the indicator display */
      objreplace(speedtag);
      charstr(chspeed);
      objreplace(headingtag);
      charstr(chheading);
      objreplace(elevtag);
      charstr(chelev);
      objreplace(altmsltag);
      charstr(chaltmsl);
      objreplace(zoomtag);
      move2(28.0,zoomtic);
      objreplace(tilttag);
```

```
    move2(42.0,tilttic);
    objreplace(pantag);
    move2(pantic,27.0);
    objreplace(desigtag);
    cmov2i(designate ? 10 : 19,10);
    charstr(designate ? "DESIGNATE" : "REJECT");
    closeobj();
}
```

```
#include "fogm.h"
#include "math.h"
#include "gl.h"

edit_navbox(navbox, arrowtag, vx, vz, direction,firstxgrid, firstzgrid,
lastxgrid, lastzgrid)
Object navbox;
Tag arrowtag;
Coord vx, vz;
double direction;
short firstxgrid, firstzgrid, lastxgrid, lastzgrid;
{
      Coord arrowx, arrowy, larrowx, larrowy, rarrowx, rarrowy;

      /* compute coordinates of arrow line segments for nav control box */
      arrowx = vx + cos(direction) * 2.0 * FEETPERGRID;
      arrowy = vz - sin(direction) * 2.0 * FEETPERGRID;
      larrowx = arrowx + cos(direction - 2.3561945) * FEETPERGRID;
      larrowy = arrowy - sin(direction - 2.3561945) * FEETPERGRID;
      rarrowx = arrowx + cos(direction + 2.3561954) * FEETPERGRID;
      rarrowy = arrowy - sin(direction + 2.3561945) * FEETPERGRID;

      /* update the contour map display with new info          */
      editobj(navbox);
      objreplace(arrowtag);
      move2(vx,vz);
      draw2(arrowx, arrowy);
      draw2(larrowx, larrowy);
      move2(arrowx, arrowy);
      draw2(rarrowx, rarrowy);
      rect(firstxgrid*FT_100M,-firstzgrid*FT_100M,
      (lastxgrid+1)*FT_100M, (-lastzgrid-1)*FT_100M);
      closeobj();
}
```

EXPLOSION

```
#include "gl.h"

explosion()

{
      int i,j;

      pushviewport();
      viewport(0,1023,0,767);
      color(BLACK);
      clear();
      swapbuffers();
      color(RED);
      clear();
      swapbuffers();
      swapbuffers();
      color(YELLOW);
      clear();
      swapbuffers();
      swapbuffers();
      color(RED);
      clear();
      swapbuffers();
      swapbuffers();
      color(YELLOW);
      clear();
      swapbuffers();
      swapbuffers();
      color(RED);
      clear();
      swapbuffers();
      swapbuffers();
      for (i = 0; i < 100000; i++)
            for (j = 0; j < 10; j++);
      popviewport();
}
```

FOGM (MAIN)

```
/* fogm.c --  an IRIS-2400 program  by Doug Smith & Dale Streyle
   It reads in a 10km x 10km section of a terrain map, computes a lighting
   and shading model for the terrain, and allows overflight                */

#include "gl.h"          /* get the graphics defs          */
#include "device.h"       /* get the graphics device defs    */
#include "fogm.h"         /* constants                      */
#include "math.h"         /* math function declarations     */
#include "get.h"         /* monitor type include file      */
#include "stdio.h"
#include "sys/signal.h"    /* used for screen dump utility    */
#include <sys/types.h>     /* contains the time sturcture tms */
#include <sys/times.h>     /* for time calls                 */

short gridpixel[100][100]; /* DMA elevation and vegatation data     */
float savetriangle[99][99][2][3][3];
long gridcolor[99][99];
Object road[99][99];
Object target[99][99];

float ground_plane[4][3];
long gnd_plane_color;
float tgt_pos[MAX_TGTS][3];
short tgt_grid_idx[MAX_TGTS][2];
short tgt_dir[MAX_TGTS], tgt_total = 0;
float randx, randy, randz; /* random offsets from tank reference point */

int framecnt;

float min_elev, max_elev;

Coord tankx, tanky, tankz;

float frames_sec[1000][2];


main()
{
      int greyscale = FALSE;   /* FALSE = color, TRUE = greys  */


      int designate;          /* boolean indicating desig/reject status       */

      int flying = TRUE;       /* boolean controlling flying loop             */

      int active = TRUE;       /* boolean controlling main program loop       */

      int nbyte, socket, connect_client(); /* networking variables & subroutine */
```

```
struct tms timestruct;   /* structure for real-time clock calls          */

int tgt_idx;             /* index of designated target                   */

double direction;        /* direction of travel in radians               */

float speed;             /* speed of travel in knots                     */

float compassdir;        /* desired direction of travel in compass deg   */

int fovy = 550;          /* field of view in perspective command         */

double pan = 0.0,
tilt = -15.0 * DTOR;  /* pan and tilt angles                             */

/* contour map, indicator, instruction  */
Object contour, navbox, indbox, instrbox;
Object tank, pre_l_obj 7];

Tag  headingtag, elevtag, speedtag, zoomtag, arrowtag, tilttag, pantag;
Tag  desigtag, altmsltag, pre_l_tag[6];

Colorindex unmask;

Coord vx, vy, vz;            /*  viewer x y and z coordinates */

Coord px, py, pz;            /* reference x y z coordinates for lookat */

Coord tgtx, tgty, tgtz;      /* targeted position on tank */

float randseed();       /* random number generator initialization */

int frames = -1;
long seconds, lastseconds, totalseconds = 0;
int numpolys;
float elapsed;
int idx;
FILE *fopen(), *fp;

/* first and last x and z indexes of the grid objects to draw */
short firstxgrid, firstzgrid, lastxgrid, lastzgrid;

readdata();  /* read the data file into the gridpixel array */

/* get socket number for networking */
/*if (NETWORKING) socket = connect_client("npscs-iris1",3); */

init_iris();                 /* initialize the iris */
unmask = (1<<getplanes()) - 1;
writemask(unmask);

randseed(times(&timestruct));    /* seed the random # generator */
```

```
init_tgts();                    /* define targets */

Screen_Dump(SCREENDUMP);              /* enable screen dumping */

billboard();                    /* produce intro screen  */

colorramp(greyscale,TRUE);      /* build all color ramps */

makescreens(pre_l_obj, pre_l_tag);  /* build objects for prelaunch */
makemap(&contour);                  /* build map object */
pre_l_obj[CONTOUR] = contour;

prelaunch(&vx, &vy, &vz, &direction, &compassdir,
&active, pre_l_obj, pre_l_tag);

if (active) {
      maketank(&tank);          /* build object for a tank */

      build_road();    /* build the objects that comprise the roads */

      /* process terrain data to build polygons and compute lighting */
      buildterrain();

      /* build object for the navigation display contour map */
      drawnavbox(&navbox, &arrowtag);

      /* build an object for the indicator box */
      makeindbox(&indbox,&headingtag,&elevtag,&altmsltag,&speedtag,
      &zoomtag,&tilttag,&pantag,&desigtag);


      makeinstrbox(&instrbox);  /* build object for control instruction box */

}  /* end of if (active) block */

while (active) {

      framecnt = 0;

      /* initialize the operator controls (mouse and dials) */
      init_controls(&pan, &tilt, &fovy, vy, greyscale, compassdir);

      pushviewport();
      viewport(0,1023,0,767);
      color(SKYBLUE);
      clear();
      popviewport();
      callobj(instrbox);
      callobj(indbox);
      editobj(contour);
      objreplace(STARTTAG);
      viewport(768,1023,512,767);
```

```
closeobj();
callobj(contour);
swapbuffers();
callobj(instrbox);
callobj(contour);
editobj(contour);
objreplace(STARTTAG);
viewport(0,768,0,768);
closeobj();

flying = TRUE;        /* missile is flying */
designate = TRUE;     /* a target can be designated */

while(flying)  {      /* until tgt is hit or 3-button exit */


    /* get values from user contols (mouse and dials) */
    read_controls(&designate, &greyscale, &flying, &active,
    &speed, &direction, &compassdir, &vy,
    &pan, &tilt, &fovy);

    /* calculate which target was closest to the line of
          sight */
    if (!designate) {
         nearest_tgt(vx,vy,vz,px,py,pz,&tgt_idx);
    }

    /* update targets' positions */
    get_tgt_posit(socket, designate, tgt_idx, &tgtx, &tgty, &tgtz, tank);

    /* update missile position */
    update_missile_posit(&direction, &compassdir, speed,
    designate, tgtx, tgty, tgtz,
    &vx, &vy, &vz, &flying);

    /* update camera lookat position */
    update_look_posit(direction, pan, tilt, vx, vy, vz,
    tgtx, tgty, tgtz, designate, &px, &py, &pz);

    /* determine which polygons need to be drawn */
    view_bounds(vx, vy, vz, px, py, pz, tilt, fovy,
    &firstxgrid, &firstzgrid, &lastxgrid, &lastzgrid);

    /* edit control display objects to reflect new values */
    edit_navbox(navbox, arrowtag, vx, vz, direction, firstxgrid,
    firstzgrid, lastxgrid, lastzgrid);
    edit_indbox(indbox, speedtag, headingtag, elevtag, altmsltag,
    zoomtag, tilttag, pantag, desigtag, speed,
    compassdir, vx, vy, vz, pan, tilt, fovy, designate);


    /* display the 3-D view of the terrain as seen by
```

```
                the camera      */
            display_terrain(vx, vy, vz, px, py, pz, fovy,
            firstxgrid, firstzgrid, lastxgrid, lastzgrid);

            /* display the control boxes */
            writemask(SAVEMAP);
            callobj(navbox);
            writemask(unmask);
            callobj(indbox);

            swapbuffers();

            seconds = times(&timestruct);
            numpolys = (lastxgrid - firstxgrid)*(lastzgrid-firstzgrid)*2;
            elapsed = (float)(seconds - lastseconds)/60.0;
            if ((frames >= 0) && (frames < 1000) ){
                frames_sec[frames][0] = (float)numpolys;
                frames_sec[frames][1] = 1.0/elapsed;
            }
            totalseconds += (seconds-lastseconds);
            if (totalseconds > 7200) {
                compactify();  /* do garbage collection every 2 mins */
                totalseconds = 0.0;
            }
            lastseconds = seconds;
            frames += 1;

        }  /* end of flying loop */

        if (active) {  /* explode & restart */
            explosion();
            prelaunch(&vx, &vy, &vz, &direction, &compassdir,
            &active, pre_l_obj, pre_l_tag);
        }

} /* end of active loop */

/* write out performance stats */
fp = fopen("speed.data", "w");
if (frames > 999) frames = 999;
for (idx = 0; idx <= frames; ++idx) {
    fprintf(fp,"%.2f %.2f0, frames_sec[idx][0], frames_sec[idx][1]);
}
```

```
    /* gracefully exit */
    if (NETWORKING) close(socket);
    setmonitor(HZ60);
    color(BLACK);
    clear();
    swapbuffers();
    clear();
    gexit();
    textinit();
    exit();
}   /* end of main */
```

FILES.H

```
/*  These are the files which contain data for the terrain elevations
   and roads   */
#define TERRAIN_FILE "/work/terrain/tenkmsq.dat"
#define ROAD_FILE "/work/terrain/Road.data"
```

FOGM.H

```
#define     elev_mask       0x1fff        /* mask to obtain elev value from datum    */

#define veg_mask     0x0007       /* mask to obtain vegatation value from
                        shifted datum */
#define     RD          0        /* code for reading a file in "open"      */

#define MAX           2800      /* max elev (ft) in contour map          */

#define MIN           967       /* min elev (ft) in contour map          */

#define SKYBLUE    4095      /* color index for sky color             */

#define ROADGREY       850       /* color index for the road              */

#define DELTAFOVY      50        /* field of view (zoom) increment of 5 deg */

#define PI             3.1415927

#define TWOPI          6.2831853

#define HALFPI         1.5707963

#define THREE_HALVES_PI 4.7123889

#define QTR_PI         0.7853982

#define THREE_QTR_PI    2.3561945

#define FIVE_QTR_PI     3.9269908

#define SEVEN_QTR_PI    5.4977871

#define RTOD           57.29578  /* radians to degrees conversion factor    */

#define DTOR           0.0174533 /* degrees to radians conversion factor    */

#define FPS_TO_KTS      35.525148 /* convert feet per 60th seconds to knots  */
```

```
#define PANSENS    30.0     /* scale factors (sensitivity) for
                            navigaion controls (mouse and dials)   */

#define SPEEDSENS      20

#define TILTSENS   50.0

#define DIRSENS    20.0

#define MAXLOOKDIST  32808.0  /* maximum distance that the camera can
                            look ahead in feet                */

#define FEETPERGRID   3280.8   /* number of feet in 1000 meters        */

#define ALTSCALE  1.05     /* altitude expansion factor, altitudes are
                            raised to this power to give an
                            exagerrated effect                */

#define NUMXGRIDS      10      /* number of 1K grid squares in the East-
                            West direction                    */

#define NUMZGRIDS      10      /* number of 1k grid squares in the North-
                            South direction                   */

#define FT_10K         32808    /* number FT in 10Km                  */

#define FT_100M        328.08   /* number FT in 100m                  */

#define GRID_FACTOR          13.03781  /* conversion factor                */

#define TV             0       /* 0 for SGI monitor, 1 for TV          */

#define SCREENDUMP     1       /* 1 to enable screen dumping, 0 otherwise  */

#define NETWORKING     0       /* 1 for target networking, 0 otherwise    */

#define INIT_PAN       0       /* initial, min and max pan angles in deg. */
#define MIN_PAN        -25
#define MAX_PAN         25

#define INIT_TILT      -15     /* initial, min and max tilt angles in deg.*/
#define MIN_TILT       -25
#define MAX_TILT        15

#define MAX_ALT         17000    /* maximum altitude for missle    */
#define MIN_ALT        0       /* minimum altitude for missle    */

#define INIT_SPEED      200       /* init, min and max spd (kts) for missle */
#define MIN_SPEED       0
#define MAX_SPEED       400

