Appendix A.1 - Rendering Code

```
/****
  rend.c
  Version .08
  Copyright (C) 1997 Colin Bulthaup
****/

void mat_v_mul_4( float mat[4][4], float v[4], float dest[4]) {
  dest[0] = (mat[0][0]*v[0] + mat[0][1]*v[1]
                           + mat[0][2]*v[2] + mat[0][3]*v[3]);
  dest[1] = (mat[1][0]*v[0] + mat[1][1]*v[1]
                           + mat[1][2]*v[2] + mat[1][3]*v[3]);
  dest[2] = (mat[2][0]*v[0] + mat[2][1]*v[1]
                           + mat[2][2]*v[2] + mat[2][3]*v[3]);
  dest[3] = (mat[3][0]*v[0] + mat[3][1]*v[1]
                           + mat[3][2]*v[2] + mat[3][3]*v[3]);
}

void mat_mat_mul_4( float mat1[4][4], float mat2[4][4], float dest[4][4]) {
  dest[0][0] = (mat2[0][0]*mat1[0][0] + mat2[1][0]*mat1[0][1]
                           + mat2[2][0]*mat1[0][2] + mat2[3][0]*mat1[0][3]);
  dest[0][1] = (mat2[0][1]*mat1[0][0] + mat2[1][1]*mat1[0][1]
                           + mat2[2][1]*mat1[0][2] + mat2[3][1]*mat1[0][3]);
  dest[0][2] = (mat2[0][2]*mat1[0][0] + mat2[1][2]*mat1[0][1]
                           + mat2[2][2]*mat1[0][2] + mat2[3][2]*mat1[0][3]);
  dest[0][3] = (mat2[0][3]*mat1[0][0] + mat2[1][3]*mat1[0][1]
                           + mat2[2][3]*mat1[0][2] + mat2[3][3]*mat1[0][3]);
  dest[1][0] = (mat2[0][0]*mat1[1][0] + mat2[1][0]*mat1[1][1]
                           + mat2[2][0]*mat1[1][2] + mat2[3][0]*mat1[1][3]);
  dest[1][1] = (mat2[0][1]*mat1[1][0] + mat2[1][1]*mat1[1][1]
                           + mat2[2][1]*mat1[1][2] + mat2[3][1]*mat1[1][3]);
  dest[1][2] = (mat2[0][2]*mat1[1][0] + mat2[1][2]*mat1[1][1]
                           + mat2[2][2]*mat1[1][2] + mat2[3][2]*mat1[1][3]);
  dest[1][3] = (mat2[0][3]*mat1[1][0] + mat2[1][3]*mat1[1][1]
                           + mat2[2][3]*mat1[1][2] + mat2[3][3]*mat1[1][3]);
  dest[2][0] = (mat2[0][0]*mat1[2][0] + mat2[1][0]*mat1[2][1]
                           + mat2[2][0]*mat1[2][2] + mat2[3][0]*mat1[2][3]);
  dest[2][1] = (mat2[0][1]*mat1[2][0] + mat2[1][1]*mat1[2][1]
                           + mat2[2][1]*mat1[2][2] + mat2[3][1]*mat1[2][3]);
  dest[2][2] = (mat2[0][2]*mat1[2][0] + mat2[1][2]*mat1[2][1]
                           + mat2[2][2]*mat1[2][2] + mat2[3][2]*mat1[2][3]);
  dest[2][3] = (mat2[0][3]*mat1[2][0] + mat2[1][3]*mat1[2][1]
                           + mat2[2][3]*mat1[2][2] + mat2[3][3]*mat1[2][3]);
  dest[3][0] = (mat2[0][0]*mat1[3][0] + mat2[1][0]*mat1[3][1]
                           + mat2[2][0]*mat1[3][2] + mat2[3][0]*mat1[3][3]);
  dest[3][1] = (mat2[0][1]*mat1[3][0] + mat2[1][1]*mat1[3][1]
                           + mat2[2][1]*mat1[3][2] + mat2[3][1]*mat1[3][3]);
  dest[3][2] = (mat2[0][2]*mat1[3][0] + mat2[1][2]*mat1[3][1]
                           + mat2[2][2]*mat1[3][2] + mat2[3][2]*mat1[3][3]);
  dest[3][3] = (mat2[0][3]*mat1[3][0] + mat2[1][3]*mat1[3][1]
                           + mat2[2][3]*mat1[3][2] + mat2[3][3]*mat1[3][3]);