#define INIT_FOVY      550     /* initial field of view in tenth degrees */
```

```
#define    CONTOUR          0      /* Indicies for array obj          */
#define    SCREEN1      1
#define    SCREEN2      2
#define    SCREEN3      3
#define    INSTR        4
#define    STATS        5
#define    FLTPATH          6

#define LAUNCH          0      /* Indicies for array tag          */
#define    TARGET       1
#define    DIR          2
#define    HEAD         3
#define    TGT      4
#define    MISSILE      5

#define MAX_TGT_COLOR  847
#define MIN_TGT_COLOR  668

#define    MAX_TGTS    100

#define    SAVEMAP          0x00C0
```

GAMMARAMP

```
/*    This routine puts a gamma-corrected color ramp into the color map.     */

#include <math.h>

gammaramp(gammaconst,firstcolor,ncolors,
brightred,brightgreen,brightblue,
darkred,darkgreen,darkblue)

float gammaconst;   /* Strength of Gamma correction (try 1.0) */

long firstcolor;    /* index number of the first color to set */

long ncolors;       /* the number of colors to set */

long brightred,brightgreen,brightblue;  /* the bright end of the ramp */

long darkred,darkgreen,darkblue;        /* the dark end of the ramp   */

{

      long i;   /* temp loop index */

      float scl;  /* scale factor for gamma correction */

      long gcred,gcgreen,gcblue;  /* gamma corrected colors */


      for(i=0; i < ncolors; i++) /* for all colors...*/
      {
            /* compute the scale factor */
            scl = pow((float)i/(float)(ncolors-1) , 1.0/gammaconst);

            /* compute the gamma corrected colors */
            gcred = scl * (brightred - darkred) + darkred;
            gcgreen = scl * (brightgreen - darkgreen) + darkgreen;
            gcblue = scl * (brightblue - darkblue) + darkblue;

            mapcolor(firstcolor+i, gcred, gcgreen, gcblue); /* set the color */

      }

}
```

GET_TGT_POS

```
/* get targets' positions from iris1 if networking. Otherwise moves 10 targets
   in straight lines, reversing when they hit an edge                    */

#include "fogm.h"
#include "gl.h"
#include "math.h"
#include <sys/types.h>    /* contains the time sturcture tms */
#include <sys/times.h>    /* for time calls */

get_tgt_posit(socket,designate,tgt_idx,tgtx,tgty,tgtz,tank)

int socket, designate, tgt_idx;
float *tgtx, *tgty, *tgtz;
Object tank;

{
      extern float tgt_pos[MAX_TGTS][3];
      extern float randx, randy, randz;
      extern Object target[99][99];
      extern short tgt_grid_idx[MAX_TGTS][2];
      extern short tgt_total, tgt_dir[MAX_TGTS];
      short i, tgt_num;
      int nbyte, add1();
      float gnd_level(), dir, dx, dz, distance;
      long dist, d2;
      static long seconds;
      static long lastsec = -999;     /* -999 is flag to indicate no value */
      struct tms timestruct;

      seconds = times(&timestruct);

      if (lastsec  == -999)  /* compute distance targets move ahead */
            distance = 0.0;
      else
            distance = (float)((15.0/FPS_TO_KTS)*(seconds - lastsec));

      lastsec = seconds;              /* save for next pass */

      for (i = 0; i < tgt_total; i++)     /* delete targets from old positions */
            if (target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]]) {
                  delobj(target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]]);
                  target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]] = 0;
            }

      if (NETWORKING) {
            nbyte = read(socket, &tgt_total, sizeof(tgt_total));
            for (i = 0; i < tgt_total; i++) {
                  nbyte = read(socket, &tgt_grid_idx[i][0], sizeof(short));
                  nbyte = read(socket, &tgt_grid_idx[i][1], sizeof(short));
```

```
            nbyte = read(socket, &tgt_pos[i][0], sizeof(float));
            nbyte = read(socket, &tgt_pos[i][1], sizeof(float));
            nbyte = read(socket, &tgt_pos[i][2], sizeof(float));
            nbyte = read(socket, &tgt_dir[i], sizeof(short));
      }
}
else {
      tgt_total = 10;
      for (i = 0; i < tgt_total; i++) {
            dir = (float)(tgt_dir[i] / 10) * DTOR;
            tgt_pos[i][0] += cos(dir) * distance;
            tgt_pos[i][2] -= sin(dir) * distance;
            tgt_grid_idx[i][0] = (short)(tgt_pos[i][0]/FT_100M);
            tgt_grid_idx[i][1] = (short)(-tgt_pos[i][2]/FT_100M);
            if ((tgt_pos[i][0] > FT_10K) || (tgt_pos[i][0] < 0)) {
                  if (tgt_dir[i] > 1800) tgt_dir[i] -= 1800;
                  else tgt_dir[i] += 1800;
                  tgt_pos[i][1] = 0.0;
            }
            else if ((tgt_pos[i][2] < -FT_10K) || (tgt_pos[i][2] > 0)) {
                  if (tgt_dir[i] > 1800) tgt_dir[i] -= 1800;
                  else tgt_dir[i] += 1800;
                  tgt_pos[i][1] = 0.0;
            }
            else tgt_pos[i][1] = gnd_level(tgt_pos[i][0], tgt_pos[i][2]);
      }
}
if (!designate) {
      if (NETWORKING) {       /* find which target is designated */
            dist = up_i((float)(tgt_pos[0][0] - *tgtx),2) +
               up_i((float)(tgt_pos[0][2] - *tgtz),2);
            tgt_idx = 0;
            for (i = 1; i < tgt_total; i++) {
                  d2 = up_i((float)(tgt_pos[i][0] - *tgtx),2) +
                     up_i((float)(tgt_pos[i][2] - *tgtz),2);
                  if (d2 < dist) {
                        dist = d2;
                        tgt_idx = (int)i;
                  }
            }
      }
      *tgtx = tgt_pos[tgt_idx][0] + randx;
      *tgty = tgt_pos[tgt_idx][1] + randy;
      *tgtz = tgt_pos[tgt_idx][2] + randz;
}
tgt_num = tgt_total;
for (i = 0; i < tgt_num; i++) {
      dx = tgt_pos[i][0] - (float)tgt_grid_idx[i][0] * FT_100M;
      dz = (float)(-tgt_grid_idx[i][1]) * FT_100M - tgt_pos[i][2];
      if (dx < 15.0)
            if (dz < 15.0) {
                  add1(i,-1,0);
```

```
                    add1(i,-1,-1);
                    add1(i,0,-1);
                }
                else if (dz > 313.0) {
                    add1(i,0,1);
                    add1(i,-1,1);
                    add1(i,-1,0);
                }
                else            add1(i,-1,0);
            else if (dx > 313.0)
                if (dz < 15.0) {
                    add1(i,0,-1);
                    add1(i,1,-1);
                    add1(i,1,0);
                }
                else if (dz > 313.0) {
                    add1(i,1,0);
                    add1(i,1,1);
                    add1(i,0,1);
                }
                else            add1(i,1,0);
            else if (dz < 15.0)     add1(i,0,-1);
            else if (dz > 313.0)    add1(i,0,1);
        }
        for (i = 0; i < tgt_total; i++)    /* add targets to new positions */
            if (target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]]) {
                editobj(target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]]);
                pushmatrix();
                translate(tgt_pos[i][0],tgt_pos[i][1],tgt_pos[i][2]);
                rotate(tgt_dir[i], 'Y');
                callobj(tank);
                popmatrix();
                closeobj();
            }
            else {
                target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]] = genobj();
                makeobj(target[tgt_grid_idx[i][0]][tgt_grid_idx[i][1]]);
                pushmatrix();
                translate(tgt_pos[i][0],tgt_pos[i][1],tgt_pos[i][2]);
                rotate(tgt_dir[i], 'Y');
                callobj(tank);
                popmatrix();
                closeobj();
            }
}


add1(tgt_num,x,z)

short tgt_num,x,z;
{
    extern float tgt_pos[MAX_TGTS][3];
```

```
    extern short tgt_grid_idx[MAX_TGTS][2];
    extern short tgt_total, tgt_dir[MAX_TGTS];
    short i;

    tgt_pos[tgt_total][0] = tgt_pos[tgt_num][0];       /* copy pos. for "new" tgt */
    tgt_pos[tgt_total][1] = tgt_pos[tgt_num][1];
    tgt_pos[tgt_total][2] = tgt_pos[tgt_num][2];
    tgt_dir[tgt_total] = tgt_dir[tgt_num]; /* copy dir for "new" tgt */
    tgt_grid_idx[tgt_total][0] = tgt_grid_idx[tgt_num][0] + x; /* set pos in  */
    tgt_grid_idx[tgt_total][1] = tgt_grid_idx[tgt_num][1] + z; /* new grid sq */
    for (i = 0; i < 2; i++) {     /* reset if new grid sq outside 10km square */
        if (tgt_grid_idx[tgt_total][i] < 0) tgt_grid_idx[tgt_total][i] = 0;
        if (tgt_grid_idx[tgt_total][i] > 98) tgt_grid_idx[tgt_total][i] = 98;
    }
    tgt_total ++;
}
```

```
#include "math.h"
#include "fogm.h"
#define X  0
#define Y  1
#define Z  2
float gnd_level(vx, vz)

float vx, vz;

{
      extern short gridpixel[100][100];
      float interp_elev();
      float grid_level();
      float point[3], nw_corner[3], ne_corner[3], sw_corner[3], se_corner[3];
      float intersect[3];
      float elev;
      int xgrid, zgrid, intersect_type;

      /* determine which triangle the point falls in */
      xgrid = (int)(vx/FT_100M);
      zgrid = (int)(-vz/FT_100M);
      if (xgrid < 0) xgrid = 0;
      if (xgrid > 98) xgrid = 98;
      if (zgrid < 0) zgrid = 0;
      if (zgrid > 98) zgrid = 98;
      point[X] = vx;
      point[Z] = -vz;
      nw_corner[X] = (float)(xgrid*FT_100M);
      nw_corner[Z] = (float)((zgrid + 1)*FT_100M);
      elev = gridpixel[zgrid+1][xgrid] & elev_mask;
      nw_corner[Y] = pow(elev, ALTSCALE);
      sw_corner[X] = (float)(xgrid*FT_100M);
      sw_corner[Z] = (float)(zgrid*FT_100M);
      elev = gridpixel[zgrid][xgrid] & elev_mask;
      sw_corner[Y] = pow(elev, ALTSCALE);
      ne_corner[X] = (float)((xgrid+1)*FT_100M);
      ne_corner[Z] = (float)((zgrid+1)*FT_100M);
      elev = gridpixel[zgrid+1][xgrid+1] & elev_mask;
      ne_corner[Y] = pow(elev, ALTSCALE);
      se_corner[X] = (float)((xgrid+1)*FT_100M);
      se_corner[Z] = (float)(zgrid*FT_100M);
      elev = gridpixel[zgrid][xgrid + 1] & elev_mask;
      se_corner[Y] = pow(elev, ALTSCALE);

      if (-vz < (nw_corner[Z] - (vx - nw_corner[X]))) {
            /* point is in the lower triangle */

            /* find the point of intersection of a line through vx,vz
                  and the sw_corner with the diagonal */
```

```
        line_intersect2(sw_corner, point, nw_corner, se_corner, intersect,
        &intersect_type);

        /* find the elevation of the intersection on the diagonal */
        intersect[Y] = interp_elev(nw_corner, se_corner, intersect);

        /* find the elevation of the point vx, vy */
        return(interp_elev(sw_corner, intersect, point));
    }
    else {
        /* point is in the upper triangle */

        /* find the point of intersection of the diagonal with a line
             through th ne_corner and the point */
        line_intersect2(ne_corner, point, nw_corner, se_corner, intersect,
        &intersect_type);

        /* find the elevation of the intersection on the diagonal */
        intersect[Y] = interp_elev(nw_corner, se_corner, intersect);

        /* find the elevation of the point vx, vz */
        return(interp_elev(ne_corner, intersect, point));
    }
}
```

IN_THIS_POLY

```
#include "gl.h"

#define X      0
#define Y      1
#define Z      2

#define PROPER  2


int in_this_poly(polygon, num_vertex, point)
float polygon[10][3];
int num_vertex;
float point[3];

{
      int index;
      int pt_in, intersect_type;
      int num_crossings;
      float max_x, max_z, min_x, min_z;
      float intersect[3];
      float old_intersect[3];
      float start_test_line[3];

      max_x = polygon[0][X];
      min_x = polygon[0][X];
      max_z = polygon[0][Z];
      min_z = polygon[0][Z];

      for (index = 1; index < num_vertex; ++index) {
            if (polygon[index][X] < min_x) min_x = polygon[index][X];
            if (polygon[index][X] > max_x) max_x = polygon[index][X];
            if (polygon[index][Z] < min_z) min_z = polygon[index][Z];
            if (polygon[index][Z] > max_z) max_z = polygon[index][Z];
      }

      if ((point[X] < max_x) && (point[X] > min_x) && (point[Z] < max_z) &&
         (point[Z] > min_z)) {

            /* point may be polygon, test further by constructing a vertical line
                 from the point to a point outside the polygons bounds.  Count the number
                 of times this line crosses a side of the polygon.  If it crosses an
                 odd number of times the point is in the polygon, otherwise it is
                 outside the polygon */

            start_test_line[X] = point[X];
            start_test_line[Z] = max_z + 1000.0;

            num_crossings = 0;
            old_intersect[X] = -999.0;
```

```
        old_intersect[Z] = -999.0;
        for (index = 0; index < num_vertex -1; ++index) {
            line_intersect2(start_test_line, point, &polygon[index][0],
            &polygon[index+1][0], intersect, &intersect_type);
            /* if a proper intersection exists and it is not the same point
                    as the previous intersection (i.e it didn't intersect a vertex),
                    then add one to the number of crossings */
            if ((intersect_type == PROPER) && ((intersect[X] != old_intersect[X])
               || (intersect[Z] != old_intersect[Z]))) num_crossings += 1;
            old_intersect[X] = intersect[X];
            old_intersect[Z] = intersect[Z];
        }
        line_intersect2(start_test_line, point, &polygon[num_vertex-1][0],
        &polygon[0][0], intersect, &intersect_type);
        if (intersect_type == PROPER) num_crossings += 1;

        /* if the number of crossings is even, the point was outside */
        pt_in = ((num_crossings % 2) != 0);
        return(pt_in);
    }
    else {
        return(FALSE);
    }
}
```

INIT_CTRLS

```
/*  initialize the operator controls */

#include "fogm.h"          /* fogm constants */
#include "device.h"        /* graphics device definitions */
#include "gl.h"           /* graphics routine definitions */
#include "math.h"          /* math function definitions   */

init_controls(pan, tilt, fovy, alt, greyscale, compassdir)

double *pan;          /* initial pan angle in radians */
double *tilt;        /* initial tilt angle in radians */
int *fovy;          /* initial field of view in tenths of degrees */
Coord alt;          /* initial altitude of missile  */
int greyscale;      /* initial value of greyscale boolean */
float compassdir;     /* initial compass direction */

{

      *pan = INIT_PAN * DTOR;
      *tilt = INIT_TILT * DTOR;
      *fovy = INIT_FOVY;

      /* set initial, min, and max values for mouse & dials */
      setvaluator(MOUSEX,(short)(INIT_PAN*PANSENS),(short)(MIN_PAN*PANSENS),
      (short)(MAX_PAN*PANSENS));

      setvaluator(MOUSEY,(short)(INIT_TILT*TILTSENS),(short)(MIN_TILT*TILTSENS),
      (short)(MAX_TILT*TILTSENS));

      setvaluator(DIAL0,(short)(compassdir*DIRSENS), (short)(-360*DIRSENS),
      (short)(720*DIRSENS));

      setvaluator(DIAL4,(short)alt,MIN_ALT,MAX_ALT);

      setvaluator(DIAL2, (short)(INIT_SPEED*SPEEDSENS),
      (short)(MIN_SPEED*SPEEDSENS),
      (short)(MAX_SPEED*SPEEDSENS));

      setvaluator(DIAL3,greyscale,0,1);
}
```

INIT_IRIS

```
/* Initialize the graphics environment for the iris workstation */

#include "gl.h"            /* graphics definitions  */
#include "get.h"           /* monitor type definitions */
#include "fogm.h"          /* fogm constants  */

init_iris()
{
      long chunk;              /* number of bytes be which objects
                               increment  */
        ginit();               /* initialize the IRIS system          */
      doublebuffer();          /* put the IRIS into double buffer mode  */
      chunk = 128;
      chunksize(chunk);
      gconfig();               /* (means use the above command settings) */
      if (TV) {
            setmonitor(NTSC);          /* choose tv or SGI monitor          */
            fontdef(1,"TV.font");
            font(1);
      }
      else setmonitor(HZ60);
      cursoff();               /* turn off the cursor               */

      backface(TRUE);          /* turn on backface polygon removal     */

      color(BLACK);
      clear();
      swapbuffers();
}
```

INIT_TGTS

```
#include "fogm.h"
#include "gl.h"

init_tgts()
{
      extern short tgt_total;
      extern Object target[99][99];
      short x, y;
      int init_tgt();

      for (x = 0; x < 99; x++) for (y = 0; y < 99; y++) target[x][y] = 0;
      if (!NETWORKING) {
            tgt_total = 10;
            init_tgt(0,9.8,3.5,1295);
            init_tgt(1,9.5,3.5,1295);
            init_tgt(2,9.4,3.1,1295);
            init_tgt(3,9.8,0.5,1800);
            init_tgt(4,9.5,0.0,1355);
            init_tgt(5,8.0,0.0,1445);
            init_tgt(6,4.0,0.0,1450);
            init_tgt(7,0.0,0.5,450);
            init_tgt(8,9.5,9.8,2700);
            init_tgt(9,9.8,8.5,1800);
      }
}

init_tgt(tgt_num,xoffset,zoffset,direction)

short tgt_num, direction;
float xoffset, zoffset;

{
      extern short tgt_dir[MAX_TGTS];
      extern float tgt_pos[MAX_TGTS][3];

      tgt_pos[tgt_num][0] = xoffset * FEETPERGRID;
      tgt_pos[tgt_num][2] = -zoffset * FEETPERGRID;
      tgt_dir[tgt_num] = direction;
}
```

INTERP_ELEV

```
#include "math.h"

#define X    0
#define Y    1
#define Z    2

float interp_elev(line_start, line_end, point)

float line_start[3], line_end[3], point[3];
{
      long float line_deltax, line_deltaz, point_deltax, point_deltaz;
      float line_length, dist_to_point;
      float interpolation;

      line_deltax = (long float)(line_end[X] - line_start[X]);
      line_deltaz = (long float)(line_end[Z] - line_start[Z]);

      point_deltax = (long float)(line_start[X] - point[X]);
      point_deltaz = (long float)(line_start[Z] - point[Z]);

      line_length = (float)hypot(line_deltax, line_deltaz);
      dist_to_point = (float)hypot(point_deltax, point_deltaz);

      interpolation = line_start[Y] + ((line_end[Y] - line_start[Y]) *
         (dist_to_point/line_length));

      return(interpolation);
}
```

```
/* this is file lightorient.c */
/*

   It is a routine that computes lighting for a polygon based
   upon the angle between the Normal vector of the polygon
   and the direction to the light source.

   lightorient(xyz,ncoords,ax,ay,az,lx,ly,lz,colormin,colormax,colortouse)

   xyz[][3] = floating coords of the polygon.

   ncoords = number of coordinates.

   ax,ay,az = interior point of the whole object. Used to determine
         outward facing normal of the polygon. This is the same
         point of reference that would be used for backface
         polygon removal.

   lx,ly,lz = vector pointing in direction of the light source.

   colormin, colormax = indices used for the colors assigned to this
               polygon. The user is responsible for setting
               up the color ramp.

   colortouse = returned color used to light the polygon.

   Note: the routine also puts the polygons out ordered counterclockwise
      with respect to the interior point for ease of backface polygon
      removal.

*/

#include <math.h>

#include <gl.h>

#define MAXCOORDS 80

#define PIDIV2 1.570796327

float txyz[MAXCOORDS][3];   /* temp coord hold */


lightorient(xyz,ncoords,ax,ay,az,lx,ly,lz,colormin,colormax,colortouse)

float xyz[][3];
long ncoords;
```

```
float ax,ay,az;   /* interior point of the whole object. */

float lx,ly,lz;   /* direction to the light source       */

long colormin,colormax;   /* color min/max indices       */

long *colortouse; /* color used to light the polygon (return value) */

{

      long i,j;   /* loop temps */

      long npoly_orient();   /* direction test function */

      float v1[3],v2[3];        /* vectors used to compute
                                   the polygon's normal */

      float normal[3];          /* the polygon's normal */

      float normalmag;            /* normal's magnitude   */

      float lightmag;           /* magnitude of the light vector */

      double dotprod;             /* dot product of N and L */

      float radians;            /* angle between N and L */


      /* check the number of coords in the input array */
      if(ncoords > MAXCOORDS)
      {
            printf("LIGHTORIENT: too many coords passed to me! = %d0,ncoords);
            exit(1);
      }

      /* orient the polygon so that its counterclockwise with respect
          to the interior point */

      if(npoly_orient(ncoords,xyz,ax,ay,az) == 1)
      {
            /* the polygon is clockwise, reverse it. */
            for(i=0; i < ncoords; i=i+1)
            {
                  for(j=0; j < 3; j=j+1)
                  {
                        txyz[i][j] = xyz[ncoords-i-1][j];
                  }
            }

            for(i=0; i < ncoords; ++i)
                  for (j=0; j < 3; ++j)
                        xyz[i][j] = txyz[i][j];
```

```
}

/* the coordinates are ordered counterclockwise in array xyz */

/* compute the normal vector for the polygon using the first
     three vertices...*/

/* compute the first vector to use in the computation */
v1[0] = xyz[2][0] - xyz[1][0];
v1[1] = xyz[2][1] - xyz[1][1];
v1[2] = xyz[2][2] - xyz[1][2];

/* compute the second vector to use in computing the normal */
v2[0] = xyz[0][0] - xyz[1][0];
v2[1] = xyz[0][1] - xyz[1][1];
v2[2] = xyz[0][2] - xyz[1][2];

/* the normal is v1 x v2 */
normal[0] = v1[1]*v2[2] - v1[2]*v2[1];
normal[1] = v1[2]*v2[0] - v1[0]*v2[2];
normal[2] = v1[0]*v2[1] - v1[1]*v2[0];

/* compute the magnitude of the normal */
normalmag = sqrt((normal[0]*normal[0])+(normal[1]*normal[1])+
   (normal[2]*normal[2]));

/* check the magnitude of the normal */
if(normalmag == 0.0)
{
      normalmag = 0.00001; /* a small number */
}

/* compute the light mag */
lightmag = sqrt((lx*lx)+(ly*ly)+(lz*lz));

if(lightmag == 0.0)
{
      lightmag = 0.00001;  /* a small number */
}

/* compute N . L (normal dot product with the light source direction) */
dotprod = (normal[0] * lx) + (normal[1] * ly) + (normal[2] * lz);

dotprod = dotprod/(lightmag*normalmag);

/* dotprod = cos(theta) of the angle between N and L.
     Convert to angle in radians */
radians = acos(dotprod);

/* compute the color we should use */
if(-PIDIV2 <= radians && radians <= PIDIV2)
{
```

```
            /* if the angle is negative, set to positive */
            if(radians < 0.0)
            {
                  radians = -radians;
            }

            *colortouse = ((colormax-colormin)/PIDIV2)*(PIDIV2-radians)+colormin;
      }
      else
      {
            *colortouse = colormin;
      }

      /* set the color */
      color(*colortouse);

      /* draw the poly */
      /* polf(ncoords,txyz); */

}
```

LINE_INTERSECT2

```
#include "gl.h"

#define X  0
#define Z  2
#define NONE  0
#define INTERSECT  1
#define PROPER    2


line_intersect2(start1, end1, start2, end2, intersect,
intersect_type)

float start1[3], end1[3], start2[3], end2[3], intersect[3];
int *intersect_type;

{
      /* given two lines of the form z = mx + b  and  z = nx + c,
           solving for x when the z's are equal gives x = (c-b)/(m-n).
           Then solve for z using x in either of the above equations. */

      float m,n,b,c;
      float min1_x, min2_x, max1_x, max2_x, min1_z, min2_z, max1_z, max2_z;

      *intersect_type = PROPER;

      /* slope and z intercept of line1 */
      if (end1[X] != start1[X]) {
            m = (end1[Z] - start1[Z])/(end1[X] - start1[X]);
            b = ((start1[Z] - end1[Z])/(end1[X] - start1[X])) * start1[X] + start1[Z];
            if (end2[X] != start2[X]) {  /* both lines are non-vertical */
                  /* slope and z intercept of line2 */
                  n = (end2[Z] - start2[Z])/(end2[X] - start2[X]);
                  c = ((start2[Z] - end2[Z])/(end2[X] - start2[X])) * start2[X] +
                     start2[Z];

                  if (m != n) {
                        intersect[X] = (c-b)/(m-n);
                        intersect[Z] = m*intersect[X] + b;
                  }
                  else {  /* both lines have equal slopes */
                        *intersect_type = NONE;
                  }
            }
            else { /* line1 is non-vertical, line2 is vertical */
                  intersect[X] = end2[X];
                  intersect[Z] = m*intersect[X] + b;
            }
      }
      else {
```

```
        if (end2[X] != start2[X]) { /* line1 is vertical, line2 is non-vertical*/
                /* slope and z intercept of line2 */
                n = (end2[Z] - start2[Z])/(end2[X] - start2[X]);
                c = ((start2[Z] - end2[Z])/(end2[X] - start2[X])) * start2[X] +
                   start2[Z];
                intersect[X] = end1[X];
                intersect[Z] = n*intersect[X] + c;
        }
        else {  /* both lines are vertical */
                *intersect_type = NONE;
        }
}
if (*intersect_type != NONE) {
        /* see if the intersection is proper, or if only the extensions of the
                line segments intersect */
        if (start1[X] < end1[X]) {
                min1_x = start1[X];
                max1_x = end1[X];
        }
        else {
                min1_x = end1[X];
                max1_x = start1[X];
        }
        if (start1[Z] < end1[Z]) {
                min1_z = start1[Z];
                max1_z = end1[Z];
        }
        else {
                min1_z = end1[Z];
                max1_z = start1[Z];
        }
        if (start2[X] < end2[X]) {
                min2_x = start2[X];
                max2_x = end2[X];
        }
        else {
                min2_x = end2[X];
                max2_x = start2[X];
        }
        if (start2[Z] < end2[Z]) {
                min2_z = start2[Z];
                max2_z = end2[Z];
        }
        else {
                min2_z = end2[Z];
                max2_z = start2[Z];
        }
```

```
            if ((intersect[X] <= max1_x) && (intersect[X] <= max2_x) &&
               (intersect[X] >= min1_x) && (intersect[X] >= min2_x) &&
               (intersect[Z] <= max1_z) && (intersect[Z] <= max2_z) &&
               (intersect[Z] >= min1_z) && (intersect[Z] >= min2_z)) {

                  *intersect_type = PROPER;
            }
            else {
                  *intersect_type = INTERSECT;
            }
      }
}
```

```
/* drawnavbox.c - this function is called by the FOG-M missile simulator to
   build an object on top of the contour map in the upper right-hand corner
   of the screen. Navbox contains the direction arrow and view box in red. */

#include "gl.h"
#include "fogm.h"
#include "device.h"

drawnavbox(navbox, arrowtag)
Object *navbox;
Tag    *arrowtag;
{
      *navbox = genobj();  /* create the navigation contol and display object */
      makeobj(*navbox);
      if (TV) viewport(475,635,323,474);
      else viewport(768,1023,512,767);  /*  upper right hand corner of screen */
      pushmatrix();              /* draw arrow in feet coordinates */
      ortho2(-10.0,10.0 + NUMXGRIDS*FEETPERGRID, -10.0,
      -10.0 - NUMZGRIDS*FEETPERGRID);
      color(BLACK);
      clear();
      color(128);
      *arrowtag = gentag();
      maketag(*arrowtag);
      move2(0.0,0.0);
      draw2(0.0,0.0);
      draw2(0.0,0.0);
      move2(0.0,0.0);
      draw2(0.0, 0.0);
      rect(0.0,0.0,0.0,0.0);       /*  view box  */
      popmatrix();
      closeobj();
}
```

MAKEINDBOX