}

void cross_4( float v1[3], float v2[3], float dest[3]) {
  dest[0] = v1[1]*v2[2] - v2[1]*v1[2];
  dest[1] = v2[0]*v1[2] - v1[0]*v2[2];
  dest[2] = v1[0]*v2[1] - v2[0]*v1[1];
}

float vlen_4( float v[4]) {
  return( sqrt( v[0]*v[0] + v[1]*v[1] + v[2]*v[2]));
}

float vlen_3( float v[3]) {
  return( sqrt( v[0]*v[0] + v[1]*v[1] + v[2]*v[2]));
}
```

```
float normalize_3( float v[3]) {
  float len = vlen_3( v);
  v[0] /= len;
  v[1] /= len;
  v[2] /= len;
}

void generate_T( float c, float v[4], float dest[4][4]) {
  dest[0][0] = 1;
  dest[0][1] = 0;
  dest[0][2] = 0;
  dest[0][3] = c*v[0];
  dest[1][0] = 0;
  dest[1][1] = 1;
  dest[1][2] = 0;
  dest[1][3] = c*v[1];
  dest[2][0] = 0;
  dest[2][1] = 0;
  dest[2][2] = 1;
  dest[2][3] = c*v[2];
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_Rper( float VPN[4], float VUP[4], float dest[4][4]) {
  float Rz[4], Rx[4], Ry[4];
  float tmp;

  tmp = vlen_4( VPN);
  Rz[0] = VPN[0]/tmp;
  Rz[1] = VPN[1]/tmp;
  Rz[2] = VPN[2]/tmp;

  cross_4( VUP, Rz, Rx);
  tmp = vlen_4( Rx);
  Rx[0] = Rx[0]/tmp;
  Rx[1] = Rx[1]/tmp;
  Rx[2] = Rx[2]/tmp;

  cross_4( Rz, Rx, Ry);

  dest[0][0] = Rx[0];
  dest[0][1] = Rx[1];
  dest[0][2] = Rx[2];
  dest[0][3] = 0;
  dest[1][0] = Ry[0];
  dest[1][1] = Ry[1];
  dest[1][2] = Ry[2];
  dest[1][3] = 0;
  dest[2][0] = Rz[0];
  dest[2][1] = Rz[1];
  dest[2][2] = Rz[2];
  dest[2][3] = 0;
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_SH( float DOP[4], float dest[4][4]) {
  dest[0][0] = 1;
  dest[0][1] = 0;
  dest[0][2] = -1 * (DOP[0] / DOP[2]);
  dest[0][3] = 0;
  dest[1][0] = 0;
  dest[1][1] = 1;
  dest[1][2] = -1 * (DOP[1] / DOP[2]);
  dest[1][3] = 0;
  dest[2][0] = 0;
  dest[2][1] = 0;
```

```
  dest[2][2] = 1;
  dest[2][3] = 0;
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_Rz( float theta, float dest[4][4]) {
  dest[0][0] = cos( theta);
  dest[0][1] = -1*sin( theta);
  dest[0][2] = 0;
  dest[0][3] = 0;
  dest[1][0] = sin( theta);
  dest[1][1] = cos( theta);
  dest[1][2] = 0;
  dest[1][3] = 0;
  dest[2][0] = 0;
  dest[2][1] = 0;
  dest[2][2] = 1;
  dest[2][3] = 0;
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_Rx( float theta, float dest[4][4]) {
  dest[0][0] = 1;
  dest[0][1] = 0;
  dest[0][2] = 0;
  dest[0][3] = 0;
  dest[1][0] = 0;
  dest[1][1] = cos( theta);
  dest[1][2] = -1*sin( theta);
  dest[1][3] = 0;
  dest[2][0] = 0;
  dest[2][1] = sin( theta);
  dest[2][2] = cos( theta);
  dest[2][3] = 0;
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_Ry( float theta, float dest[4][4]) {
  dest[0][0] = cos( theta);
  dest[0][1] = 0;
  dest[0][2] = sin( theta);
  dest[0][3] = 0;
  dest[1][0] = 0;
  dest[1][1] = 1;
  dest[1][2] = 0;
  dest[1][3] = 0;
  dest[2][0] = -1*sin( theta);
  dest[2][1] = 0;
  dest[2][2] = cos( theta);
  dest[2][3] = 0;
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_S( float x, float y, float z, float dest[4][4]) {