```
/* makeindbox.c is a function that creates an object that displays the control
   idicators for the FOG-M missile simulation                              */

#include "gl.h"
#include "fogm.h"

makeindbox(indbox,headingtag,elevtag,altmsltag,speedtag,zoomtag,tilttag,pantag,desigtag)
Object *indbox;
Tag *headingtag, *elevtag, *speedtag, *zoomtag, *tilttag, *pantag, *desigtag;
Tag *altmsltag;
{
      *indbox = genobj();
      makeobj(*indbox);
      if (TV) viewport(475,635,162,322);
      else viewport(768,1023,256,511);        /* middle box on side of screen */
      pushmatrix();
      ortho2(0.0,255.0,0.0,255.0);      /* use screen sized coordinates */

      color(854);                 /* clear the window           */
      clear();
      linewidth(2);

      color(BLACK);
      recti(0,0,255,255);             /* outline box              */

      color(YELLOW);                  /* print labels for readouts    */
      cmov2i(10,240);
      charstr("SPEED");
      cmov2i(55,225);
      charstr("kts");
      cmov2i(90,240);
      charstr("HEADING");
      circ(140.0,232.0,3.0);              /* "degree" symbol         */
      cmov2i(180,240);
      charstr("Alt AGL");  /* AGL = above ground level */
      cmov2i(225,225);
      charstr("ft");
      cmov2i(180,200);
      charstr("Alt MSL");  /* MSL = mean sea level     */
      cmov2i(225,185);
      charstr("ft");

      cmov2i(50,130);
      charstr("ZOOM");
      move2i(45,200);                 /* draw slider bar frame   */
      draw2i(25,200);
      draw2i(25,70);
      draw2i(45,70);
      cmov2i(15,196);
```

```
charstr("8");   /* label slider bar values */
cmov2i(6,170);
charstr("15");
cmov2i(6,144);
charstr("25");
cmov2i(6,118);
charstr("35");
cmov2i(6,92);
charstr("45");
cmov2i(6,66);
charstr("55");

color(WHITE);                /* readouts in white...        */
cmov2i(10,225);              /* initialize to dummy values  */
*speedtag = gentag();
maketag(*speedtag);
charstr(" 200");             /*  speed  */

cmov2i(108,225);
*headingtag = gentag();
maketag(*headingtag);
charstr("  0");              /* heading */

cmov2i(180,225);
*elevtag = gentag();
maketag(*elevtag);
charstr("1000");             /* altitude above ground level */

cmov2i(180,185);
*altmsltag = gentag();
maketag(*altmsltag);
charstr("1000");             /* altitude from mean sea level */

color(RED);

*zoomtag= gentag();          /* indicator for zoom slider bar */
maketag(*zoomtag);
move2(28.0,135.0);
rdr2(10.0,5.0);
rdr2(0.0,-10.0);
rdr2(-10.0,5.0);

popmatrix();

if (TV) viewport(0,474,0,474);   /* reset for heads-up display */
else viewport(0,767,0,767);

pushmatrix();

ortho2(0.0,767.0,0.0,767.0);     /* use screen sized coordinates */

color(WHITE);
```

```
if (TV) linewidth(2);
else linewidth(1);

rectfi(365,370,370,375);        /* draw center of crosshairs    */
rectfi(396,370,401,375);
rectfi(365,391,370,396);
rectfi(396,391,401,396);
move2i(0,383);
draw2i(360,383); /* draw crosshairs              */
move2i(406,383);
draw2i(767,383);
move2i(383,0);
draw2i(383,365);
move2i(383,401);
draw2i(383,767);

linewidth(2);

move2i(30,50);                  /* draw TILT slider bar frame   */
draw2i(40,50);
draw2i(40,680);
draw2i(30,680);
cmov2i(0,676);
charstr("+25");    /* label slider bar values      */
cmov2i(0,613);
charstr("+20");
move2i(30,617);
draw2i(40,617);
cmov2i(0,550);
charstr("+15");
move2i(30,554);
draw2i(40,554);
cmov2i(0,487);
charstr("+10");
move2i(30,491);
draw2i(40,491);
cmov2i(0,424);
charstr(" +5");
move2i(30,428);
draw2i(40,428);
cmov2i(0,361);
charstr("  0");
move2i(30,365);
draw2i(40,365);
cmov2i(0,298);
charstr(" -5");
move2i(30,302);
draw2i(40,302);
cmov2i(0,235);
charstr("-10");
move2i(30,239);
draw2i(40,239);
```

```
cmov2i(0,172);
charstr("-15");
move2i(30,176);
draw2i(40,176);
cmov2i(0,109);
charstr("-20");
move2i(30,113);
draw2i(40,113);
cmov2i(0,46);
charstr("-25");

*tilttag = gentag();            /* indicator for TILT slider bar */
maketag(*tilttag);
move2(42.0,365.0);
rdr2(10.0,-5.0);
rdr2( 0.0,10.0);
rdr2(-8.0,-4.0);
rdr2( 6.0,-3.0);
rdr2( 0.0, 4.0);
rdr2(-2.0,-1.0);
rdr2( 1.0,-1.0);

move2i(120,15);                 /* draw PAN slider bar frame    */
draw2i(120,25);
draw2i(750,25);
draw2i(750,15);
cmov2i(107,3);
charstr("-25");    /* label slider bar values      */
cmov2i(170,3);
charstr("-20");
move2i(183,15);
draw2i(183,25);
cmov2i(233,3);
charstr("-15");
move2i(246,15);
draw2i(246,25);
cmov2i(296,3);
charstr("-10");
move2i(309,15);
draw2i(309,25);
cmov2i(363,3);
charstr("-5");
move2i(372,15);
draw2i(372,25);
cmov2i(431,3);
charstr("0");
move2i(435,15);
draw2i(435,25);
cmov2i(494,3);
charstr("+5");
move2i(498,15);
draw2i(498,25);
```

```
    cmov2i(552,3);
    charstr("+10");
    move2i(561,15);
    draw2i(561,25);
    cmov2i(615,3);
    charstr("+15");
    move2i(624,15);
    draw2i(624,25);
    cmov2i(678,3);
    charstr("+20");
    move2i(687,15);
    draw2i(687,25);
    cmov2i(741,3);
    charstr("+25");

    *pantag = gentag();          /* indicator for PAN slider bar */
    maketag(*pantag);
    move2(435.0,27.0);
    rdr2(  5.0,10.0);
    rdr2(-10.0, 0.0);
    rdr2(  4.0,-8.0);
    rdr2(  3.0, 6.0);
    rdr2( -4.0, 0.0);
    rdr2(  1.0,-2.0);
    rdr2(  1.0, 1.0);

    move2i(0,30);          /* designate/reject box */
    draw2i(100,30);
    draw2i(100,0);
    *desigtag = gentag();
    maketag(*desigtag);
    cmov2i(10,10);
    charstr("DESIGNATE");

    popmatrix();
    closeobj();
}
```

MAKEINSTRBOX

```
/* makeinstrbox.c - this function builds an object that contains an instruction
  summary for the FOG-M missile simulation                              */

#include "gl.h"
#include "fogm.h"

makeinstrbox(instrbox)

Object *instrbox;

{
      *instrbox = genobj();
      makeobj(*instrbox);
      if (TV) viewport(475,635,0,161);
      else viewport(768,1023,0,255);     /* box is in lower right hand corner */
      pushmatrix();
      ortho2(0.0,255.0,0.0,255.0);       /* use screen-sized coordinates      */

      color(851);                        /* use a medium green                */
      clear();
      linewidth(2);

      color(852);                 /*  use light brown            */
      rectfi(10,20,110,195);      /*  draw the mouse control box  */
      rectfi(135,80,245,195);     /*  draw the dial control box   */
      color(BLACK);               /*  outline controls           */
      recti(10,20,110,195);
      recti(135,80,245,195);
      recti(0,0,255,255);

      color(BLACK);
      cmov2i(60,230);
      charstr("C O N T R O L S");
      cmov2i(37,200);
      charstr("MOUSE");
      cmov2i(172,200);
      charstr("DIALS");

      cmov2i(25,60);
      charstr("TILT");
      move2i(70,62);                     /* draw arrow */
      draw2i(75,55);
      draw2i(75,75);
      draw2i(70,68);
      move2i(75,75);
      draw2i(80,68);
      move2i(75,55);
      draw2i(80,62);
```

```
cmov2i(25,30);
charstr("PAN");
move2i(67,40);                    /* draw arrow */
draw2i(60,35);
draw2i(80,35);
draw2i(73,40);
move2i(80,35);
draw2i(73,30);
move2i(60,35);
draw2i(67,30);

color(853);               /* dark brown                */
rectfi(20,85,40,185);      /* draw mouse buttons         */
rectfi(50,85,70,185);
rectfi(80,85,100,185);
color(BLACK);              /* outline buttons           */
recti(20,85,40,185);
recti(50,85,70,185);
recti(80,85,100,185);

color(853);
circfi(160,165,20);        /* draw dials               */
circfi(160,110,20);
circfi(220,165,20);
circfi(220,110,20);
color(BLACK);              /* outline dials             */
circi(160,165,20);
circi(160,110,20);
circi(220,165,20);
circi(220,110,20);
color(WHITE);
cmov2i(147,160);
charstr("SPD");  /* label dials */
cmov2i(147,106);
charstr("DIR");
cmov2i(207,106);
charstr("ALT");
cmov2i(207,160);
charstr("CLR");

cmov2i(25,170);
charstr("Z");     /* label mouse buttons */
cmov2i(25,158);
charstr("O");
cmov2i(25,146);
charstr("O");
cmov2i(25,134);
charstr("M");
cmov2i(25,110);
charstr("I");
cmov2i(25,98);
charstr("N");
```

```
    cmov2i(55,170);
    charstr("D");
    cmov2i(55,158);
    charstr("E");
    cmov2i(55,146);
    charstr("S");
    cmov2i(55,134);
    charstr("I");
    cmov2i(55,122);
    charstr("G");
    cmov2i(85,170);
    charstr("Z");
    cmov2i(85,158);
    charstr("O");
    cmov2i(85,146);
    charstr("O");
    cmov2i(85,134);
    charstr("M");
    cmov2i(85,110);
    charstr("O");
    cmov2i(85,98);
    charstr("U");
    cmov2i(85,86);
    charstr("T");

    popmatrix();
    closeobj();
}
```

```
/* makemap.c - this function is called by the FOG-M missile simulator to
   build an object containing a contour map. The map is used for the full
   screen display in prelaunch, and in the upper right corner of the flight
   display in fogm.                                                    */

#include "gl.h"
#include "fogm.h"
#include "device.h"

makemap(contour)
Object *contour;
{
      short i, j, elev, length, lastcolor, breakpt[15];
      int colour;
      extern short gridpixel[100][100]; /* terrain elevations & vegetation */

      /* compute elevations where color changes should occur */
      for (i = 1; i < 16; i++) breakpt[i-1] = (((MAX - MIN) / 16 ) * i) + MIN;

      *contour = genobj();  /* create the navigation contol and display object */
      makeobj(*contour);
      viewport(0,767,0,767);
      pushmatrix();
      ortho2(0.0,100.0,0.0,100.0);    /* use array index space */

      color(BLACK);
      clear();

      lastcolor = BLACK;
      linewidth(8);

      for (i=0; i < 100; ++i) {       /* draw column i       */
            move2i(i,0);              /* start at bottom of column     */
            length = 0;               /* # adjacent points of the same color */
            for (j = 0; j < 100; ++j) {   /* for each row in column i     */
                  elev = gridpixel[j][i] & elev_mask;   /* mask off veg code    */
                  if (elev < breakpt[0]) colour = 16;   /* assign green colors       */
                  else if (elev < breakpt[1]) colour = 17;
                  else if (elev < breakpt[2]) colour = 18;
                  else if (elev < breakpt[3]) colour = 19;
                  else if (elev < breakpt[4]) colour = 20;
                  else if (elev < breakpt[5]) colour = 21;
                  else if (elev < breakpt[6]) colour = 22;
                  else if (elev < breakpt[7]) colour = 23;
                  else if (elev < breakpt[8]) colour = 24;
                  else if (elev < breakpt[9]) colour = 25;
                  else if (elev < breakpt[10]) colour = 26;
                  else if (elev < breakpt[11]) colour = 27;
                  else if (elev < breakpt[12]) colour = 28;
```

```
            else if (elev < breakpt[13]) colour = 29;
            else if (elev < breakpt[14]) colour = 30;
            else colour = 31;

            /* if veg-code = 0 (i.e. veg < 1 meter) shift to brown colors */
            if (!((gridpixel[j][i] >> 13) & veg_mask)) colour += 16;

            if (colour == lastcolor) length++;   /* don't draw yet */
            else {                    /* draw now that color has changed */
                color(lastcolor);
                rdr2i(0,length);
                lastcolor = colour;          /* reset for new draw */
                length = 1;
            }
        } /* end for j */

        color(colour);                /* draw last (top) line */
        rdr2i(0,length);
    } /* end for i */

    if (!TV) {
        color(BLACK);                 /* draw grid on top of map */
        linewidth(1);

        for (i = 10; i < 100; i+=10) {  /* draw interior lines */
            move2i(i,0);                /* horizontals */
            draw2i(i,100);
            move2i(0,i);                /* verticals   */
            draw2i(100,i);
        }
    }

    linewidth(2);               /* draw exterior border */
    rect(0.0,0.0,100.0,100.0);

    popmatrix();
    closeobj();
}
```

MAKESCREENS

```
/* makescreens.c  -  builds graphical objects for prelaunch's instructional
  screens and readout boxes.                                          */

#include "gl.h"
#include "device.h"
#include "fogm.h"

makescreens(obj,tag)

Object obj[7];
Tag    tag[6];

{

      obj[INSTR] = genobj();          /* object for pre-launch instructions */
      makeobj(obj[INSTR]);
      if (TV) viewport(475,635,239,474);
      else viewport(767,1023,385,767);
      pushmatrix();
      ortho2(0.0,255.0,0.0,384.0);
      color(CYAN);
      clear();
      color(BLUE);
      rectfi(10,10,245,374);
      color(WHITE);
      cmov2i(30,340);
      charstr("PRE-LAUNCH INSTRUCTIONS");
      cmov2i(25,300);
      charstr("1. PRESS LEFT MOUSE");
      cmov2i(52,285);
      charstr("BUTTON TO LOCK IN");
      cmov2i(52,270);
      charstr("LAUNCH POSITION");
      cmov2i(25,220);
      charstr("2. PRESS RIGHT MOUSE");
      cmov2i(52,205);
      charstr("BUTTON TO LOCK IN");
      cmov2i(52,190);
      charstr("TARGET LOCATION");
      cmov2i(25,140);
      charstr("3. PRESS MIDDLE MOUSE");
      cmov2i(52,125);
      charstr("BUTTON TO LAUNCH");
      cmov2i(25, 75);
      charstr("4. PRESS ALL THREE");
      cmov2i(52, 60);
      charstr("BUTTONS TO EXIT");
      popmatrix();
      closeobj();
```

```
/* define object for displaying user input for missile launch
    position and target location. Also displays computed heading
    and distance to target */

obj[STATS] = genobj();
makeobj(obj[STATS]);
if (TV) viewport(475,635,0,238);
else viewport(767,1023,0,384);
pushmatrix();
ortho2(0.0,255.0,0.0,384.0);
color(CYAN);
clear();
color(BLUE);
rectfi(10,10,245,374);
color(WHITE);
cmov2i(30,340);
charstr("PRE-LAUNCH STATISTICS");
cmov2i(25,260);
charstr("LAUNCH POSITION: 10SFQ");
cmov2i(70,235);
charstr("X COORD: ");
cmov2i(70,220);
charstr("Y COORD: ");
cmov2i(170,235);
tag[LAUNCH] = gentag();
maketag(tag[LAUNCH]);
charstr("   ");
cmov2i(170,220);
charstr("   ");
cmov2i(25,180);
charstr("TARGET LOCATION: 10SFQ");
cmov2i(70,155);
charstr("X COORD:      ");
cmov2i(70,140);
charstr("Y COORD:      ");
cmov2i(170,155);
tag[TARGET] = gentag();
maketag(tag[TARGET]);
charstr("   ");
cmov2i(170,140);
charstr("   ");
cmov2i(25,100);
charstr("HEADING: ");
cmov2i(25,60);
charstr("DISTANCE: ");
cmov2i(106,100);
tag[HEAD] = gentag();
maketag(tag[HEAD]);
charstr("  ");
cmov2i(115,60);
charstr("   ");
popmatrix();
```

```
closeobj();

/* define object for lines & circles showing flightpath on contour map */

obj[FLTPATH] = genobj();
makeobj(obj[FLTPATH]);
pushmatrix();
if (TV) viewport(0,474,0,474);
else viewport(0,767,0,767);
ortho2(0.0,100.0,0.0,100.0);
color(BLACK);
clear();
color(64);
linewidth(3);
tag[MISSILE] = gentag();
maketag(tag[MISSILE]);
circf(0.0,0.0,0.0);
move2(0.0, 0.0, 0.0);
draw2(0.0, 0.0, 0.0);
color(128);
tag[TGT] = gentag();
maketag(tag[TGT]);
circf(0.0,0.0,0.0);
popmatrix();
closeobj();

/* define object for displaying first screen of operator instructions */

obj[SCREEN1] = genobj();
makeobj(obj[SCREEN1]);
color(BLUE);              /* set background color */
clear();
color(RED);
linewidth(10);
recti(0,0,1023,767);
linewidth(1);
color(WHITE);
cmov2i(420,500);
charstr("WELCOME");
cmov2i(420,450);
charstr("TO  THE");
cmov2i(320,400);
charstr("FIBER-OPTICALLY GUIDED MISSILE");
cmov2i(420,350);
charstr("(FOG-M)");
cmov2i(410,300);
charstr("SIMULATION");
cmov2i(310,100);
charstr("PRESS MIDDLE MOUSE BUTTON TO CONTINUE...");
cmov2i(315,85);
charstr("OR PRESS ALL 3 MOUSE BUTTONS TO EXIT.");
closeobj();
```

```
/* define object for displaying second screen of operator instructions */

obj[SCREEN2] = genobj();
makeobj(obj[SCREEN2]);
color(BLUE);                /* set background color */
clear();
linewidth(10);
color(RED);
recti(0,0,1023,767);
linewidth(1);
color(WHITE);
cmov2i(210,600);
charstr("THE FOG-M PROGRAM PROVIDES A SIMULATED MISSILE LAUNCH AND");
cmov2i(210,575);
charstr("OUT-THE-WINDOW VIEW OF THE TERRAIN AS SEEN FROM THE OPERATOR'S");
cmov2i(210,550);
charstr("CONSOLE ON THE GROUND.");
cmov2i(210,500);
charstr("THE GENERAL AREA FOR THIS FLIGHT SIMULATION IS FT HUNTER LIGGETT");
cmov2i(210,475);
charstr("CALIFORNIA AND VICINITY.");
cmov2i(210,425);
charstr("THE SPECIFIC TEST AREA IS A 10 KILOMETER REGION DESIGNATED BY");
cmov2i(210,400);
charstr("UNIVERSAL TRANSVERSE MERCATOR (UTM) GRID COORDINATES 10SFQ58.");
cmov2i(300,100);
charstr("PRESS MIDDLE MOUSE BUTTON TO CONTINUE,");
cmov2i(305,85);
charstr("OR PRESS ALL 3 MOUSE BUTTONS TO EXIT.");
closeobj();

/* define object for displaying third screen of operator instructions */

obj[SCREEN3] = genobj();
makeobj(obj[SCREEN3]);
color(BLUE);                /* set background color */
clear();
linewidth(10);
color(RED);
recti(0,0,1023,767);
linewidth(1);
color(WHITE);
cmov2i(385,650);
charstr("PRE-LAUNCH ORIENTATION");
cmov2i(200,600);
charstr("1. WHEN THE PRE-LAUNCH PHASE OF THE FOG-M SIMULATION BEGINS, A");
cmov2i(200,585);
charstr("2-DIMENSIONAL CONTOUR MAP OF THE TEST AREA (UTM 10SFQ58) WILL BE");
cmov2i(200,570);
charstr("DISPLAYED ON THE OPERATOR CONSOLE. TWO CONTROL PANELS CONTAINING");
cmov2i(200,555);
charstr("PRE-LAUNCH INSTRUCTIONS AND CURRENT LAUNCH STATISTICS WILL ALSO");
```

```
    cmov2i(200,540);
    charstr("BE DISPLAYED.");
    cmov2i(200,490);
    charstr("2. THE OPERATOR WILL BE REQUIRED TO PROVIDE TWO CRITICAL DATA");
    cmov2i(200,475);
    charstr("ITEMS TO THE LAUNCH CONTROL SYSTEM; INITIAL LAUNCH POSITION AND");
    cmov2i(200,460);
    charstr("TARGET LOCATION.");
    cmov2i(200,410);
    charstr("3. TO DEFINE INITIAL LAUNCH POSITION, MOVE CURSOR OVER DESIRED");
    cmov2i(200,395);
    charstr("LOCATION (REFER TO LAUNCH STATISTICS CONTROL PANEL TO VIEW THE");
    cmov2i(200,380);
    charstr("CURRENT UTM GRID COORDINATES). PRESS LEFT MOUSE BUTTON TO LOCK");
    cmov2i(200,365);
    charstr("IN LAUNCH POSITION.");
    cmov2i(200,315);
    charstr("4. TO DEFINE TARGET LOCATION, MOVE CURSOR OVER DESIRED LOCATION");
    cmov2i(200,300);
    charstr("(REFER TO LAUNCH STATISTICS CONTROL PANEL TO VIEW CURRENT UTM");
    cmov2i(200,285);
    charstr("GRID COORDINATES). PRESS RIGHT MOUSE BUTTON TO LOCK IN TARGET");
    cmov2i(200,270);
    charstr("LOCATION. THE BLUE LINE DISPLAYS THE PROJECTED FLIGHT PATH. THE");
    cmov2i(200,255);
    charstr("MISSILE WILL FLY AT A CONSTANT VELOCITY AND HEADING. THE LAUNCH");
    cmov2i(200,240);
    charstr("STATISTICS CONTROL PANEL WILL DISPLAY COMPUTED MISSILE HEADING");
    cmov2i(200,225);
    charstr("IN DEGREES (0 DEGREES DUE NORTH).");
    cmov2i(240,100);
    charstr("PRESS MIDDLE MOUSE BUTTON TO MOVE INTO PRE-LAUNCH PHASE,");
    cmov2i(326,85);
    charstr("OR PRESS ALL 3 MOUSE BUTTONS TO EXIT.");
    closeobj();
}
```

MAKETANK

```
#include "gl.h"
#include "fogm.h"

maketank(item)
Object *item;

{

        long points = 4, bigpoints = 8;
        float parray[8][3];
        float lx,ly,lz;
        long cmin = MIN_TGT_COLOR, cmax = MAX_TGT_COLOR, c1;

        lx = 400.0 * 41.01;  /* direction of lightsource */
        ly = 6000.0;
        lz = 200.0 * (-41.01);


        *item=genobj();
        makeobj(*item);

        /* draw right side of tank CCW */
        parray[0][0] = -10.0;
        parray[0][1] =  6.0;
        parray[0][2] = -5.0;
        parray[1][0] = -15.0;
        parray[1][1] =  4.0;
        parray[1][2] = -5.0;
        parray[2][0] = -15.0;
        parray[2][1] =  2.0;
        parray[2][2] = -5.0;
        parray[3][0] = -10.0;
        parray[3][1] =  0.0;
        parray[3][2] = -5.0;
        parray[4][0] =  10.0;
        parray[4][1] =  0.0;
        parray[4][2] = -5.0;
        parray[5][0] =  15.0;
        parray[5][1] =  2.0;
        parray[5][2] = -5.0;
        parray[6][0] =  15.0;
        parray[6][1] =  4.0;
        parray[6][2] = -5.0;
        parray[7][0] =  10.0;
        parray[7][1] =  6.0;
        parray[7][2] = -5.0;
        lightorient(parray,bigpoints,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
        color(c1);
```

```
polf(bigpoints,parray);

/* front of tank  CW */
parray[0][0] = 15.0;
parray[0][1] =  5.0;
parray[0][2] = -5.0;
parray[1][0] = 15.0;
parray[1][1] =  3.0;
parray[1][2] = -5.0;
parray[2][0] = 15.0;
parray[2][1] =  3.0;
parray[2][2] =  5.0;
parray[3][0] = 15.0;
parray[3][1] =  5.0;
parray[3][2] =  5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* draw left side of tank CW */
parray[0][0] =  10.0;
parray[0][1] =  6.0;
parray[0][2] = 5.0;
parray[1][0] =  15.0;
parray[1][1] =  4.0;
parray[1][2] = 5.0;
parray[2][0] =  15.0;
parray[2][1] =  2.0;
parray[2][2] = 5.0;
parray[3][0] =  10.0;
parray[3][1] =  0.0;
parray[3][2] = 5.0;
parray[4][0] = -10.0;
parray[4][1] =  0.0;
parray[4][2] = 5.0;
parray[5][0] = -15.0;
parray[5][1] =  2.0;
parray[5][2] = 5.0;
parray[6][0] = -15.0;
parray[6][1] =  4.0;
parray[6][2] = 5.0;
parray[7][0] = -10.0;
parray[7][1] =  6.0;
parray[7][2] = 5.0;
lightorient(parray,bigpoints,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(bigpoints,parray);

/* back of tank CCW */
parray[0][0] =  -15.0;
parray[0][1] =  4.0;
parray[0][2] =  5.0;
```

```
parray[1][0] =  -15.0;
parray[1][1] =  2.0;
parray[1][2] =  5.0;
parray[2][0] =  -15.0;
parray[2][1] =  2.0;
parray[2][2] = -5.0;
parray[3][0] =  -15.0;
parray[3][1] =  4.0;
parray[3][2] = -5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* top middle of tank body  CCW */
parray[0][0] = -10.0;
parray[0][1] =  6.0;
parray[0][2] = -5.0;
parray[1][0] = -10.0;
parray[1][1] =  6.0;
parray[1][2] =  5.0;
parray[2][0] =  10.0;
parray[2][1] =  6.0;
parray[2][2] =  5.0;
parray[3][0] =  10.0;
parray[3][1] =  6.0;
parray[3][2] = -5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* top front of tank body  CCW */
parray[0][0] = 10.0;
parray[0][1] =  6.0;
parray[0][2] = -5.0;
parray[1][0] = 10.0;
parray[1][1] =  6.0;
parray[1][2] =  5.0;
parray[2][0] = 15.0;
parray[2][1] =  4.0;
parray[2][2] =  5.0;
parray[3][0] = 15.0;
parray[3][1] =  4.0;
parray[3][2] = -5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* top back of tank body  CCW */
parray[0][0] = -10.0;
parray[0][1] =  6.0;
parray[0][2] = -5.0;
parray[1][0] = -15.0;
```

```
parray[1][1] =  4.0;
parray[1][2] = -5.0;
parray[2][0] = -15.0;
parray[2][1] =  4.0;
parray[2][2] =  5.0;
parray[3][0] = -10.0;
parray[3][1] =  6.0;
parray[3][2] =  5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* bottom middle of tank CW*/
parray[0][0] = -10.0;
parray[0][1] = 0.0;
parray[0][2] = -5.0;
parray[1][0] =  10.0;
parray[1][1] = 0.0;
parray[1][2] = -5.0;
parray[2][0] =  10.0;
parray[2][1] = 0.0;
parray[2][2] =  5.0;
parray[3][0] = -10.0;
parray[3][1] = 0.0;
parray[3][2] =  5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* bottom front of tank CW */
parray[0][0] = 10.0;
parray[0][1] = 0.0;
parray[0][2] = -5.0;
parray[1][0] = 15.0;
parray[1][1] = 2.0;
parray[1][2] = -5.0;
parray[2][0] = 15.0;
parray[2][1] = 2.0;
parray[2][2] =  5.0;
parray[3][0] = 10.0;
parray[3][1] = 0.0;
parray[3][2] =  5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* bottom back of tank CW */
parray[0][0] =  -10.0;
parray[0][1] = 0.0;
parray[0][2] = -5.0;
parray[1][0] =  -10.0;
parray[1][1] = 0.0;
```

```
parray[1][2] =  5.0;
parray[2][0] =  -15.0;
parray[2][1] = 2.0;
parray[2][2] =  5.0;
parray[3][0] =  -15.0;
parray[3][1] = 2.0;
parray[3][2] = -5.0;
lightorient(parray,points,0.0,0.0,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* right side of gun barrel */
parray[0][0] = 1.6667;
parray[0][1] =  8.0;
parray[0][2] =  -0.5;
parray[1][0] = 2.3333;
parray[1][1] =  7.0;
parray[1][2] =  -0.5;
parray[2][0] = 17.0;
parray[2][1] =  7.0;
parray[2][2] =  -0.5;
parray[3][0] = 17.0;
parray[3][1] =  8.0;
parray[3][2] =  -0.5;
lightorient(parray,points,5.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* top of gun barrel */
parray[0][0] = 1.6667;
parray[0][1] =  8.0;
parray[0][2] =  0.5;
parray[1][0] = 1.6667;
parray[1][1] =  8.0;
parray[1][2] = -0.5;
parray[2][0] = 17.0;
parray[2][1] =  8.0;
parray[2][2] = -0.5;
parray[3][0] = 17.0;
parray[3][1] =  8.0;
parray[3][2] =  0.5;
lightorient(parray,points,5.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* left side of gun barrel */
parray[0][0] = 17.0;
parray[0][1] =  8.0;
parray[0][2] =  0.5;
parray[1][0] = 17.0;
parray[1][1] =  7.0;
parray[1][2] =  0.5;
```

```
parray[2][0] = 2.3333;
parray[2][1] =  7.0;
parray[2][2] =  0.5;
parray[3][0] = 1.6667;
parray[3][1] =  8.0;
parray[3][2] =  0.5;
lightorient(parray,points.5.0,2.5,0.0.lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* end of gun barrel */
parray[0][0] = 17.0;
parray[0][1] =  8.0;
parray[0][2] = -0.5;
parray[1][0] = 17.0;
parray[1][1] =  7.0;
parray[1][2] = -0.5;
parray[2][0] = 17.0;
parray[2][1] =  7.0;
parray[2][2] =  0.5;
parray[3][0] = 17.0;
parray[3][1] =  8.0;
parray[3][2] =  0.5;
lightorient(parray,points,5.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/*bottom of gun barrel */
parray[0][0] = 2.3333;
parray[0][1] =  7.0;
parray[0][2] =  0.5;
parray[1][0] = 2.3333;
parray[1][1] =  7.0;
parray[1][2] = -0.5;
parray[2][0] = 17.0;
parray[2][1] =  7.0;
parray[2][2] = -0.5;
parray[3][0] = 17.0;
parray[3][1] =  7.0;
parray[3][2] =  0.5;
lightorient(parray,points,5.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
poif(points,parray);

/* right side of turret */
parray[0][0] = -3.0;
parray[0][1] = 9.0;
parray[0][2] =  -1.0;
parray[1][0] = -5.0;
parray[1][1] = 6.0;
parray[1][2] =  -3.0;
parray[2][0] =  3.0;
```

```
parray[2][1] =  6.0;
parray[2][2] =  -3.0;
parray[3][0] =  1.0;
parray[3][1] =  9.0;
parray[3][2] =  -1.0;
lightorient(parray,points,-1.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* front side of turret */
parray[0][0] = 1.6667;
parray[0][1] = 9.0;
parray[0][2] = -1.0;
parray[1][0] = 3.0;
parray[1][1] = 6.0;
parray[1][2] = -3.0;
parray[2][0] = 3.0;
parray[2][1] = 6.0;
parray[2][2] =  3.0;
parray[3][0] = 1.6667;
parray[3][1] = 9.0;
parray[3][2] =  1.0;
lightorient(parray,points,-1.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* left side of turret */
parray[0][0] =  1.6667;
parray[0][1] =  9.0;
parray[0][2] =  1.0;
parray[1][0] =  3.0;
parray[1][1] =  6.0;
parray[1][2] =  3.0;
parray[2][0] = -5.0;
parray[2][1] =  6.0;
parray[2][2] =  3.0;
parray[3][0] = -3.0;
parray[3][1] =  9.0;
parray[3][2] =  1.0;
lightorient(parray,points,-1.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
color(c1);
polf(points,parray);

/* back side of turret */
parray[0][0] = -3.0;
parray[0][1] = 9.0;
parray[0][2] =  1.0;
parray[1][0] = -5.0;
parray[1][1] = 6.0;
parray[1][2] =  3.0;
parray[2][0] = -5.0;
parray[2][1] = 6.0;
```

```
        parray[2][2] = -3.0;
        parray[3][0] = -3.0;
        parray[3][1] = 9.0;
        parray[3][2] = -1.0;
        lightorient(parray,points,-1.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
        color(c1);
        polf(points,parray);

        /* top of turret */
        parray[0][0] = -3.0;
        parray[0][1] = 9.0;
        parray[0][2] =  1.0;
        parray[1][0] = -3.0;
        parray[1][1] = 9.0;
        parray[1][2] = -1.0;
        parray[2][0] =  1.0;
        parray[2][1] = 9.0;
        parray[2][2] = -1.0;
        parray[3][0] =  1.0;
        parray[3][1] = 9.0;
        parray[3][2] =  1.0;
        lightorient(parray,points,-1.0,2.5,0.0,lx,ly,lz,cmin,cmax,&c1);
        color(c1);
        polf(points,parray);

        closeobj();

    }
```

NEAREST_TGT

```
#include "gl.h"
#include "fogm.h"
nearest_tgt(vx,vy,vz,px,py,pz,tgt_idx)
Coord vx, vy, vz, px, py, pz;
int *tgt_idx;

{
      float dist, dist_to_los();
      float min_dist;
      float num_tgts;
      extern float tgt_pos[MAX_TGTS][3];
      int index;

      num_tgts = 10;
      min_dist = dist_to_los(vx,vy,vz,px,py,pz,&tgt_pos[0][0]);
      *tgt_idx = 0;

      for (index = 1; index < num_tgts; ++index) {
            dist = dist_to_los(vx,vy,vz,px,py,pz,&tgt_pos[index][0]);
            if (dist < min_dist) {
                  min_dist = dist;
                  *tgt_idx = index;
            }
      }
}
```

```
/* npoly_orient.c  */

#include <gl.h>
#include <math.h>

int npoly_orient(ncoords,xyz,xinside,yinside,zinside)
unsigned int  ncoords;
Coord xyz[][3];
Coord xinside, yinside, zinside;
{

      register unsigned short int   i,j;         /* loop temps */

      Coord center[3];   /* center coordinate of the polygon */

      Coord a[3], b[3]; /* vector hold locations for the vectors that run
                           from the center coordinate to the points of the
                           polygon */

      Coord xn[3], xmn[3];  /* points on line containing normal that are
                               on opposite sides of the plane containing
                               the polygon.
                            */

      float distton;       /* distance to point n from pt inside. */

      float disttomn;      /* distance to point -n from pt inside. */

      Coord normal[3];     /* the normal vector computed from a x b */


      /* compute the center coordinate of the polygon */
      center[0] = 0.0;
      center[1] = 0.0;
      center[2] = 0.0;

      for(i=0; i < ncoords; i++)
      {
            for(j=0; j < 3; j++)
            {
                  center[j] += xyz[i][j];
            }
      }

      /* divide out by the number of coordinates */
      for(j=0; j < 3; j++)
      {
            center[j] = center[j]/(float)ncoords;
      }
```

```
/* check the first 2 coordinates of the polygon for their direction */

/* compute vector a. It runs from the center coordinate to coordinate 0 */
for(j=0; j < 3; j++)
{
      a[j] = xyz[0][j] - center[j];
}

/* compute vector b. It runs from the center coordinate to coordinate 1 */
for(j=0; j <3; j++)
{
      b[j] = xyz[1][j] - center[j];
}

/* compute a x b to get the normal vector */
normal[0] = a[1]*b[2] - a[2]*b[1];
normal[1] = a[2]*b[0] - a[0]*b[2];
normal[2] = a[0]*b[1] - a[1]*b[0];

/* compute point n, offset pt from center in direction of normal */
for(j=0; j < 3; j++)
{
      xn[j] = center[j] + normal[j];
}

/* compute point -n, offset pt from center in opposite direction
     from normal.
   */
for(j=0; j < 3; j++)
{
      xmn[j] = center[j] - normal[j];
}

/* compute the distance the inside pt is from point n */
distton = sqrt((xn[0] - xinside) * (xn[0] - xinside) +
   (xn[1] - yinside) * (xn[1] - yinside) +
   (xn[2] - zinside) * (xn[2] - zinside));

/* compute the distance the inside pt is from point -n */
disttomn = sqrt((xmn[0] - xinside) * (xmn[0] - xinside) +
   (xmn[1] - yinside) * (xmn[1] - yinside) +
   (xmn[2] - zinside) * (xmn[2] - zinside));
```

```
        /* if the dist(n) < dist(-n), then n points back towards the
             inside point and is on the same side of the plane as inside.
             a x b is then clockwise.
           */
        if(distton < disttomn)
        {
             return(1);   /* clockwise */
        }
        else
        {
             return(0);   /* counterclockwise */
        }

}
```

PRELAUNCH

```
/* The function prelaunch is the user interface portion of the FOG-M
   flight simulation. It allows the operator to interactively enter
   critical data items necessary to simulate the missile in flight.
   The function returns the initial launch position in the x-z plane
   and also the direction of flight. */


#include "gl.h"
#include "device.h"
#include "fogm.h"
#include "math.h"

prelaunch(vx, vy, vz, direction, compassdir, active, obj, tag)

Coord  *vx, *vy, *vz;
double *direction;
float *compassdir;
int    *active;
Object obj[7];
Tag    tag[6];

{
      float gnd_level();
      float compass();
      int  screencnt, launchlock, targetlock;
      int xval, yval, xlaunch, ylaunch, xtarget, ytarget, utm_x, utm_y;
      char xtemp[35], ytemp[35], dist[35], heading[35];
      float distance;
      double xdistance, ydistance;
      Colorindex unmask;

      xtemp[0] = ' ';
      ytemp[0] = ' ';
      dist[0]  = ' ';
      heading[0] = ' ';

      unmask=(1<<getplanes()) -1;
      writemask(unmask);
     .if (TV) viewport(0,635.0,474);
      else viewport(0,1023,0,767);
      pushmatrix();
      ortho2(0.0,1023.0,0.0,767.0);

      *direction = 0.0;                  /* initialize the direction */

      cursoff();                  /* turn the cursor off */

      callobj(obj[SCREEN1]);           /* display screen 1 */
      swapbuffers();
```

```
screencnt = 1;       /* initialize counter for screen displays */

while(TRUE) {
      frontbuffer(TRUE);
      if (getbutton(MOUSE2) && !(getbutton(MOUSE1)) && !(getbutton(MOUSE3))) {
            ringbell();
            while (getbutton(MOUSE2));
            screencnt += 1;
            if (screencnt == 2) callobj(obj[SCREEN2]);
            else if (screencnt == 3) callobj(obj[SCREEN3]);
            else break;
      }
      if (getbutton(MOUSE1) && (getbutton(MOUSE2)) && (getbutton(MOUSE3))) {
            *active = FALSE;
            goto exit;
      }
}
frontbuffer(FALSE);

editobj(obj[FLTPATH]);   /* erase previous missile path */
objreplace(tag[MISSILE]);
circf(0.0, 0.0, 0.0);
move2(0.0, 0.0);
draw2(0.0, 0.0);
objreplace(tag[TGT]);
circf(0.0, 0.0, 0.0);
closeobj();

editobj(obj[STATS]);       /* erase previous launch statistics */
objreplace(tag[HEAD]);
charstr("");
cmov2i(115,60);
charstr("");
objreplace(tag[TARGET]);
charstr("");
cmov2i(0,0);
charstr("");
closeobj();

setcursor(0,RED,unmask);              /* set up cursor and mouse */
attachcursor(MOUSEX,MOUSEY);
setvaluator(MOUSEX,384,0,767);
setvaluator(MOUSEY,384,0,767);
curson();

launchlock = FALSE;
targetlock = FALSE;

callobj(obj[CONTOUR]);  /* load static displays into both buffers */
callobj(obj[INSTR]);
callobj(obj[STATS]); /* included so swapped buffer doesn't have "hole" */
swapbuffers();
```

```
callobj(obj[CONTOUR]);
callobj(obj[INSTR]);

while(TRUE) {
      if (getbutton(MOUSE1) && (getbutton(MOUSE2)) && (getbutton(MOUSE3))) {
            *active = FALSE;
            goto exit;
      }

      xval = getvaluator(MOUSEX);       /* read the x and y mouse positions */
      yval = getvaluator(MOUSEY);

      utm_x = (50000 + (int)(xval * GRID_FACTOR)); /* compute grid coordinates */
      utm_y = (80000 + (int)(yval * GRID_FACTOR));

      sprintf(xtemp,"%4d",utm_x);     /* store coordinates in temporary buffer */
      sprintf(ytemp,"%4d",utm_y);

      /* if LEFT MOUSE selected lock in launch position and update control panel */

      if (getbutton(MOUSE3) && (!getbutton(MOUSE2)) && (!getbutton(MOUSE1))) {
            ringbell();
            xlaunch = xval;
            ylaunch = yval;
            launchlock = TRUE;
            *vx = ((float)((xval * FT_10K)/767));
            *vz = -((float)((yval * FT_10K)/767));
            *vy = gnd_level(*vx, *vz) + 200.0;
            editobj(obj[STATS]);
            objreplace(tag[LAUNCH]);
            charstr(xtemp);
            cmov2i(170,220);
            charstr(ytemp);
            closeobj();
      }     /* end of MOUSE3 hit */

      /* As long as LEFT MOUSE not selected, keep on displaying current UTM
            grid coordinates in control panel area. */

      if (!launchlock) {
            editobj(obj[STATS]);
            objreplace(tag[LAUNCH]);
            charstr(xtemp);
            cmov2i(170,220);
            charstr(ytemp);
            closeobj();
      }

      /* if RIGHT MOUSE selected lock in target and update control panel. */

      if (getbutton(MOUSE1) && (!getbutton(MOUSE3)) && (!getbutton(MOUSE2))) {
            ringbell();
```

```
        xtarget = xval;
        ytarget = yval;
        targetlock = TRUE;
        editobj(obj[STATS]);
        objreplace(tag[TARGET]);
        charstr(xtemp);
        cmov2i(170,140);
        charstr(ytemp);
        closeobj();
}
/* As long as RIGHT MOUSE not selected keep on displaying current UTM
     grid coordinates in control panel area. */
if (!targetlock) {
        if (launchlock) {
                xdistance = ((double)(xval - xlaunch));
                ydistance = ((double)(yval - ylaunch));
                distance = sqrt((float)(xdistance * xdistance + ydistance * ydistance));
                distance = distance * GRID_FACTOR;
                sprintf(dist,"%5.0f METERS", distance);
                *direction = atan2(ydistance, xdistance);
                if (*direction < 0.0) *direction += TWOPI;
                *compassdir = compass(*direction);

                sprintf(heading,"%d DEGREES", (int)*compassdir);

                editobj(obj[STATS]);
                objreplace(tag[TARGET]);
                charstr(xtemp);
                cmov2i(170,140);
                charstr(ytemp);
                objreplace(tag[HEAD]);
                charstr(heading);
                cmov2i(115,60);
                charstr(dist);
                closeobj();
        }
}

/* if launch position and target location have been selected by the
     operator compute the direction of the missile and distance to target. */

if (launchlock && targetlock) {
        xdistance = ((double)(xtarget - xlaunch));
        ydistance = ((double)(ytarget - ylaunch));
        distance = sqrt((float)((xdistance * xdistance) +
           (ydistance * ydistance)));
        distance = distance * GRID_FACTOR;
        sprintf(dist,"%5.0f METERS", distance);
        *direction = atan2(ydistance, xdistance);
        if (*direction < 0.0) *direction += TWOPI;
        *compassdir = compass(*direction);
```

```
        sprintf(heading,"%d DEGREES", (int)*compassdir);
        editobj(obj[STATS]);
        objreplace(tag[HEAD]);
        charstr(heading);
        cmov2i(115,60);
        charstr(dist);
        closeobj();
}

/* add small red and blue circles to contour map to indicate launch
      position and target location. Connect circles to indicate missile
      flight path */

if (launchlock)
      if (targetlock) {
            editobj(obj[FLTPATH]);
            objreplace(tag[MISSILE]);
            circf((float)(xlaunch)/767.0*100.0, (float)(ylaunch)/767.0*100.0, 0.6);
            move2((float)(xtarget)/767.0*100.0, (float)(ytarget)/767.0*100.0);
            draw2((float)(xlaunch)/767.0*100.0, (float)(ylaunch)/767.0*100.0);
            objreplace(tag[TGT]);
            circf((float)(xtarget)/767.0*100.0, (float)(ytarget)/767.0*100.0, 0.6);
            closeobj();
      }
      else {
            editobj(obj[FLTPATH]);
            objreplace(tag[MISSILE]);
            circf((float)(xlaunch)/767.0*100.0, (float)(ylaunch)/767.0*100.0, 0.6);
            move2((float)(xval)/767.0*100.0, (float)(yval)/767.0*100.0);
            draw2((float)(xlaunch)/767.0*100.0, (float)(ylaunch)/767.0*100.0);
            closeobj();
      }

/* if MIDDLE MOUSE selected, launch has occurred and control transfers
      back to main portion of FOG-M program displaying out-the-window 3-D
      view of the flight area. */

if (getbutton(MOUSE2) && (!getbutton(MOUSE1)) && (!getbutton(MOUSE3))
   && launchlock && targetlock) {
      ringbell();
      while (getbutton(MOUSE2));
      break;
}
```

```
            writemask(SAVEMAP);
            callobj(obj[FLTPATH]);
            writemask(unmask);
            callobj(obj[STATS]);
            swapbuffers();
        }
exit:
        cursoff();
        popmatrix();
}
```

RANDNUM

```
/*  randnum.c - returns a random float between zero and one */

static long seed = 1234567;

randseed(newseed)
long newseed;
{
      seed = newseed;
}


float randnum()
{
      long mult();

      seed = (mult(seed,31415821) + 1) % 100000000;
      return(seed / 100000000.0);
}


long mult(p,q)
long p,q;
{
      long p0, p1, q0, q1;

      p1 = p / 10000;
      p0 = p % 10000;
      q1 = q / 10000;
      q0 = q % 10000;
      return((((p0*q1 + p1*q0) % 10000) * 10000 + p0*q0) % 100000000);
}
```

READCONTROLS

```
/* reads the values from the operator's controls (mouse and dials) */

#include "gl.h"              /* graphics lib defs */
#include "fogm.h"              /* fogm constants */
#include "device.h"           /* device definitions */

read_controls(designate, greyscale, flying, active, speed, direction,
compassdir, alt, pan, tilt, fovy)

int *designate, *greyscale, *flying, *active, *fovy;
float *speed, *compassdir;
double *direction, *pan, *tilt;
Coord *alt;

{
      extern float randx, randy, randz;
      float randnum();
      Colorindex colors[1];

      /* quit if all three mouse buttons are pushed */
      if(getbutton(MOUSE1) && getbutton(MOUSE2) && getbutton(MOUSE3)) {
            *flying = FALSE;
            *active = FALSE;
      }
      else {
            if (getbutton(MOUSE3) && !(getbutton(MOUSE2))) {   /* Zoom In */
                  *fovy = (*fovy < (80 + DELTAFOVY)) ? 80 : *fovy - DELTAFOVY;
            }

            if (getbutton(MOUSE1) && !(getbutton(MOUSE2))) {   /* Zoom Out */
                  *fovy = (*fovy > (550 - DELTAFOVY)) ? 550 : *fovy + DELTAFOVY;
            }

            if (getbutton(MOUSE2)) {              /* designate/reject target */
                  if (*designate) {  /* see if target in sights */
                        /*pushmatrix();
                              pushviewport();
                              pushattributes();
                              viewport(0, 1023, 0, 767);
                              ortho2(0.0, 1023.0, 0.0, 767.0);
                              cmov2s((Scoord)(768/2), (Scoord)(768/2));
                              readpixels(1,colors);
                              if ((colors[0] >= MIN_TGT_COLOR) && (colors[0] <= MAX_TGT_COLOR)) {
                        *designate = FALSE;
                        ringbell();
                        randx = 30.0 * randnum() - 15.0;
                        randy = 10.0 * randnum() -  5.0;
                        randz = 10.0 * randnum();
                        while (getbutton(MOUSE2));
```

```
                /*}
                        popattributes();
                        popviewport();
                        popmatrix(); */
            }
            else {    /* reject currently designated target */
                ringbell();
                *designate = TRUE;
                /* re-adjust tilt and pan values appropriately */ ;
            }
        }

        if (*greyscale != getvaluator(DIAL3)) { /* DIAL3 indicates color change */
            *greyscale = !*greyscale;
            setvaluator(DIAL3,*greyscale,0,1);
            colorramp(*greyscale,FALSE);
        }

        *speed = (float)(getvaluator(DIAL2) / SPEEDSENS); /* get desired speed */
        *alt = (Coord)(getvaluator(DIAL4));

        *pan = DTOR * (double)(-getvaluator(MOUSEX)) / PANSENS;
        *tilt = DTOR * (double)(getvaluator(MOUSEY)) / TILTSENS;

        *compassdir = (float)getvaluator(DIAL0) / DIRSENS;
        /* keep *direction between 0 and 360, update valuator if changed   */
        if (*compassdir >= 360.0) {
            *compassdir -= 360.0;
            setvaluator(DIAL0,(int)(*compassdir*DIRSENS), (int)(-360*DIRSENS),
            (int)(720*DIRSENS));
        }

        if (*compassdir < 0.0) {
            *compassdir += 360.0;
            setvaluator(DIAL0,(int)(*compassdir*DIRSENS), (int)(-360*DIRSENS),
            (int)(720*DIRSENS));
        }

        /*convert *direction from compass degrees to trigonometric radians    */
        *direction = (*compassdir <= 90.0) ? DTOR * (90.0 - *compassdir) :
        DTOR * (450.0 - *compassdir);
    }
}
```

READDATA

```
/* reads the raw 16 bit elevation and vegetation code data
  from the DMA data file and inserts it into the global
  gridpixel array     */

#include "fogm.h"
#include "files.h"

readdata()
{
      int fd;           /* file descriptor for the data file */
      short row, col, rowoffset, coloffset;   /* loop indicies  */
      extern short gridpixel[100][100];       /* DMA elev and veg. data */


      /* read the data from the data file into the gridpixel array */
      fd = open(TERRAIN_FILE,RD);
      lseek(fd,0,0);
      for (coloffset = 0; coloffset < NUMXGRIDS * 10; coloffset += 10) {
            for (rowoffset = 0; rowoffset < NUMZGRIDS*10; rowoffset += 10) {
                  for (col = 0; col < 10; ++col) {
                        for (row = 0; row < 10; ++row) {
                              read(fd,&gridpixel[rowoffset+row][coloffset+col],2);
                        }
                  }
            }
      }
}
```

ROAD_BOUNDS

```
#include "math.h"
#include "fogm.h"

#define X   0
#define Y   1
#define Z   2

#define NONE   0

road_bounds(pt1, pt2, pt3, road_width, left_pt1, right_pt1, left_pt2,
right_pt2, first_xgrid, first_zgrid,  last_xgrid,  last_zgrid)

float pt1[3], pt2[3], pt3[3], road_width;
float left_pt1[3], right_pt1[3], left_pt2[3], right_pt2[3];
int   *first_xgrid, *last_xgrid, *first_zgrid, *last_zgrid;
{
      float delta_x, delta_z, seg_dir, min_x, max_x, min_z, max_z;
      float left_end1[3], right_end1[3], left_start2[3], right_start2[3],
      left_end2[3], right_end2[3];
      int intersection_type;

      /* determine the corner points of the segment */
      delta_x = pt2[X] - pt1[X];
      delta_z = pt2[Z] - pt1[Z];
      seg_dir = atan2(delta_z, delta_x);
      left_end1[X] = pt2[X] + (cos(seg_dir + HALFPI)*road_width/2.0);
      right_end1[X] = pt2[X] + (cos(seg_dir - HALFPI)*road_width/2.0);
      left_end1[Z] = pt2[Z] + (sin(seg_dir + HALFPI)*road_width/2.0);
      right_end1[Z] = pt2[Z] + (sin(seg_dir - HALFPI)*road_width/2.0);

      if ((pt2[X] != pt3[X]) || (pt2[Z] != pt3[Z])) {
            /* we are not working with the final segment of this road, find
                  the intersection of this segment with the next one */
            delta_x = pt3[X] - pt2[X];
            delta_z = pt3[Z] - pt2[Z];
            seg_dir = atan2(delta_z, delta_x);
            left_start2[X] = pt2[X] + (cos(seg_dir + HALFPI)*road_width/2.0);
            right_start2[X] = pt2[X] + (cos(seg_dir - HALFPI)*road_width/2.0);
            left_start2[Z] = pt2[Z] + (sin(seg_dir + HALFPI)*road_width/2.0);
            right_start2[Z] = pt2[Z] + (sin(seg_dir - HALFPI)*road_width/2.0);
            left_end2[X] = pt3[X] + (cos(seg_dir + HALFPI)*road_width/2.0);
            right_end2[X] = pt3[X] + (cos(seg_dir - HALFPI)*road_width/2.0);
            left_end2[Z] = pt3[Z] + (sin(seg_dir + HALFPI)*road_width/2.0);
            right_end2[Z] = pt3[Z] + (sin(seg_dir - HALFPI)*road_width/2.0);

            /* find the intersection point of the left hand sides of the
                  first and second road segments */
            line_intersect2(left_pt1, left_end1, left_start2, left_end2,
            left_pt2, &intersection_type);
```

```
        if (intersection_type == NONE) {
            left_pt2[X] = left_end1[X];
            left_pt2[Z] = left_end1[Z];
        }

        /* find the intersection point of the right hand sides of the
            first and second road segments */
        line_intersect2(right_pt1, right_end1, right_start2, right_end2,
        right_pt2, &intersection_type);
        if (intersection_type == NONE) {
            right_pt2[X] = right_end1[X];
            right_pt2[Z] = right_end1[Z];
        }
    }
    else {
        /* this is the final segment of this road */
        left_pt2[X] = left_end1[X];
        left_pt2[Z] = left_end1[Z];
        right_pt2[X] = right_end1[X];
        right_pt2[Z] = right_end1[Z];
    }

    /* determine the min and max x and z values */
    min_x = left_pt1[X];
    max_x = left_pt1[X];
    min_z = left_pt1[Z];
    max_z = left_pt1[Z];
    if (right_pt1[X] < min_x) min_x = right_pt1[X];
    if (right_pt1[X] > max_x) max_x = right_pt1[X];
    if (right_pt1[Z] < min_z) min_z = right_pt1[Z];
    if (right_pt1[Z] > max_z) max_z = right_pt1[Z];
    if (left_pt2[X] < min_x) min_x = left_pt2[X];
    if (left_pt2[X] > max_x) max_x = left_pt2[X];
    if (left_pt2[Z] < min_z) min_z = left_pt2[Z];
    if (left_pt2[Z] > max_z) max_z = left_pt2[Z];
    if (right_pt2[X] < min_x) min_x = right_pt2[X];
    if (right_pt2[X] > max_x) max_x = right_pt2[X];
    if (right_pt2[Z] < min_z) min_z = right_pt2[Z];
    if (right_pt2[Z] > max_z) max_z = right_pt2[Z];
    *first_xgrid = (int)(min_x/FT_100M);
    *first_zgrid = (int)(min_z/FT_100M);
    *last_xgrid = (int)(max_x/FT_100M);
    *last_zgrid = (int)(max_z/FT_100M);
    if (*first_xgrid < 0) *first_xgrid = 0;
    if (*first_zgrid < 0) *first_zgrid = 0;
    if (*last_xgrid > 98) *last_xgrid = 98;
    if (*last_zgrid > 98) *last_zgrid = 98;
}
```

SORT_ARRAY

```
sort_array(array, num_entries, decending, test_index)
float array[10][3];
int num_entries, decending, test_index;
{
      int i,j;
      float temp[3];

      for (i = 0; i < num_entries; ++i) {
            for (j = i + 1; j <= num_entries; ++j) {

                  if (((decending) && (array[j][test_index] > array[i][test_index])) ||
                     ((!decending) && (array[j][test_index] < array[i][test_index]))) {
                        temp[0] = array[i][0];
                        temp[1] = array[i][1];
                        temp[2] = array[i][2];
                        array[i][0] = array[j][0];
                        array[i][1] = array[j][1];
                        array[i][2] = array[j][2];
                        array[j][0] = temp[0];
                        array[j][1] = temp[1];
                        array[j][2] = temp[2];
                  }
            }
      }
}
```

UP_LOOK_POS

```
/* compute the camera's lookat position     */

#include "fogm.h"       /* fogm constants */
#include "math.h"       /* math routine definitions */
#include "gl.h"        /* graphics definitions  */

update_look_posit(direction, pan, tilt, vx, vy, vz,
tgtx, tgty, tgtz, designate, px, py, pz)

double direction, pan, tilt;
Coord vx, vy, vz, tgtx, tgty, tgtz, *px, *py, *pz;
int designate;
{
      extern int framecnt;
      double lookdir;

      if (designate) {   /* missile is not locked on to a target */

            /* compute direction camera is looking */
            lookdir = direction + pan;

            /* compute a coordinate along camera's line of sight */
            *px = vx + cos(lookdir) * MAXLOOKDIST;
            *pz = vz - sin(lookdir) * MAXLOOKDIST;

            if (framecnt < 15) {
                  *py = 4.0 * vy * (14 - framecnt) / 14.0;
                  framecnt++;
            }
            else  {
                  *py = vy + MAXLOOKDIST * tan(tilt);
            }
      }
      else {
            *px = tgtx;
            *py = tgty;
            *pz = tgtz;
      }

}
```

UP_MSL_POSIT

```
/* Compute new missile position */

#include "gl.h"          /* graphics definitions */
#include "device.h"      /* graphics device definitions */
#include "fogm.h"        /* fogm constants */
#include "math.h"        /* math function declarations */
#include <sys/types.h>   /* contains the time sturcture tms */
#include <sys/times.h>   /* for time calls */

update_missile_posit(direction, compassdir, speed, designate,
tgtx, tgty, tgtz, vx, vy, vz, flying)

double *direction;
float *compassdir;
float speed;
int designate;
Coord tgtx, tgty, tgtz;
int *flying;
Coord *vx, *vy, *vz;

{
      static long seconds;
      static long lastsec = -999;     /* -999 is flag to indicate no value */
      struct tms timestruct;
      float deltadist, gndlevel, gnd_level(), compass(), ht_above_tank;
      long float deltax, deltaz, dist_to_tank;

      seconds = times(&timestruct);

      /* compute distance missile must move ahead to maintain speed */
      if (lastsec  == -999)
            deltadist = 0.0;
      else
            deltadist = (speed/FPS_TO_KTS)*(seconds - lastsec);

      lastsec = seconds;        /* save for next pass */
      if (designate) {   /* missile under operator contol, not locked on tgt */
            *vx += deltadist * cos(*direction);
            *vz -= deltadist * sin(*direction);

            /* keep missile at least 50 ft above ground level */
            gndlevel = gnd_level(*vx, *vz);
            if (*vy < (gndlevel + 50.0)) *vy = gndlevel + 50.0;
      }
      else  {

            deltax = *vx - tgtx;
            deltaz = *vz - tgtz;
            dist_to_tank = hypot(deltax, deltaz);
```

```
        if (deltadist > (float)dist_to_tank) {    /* hit on target */
            deltadist = (float)dist_to_tank - 5.0;
            *flying = FALSE;
            lastsec = -999; /* no value flag for next launch */
        }

        *direction = (double)atan2((float)deltaz, (float)-deltax);
        if (*direction < 0.0) *direction += TWOPI;
        *compassdir = compass(*direction);

        setvaluator(DIAL0,(int)(*compassdir*DIRSENS), (int)(-360*DIRSENS),
        (int)(720*DIRSENS));


        *vx += (deltadist * cos(*direction));
        *vz -= (deltadist * sin(*direction));
        ht_above_tank = (float)*vy - gnd_level(tgtx,tgtz);
        *vy  -= (Coord)((ht_above_tank * deltadist) / (float)dist_to_tank);
    }
}
```

```
#include "fogm.h"
#include "gl.h"
#include "math.h"

view_bounds(vx, vy, vz, px, py, pz, tilt, fovy,
firstxgrid, firstzgrid, lastxgrid, lastzgrid)
Coord vx,vy,vz;
double tilt;
int fovy;
short *firstxgrid, *firstzgrid, *lastxgrid, *lastzgrid;
{
      float ix, iz;    /* the intersection points */
      float lookdir;
      float deltax, deltay, deltaz, delta_alt, fx, fy, fz;
      float half_fovy;
      float lower_edge_angle;

      /* compute the direction the camera is looking */
      lookdir = atan2((float)(vz - pz), (float)(-(vx-px)));
      if (lookdir < 0.0) lookdir += TWOPI;

      if (vy > py) {
            /* tilt angle is negative */
            deltax = px - vx;
            deltay = py - vy;
            deltaz = pz - vz;
            delta_alt = pow((float)MIN, ALTSCALE) - vy;
      }
      else {
            /* tilt angle is positive, use the lower fustrum edge instead
                  of the line of sight to compute the view bounds */
            /* compute a coordinate along the lower fustrum edge */
            half_fovy = ((float)fovy/20.0*DTOR);
            lower_edge_angle = tilt - half_fovy;
            fx = vx + cos(lookdir)*MAXLOOKDIST;
            fz = vz - sin(lookdir)*MAXLOOKDIST;
            fy = vy + tan(lower_edge_angle)*MAXLOOKDIST;
            deltax = fx - vx;
            deltay = fy - vy;
            deltaz = fz - vz;
            delta_alt = pow((float)MIN, ALTSCALE) - vy;
      }

      ix = vx + ((deltax/deltay)*delta_alt);
      iz = vz + ((deltaz/deltay)*delta_alt);


      /* compute which grid objects should be sent through the geometry
           pipeline */
```

```
    if (deltay > 0.0) {
        /* the fustrum is looking totally skyward, don't bother doing
            any terrain */
        *firstxgrid = 0;
        *firstzgrid = 0;
        *lastxgrid = 0;
        *lastzgrid = 0;
    }
    else {
        /* display 20 grid squares on all sides of the intersection point */
        *firstxgrid = (int)(ix/FT_100M) - 20;
        *lastxgrid = (int)(ix/FT_100M) + 20;
        *firstzgrid = (int)(-iz/FT_100M) - 20;
        *lastzgrid = (int)(-iz/FT_100M) + 20;

        /* insure that objects drawn include the current missile position */
        if ((int)(vx/FT_100M) < *firstxgrid)
            *firstxgrid = (int)(vx/FT_100M);
        if ((int)(vx/FT_100M) > *lastxgrid)
            *lastxgrid = (int)(vx/FT_100M);
        if ((int)(-vz/FT_100M) < *firstzgrid)
            *firstzgrid = (int)(-vz/FT_100M);
        if ((int)(-vz/FT_100M) > *lastzgrid)
            *lastzgrid = (int)(-vz/FT_100M);
        if (*firstzgrid < 0) *firstzgrid = 0;
        if (*firstxgrid < 0) *firstxgrid = 0;
        if (*lastzgrid > 98) *lastzgrid = 98;
        if (*lastxgrid > 98) *lastxgrid = 98;
    }
}
```

LIST OF REFERENCES

1. PC Connection advertisement, *PC Magazine,* v. 6, no. 11, p. 241, June 9, 1987.

2. Orlansky, J. and String, J., "Reaping the Benefits of Flight Simulation," in *Computer Image Generation,* edited by B. Schachter, John Wiley & Sons, Inc., New York, New York, 1983.

3. US Army Combat Developments Experimentation Center (USACDEC) Technical Report, *Computer Graphics Fiber Optics Guided Missile Flight Simulator (FOG-M Simulator) Required Instrumentation Capability (RIC),* Fort Ord, California, 1986.

4. Mar, Roland K., "FOG-M: Another Army Orphan for the Marines?" U. S. Naval Institute *Proceedings,* v. 113/6/1012, pp. 95-97, June 1987.

5. Kotas, Jim, "Computer Image Generation: Realistic Simulation," *National Defense,* v. 70, no. 412, pp. 26-31, November 1985.

6. Berthiaume, Richard, Karnavas, Gary, and Bernsteen, Stan, "Graphical Representations of DMA Digital Terrain Data on Low Cost Commercial Graphics Workstation," *Proceedings of the IEEE 1986 National Aerospace and Electronics Conference,* v. 3, pp. 992-996, 1986.

7. Silicon Graphics, Inc., *IRIS User's Guide,* Mountain View, California, 1986.

8. Fox, Teresa A., Clark, Philip D., "Development of Computer-generated Imagery for a Low-cost Real-time Terrain Imaging System," *Proceedings of the IEEE 1986 National Aerospace and Electronics Conference,* v. 3, pp. 986-991, 1986.

9. Defense Mapping Agency, *Product Specifications for Digital Landmass System (DLMS) Data Base,* 2d ed., April 1983.

10. US Army Combat Developments Experimentation Center (USACDEC) Technical Report, *Fort Hunter Liggett Digital Terrain Database on the VAX Computer,* Fort Ord, California, 1985.

11. Hearn, Donald, and Baker, M. Pauline, *Computer Graphics,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.

12. McGrew, J. F., "Exaggerated Vertical Scale in CGI Terrain Perspectives," *Proceedings of the Human Factors Society 27th Annual Meeting,* v. 1, pp. 33-35, 1983.

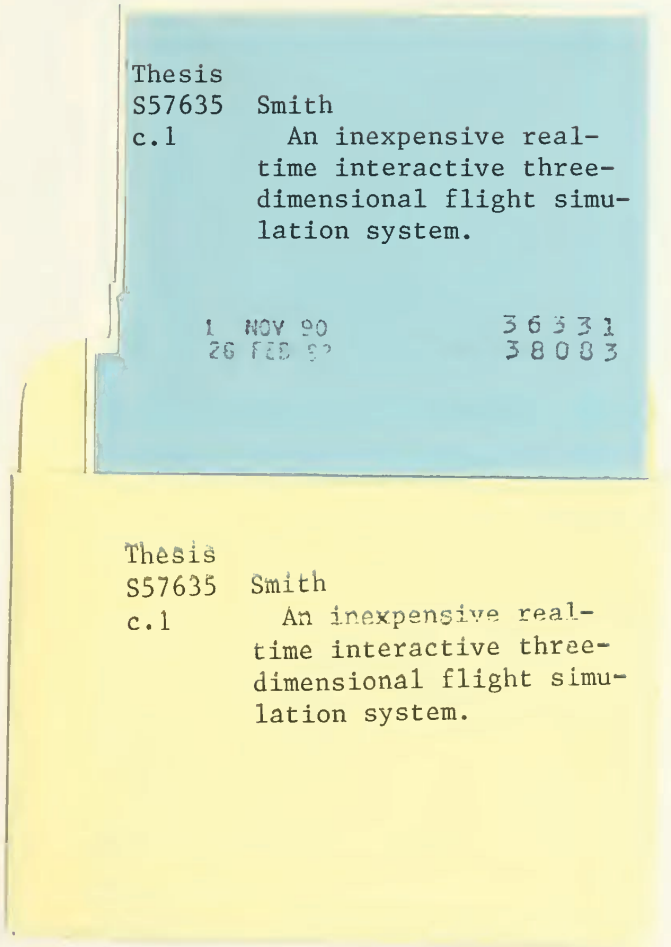
13 Fuchs, Henry, Abram, Gregory D., and Grant, Eric D., "Near Real-Time Shaded Display of Rigid Objects," *Computer Graphics,* v. 17, no. 3, pp. 65-72, July 1983.

14. Sedgewick, Robert, *Algorithms,* Addison-Wesley Publishing Co., Reading, Massachusetts, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, DC 20350-2000	2
3.	Commanant (G-PTE) United States Coast Guard 2100 Second Street SW Washington, DC 20593	2
4.	Superintendent Attn: Library (Code 0142) Naval Postgraduate School Monterey, California 93943-5002	2
5.	Chairman (Code 52) Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
6.	Computer Technology Curricular Officer (Code 37) Naval Postgraduate School Monterey, California 93943	1
7.	Michael J. Zyda (Code 52Zk) Department of Computer Science Naval Postgraduate School Monterey, California 93943	2

8.	Robert B. McGhee (Code 52Mz) Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
9.	Captain Douglas B. Smith Headquarters, United States Marine Corps Code CCA Washington, DC 20380	5
10.	Lieutenant Dale G. Streyle CG EECEN (Computer Systems Branch) Wildwood, New Jersey 08260	3


Thesis
S57635 Smith
c.1 An inexpensive real-
time interactive three-
dimensional flight simu-
lation system.
1 NOV 90 36331
26 FEB 92 38083
Thesis
S57635 Smith
c.1 An inexpensive real-
time interactive three-
dimensional flight simu-
lation system.