  dest[0][0] = x;
  dest[0][1] = 0;
  dest[0][2] = 0;
  dest[0][3] = 0;
  dest[1][0] = 0;
  dest[1][1] = y;
```

```
  dest[1][2] = 0;
  dest[1][3] = 0;
  dest[2][0] = 0;
  dest[2][1] = 0;
  dest[2][2] = z;
  dest[2][3] = 0;
  dest[3][0] = 0;
  dest[3][1] = 0;
  dest[3][2] = 0;
  dest[3][3] = 1;
}

void generate_M( float xmax, float xmin, float ymax, float ymin,
                                  float zmax, float zmin, float dest[4][4]) {
  float Mper[4][4];
  float T111[4][4];
  float Txyz[4][4];
  float xyz[3];
  float S[4][4];

  Mper[0][0] = 1;      Mper[1][0] = 0;
  Mper[2][0] = 0;      Mper[3][0] = 0;
  Mper[0][1] = 0;      Mper[1][1] = 1;
  Mper[2][1] = 0;      Mper[3][1] = 0;
  Mper[0][2] = 0;      Mper[1][2] = 0;
  Mper[2][2] = 1;      Mper[3][2] = 1;
  Mper[0][3] = 0;      Mper[1][3] = 0;
  Mper[2][3] = -1;      Mper[3][3] = 1;

  T111[0][0] = 1;      T111[1][0] = 0;
  T111[2][0] = 0;      T111[3][0] = 1;
  T111[0][1] = 0;      T111[1][1] = 1;
  T111[2][1] = 0;      T111[3][1] = 1;
  T111[0][2] = 0;      T111[1][2] = 0;
  T111[2][2] = 1;      T111[3][2] = 1;
  T111[0][3] = 0;      T111[1][3] = 0;
  T111[2][3] = 0;      T111[3][3] = 1;

  xyz[0] = xmin;
  xyz[1] = ymin;
  xyz[2] = zmin;

  generate_T( 1, xyz, Txyz);
  generate_S( (xmax-xmin)/2, (ymax-ymin)/2, (zmax-zmin), S);

  mat_mat_mul_4( T111, Mper, dest);
  mat_mat_mul_4( S, dest, dest);
  mat_mat_mul_4( Txyz, dest, dest);
}

void generate_N( float VRP[4], float VPN[4], float VUP[4],
                                  float PRP[4], float umax, float umin, float vmax,
                                  float vmin, float B, float F, float dest[4][4]) {
  float Tvrp[4][4];
  float R[4][4];
  float Tprp[4][4];
  float SHpar[4][4];
  float Sper[4][4];
  float VRP2[4];
  float tmp[4] = {0,0,0,1};
  float DOP[4];

  DOP[0] = (umax+umin)/2 - PRP[0];
  DOP[1] = (vmax+vmin)/2 - PRP[1];
  DOP[2] = -1 * PRP[2];

  generate_T( -1, VRP, Tvrp);
  generate_T( -1, PRP, Tprp);
  generate_Rper( VPN, VUP, R);
  generate_SH( DOP, SHpar);

  mat_v_mul_4( Tprp, tmp, tmp);
```

```
  mat_v_mul_4( SHpar, tmp, VRP2);

  generate_S( 2*VRP2[2]/((umax-umin)*(VRP2[2]+B)),
              2*VRP2[2]/((vmax-vmin)*(VRP2[2]+B)),
              -1/(VRP2[2]+B), Sper);

  mat_mat_mul_4( R, Tvrp, dest);
  mat_mat_mul_4( Tprp, dest, dest);
  mat_mat_mul_4( SHpar, dest, dest);
  mat_mat_mul_4( Sper, dest, dest);
}

void pp_mat_4( float mat[4][4]) {
  printf( "| %f %f %f %f |\n", mat[0][0], mat[0][1], mat[0][2], mat[0][3]);
  printf( "| %f %f %f %f |\n", mat[1][0], mat[1][1], mat[1][2], mat[1][3]);
  printf( "| %f %f %f %f |\n", mat[2][0], mat[2][1], mat[2][2], mat[2][3]);
  printf( "| %f %f %f %f |\n", mat[3][0], mat[3][1], mat[3][2], mat[3][3]);
}

void pp_v_4( float v[4]) {
  printf( "| %f %f %f %f |\n", v[0], v[1], v[2], v[3]);
}

void pp_v_3( float v[4]) {
  printf( "| %f %f %f |\n", v[0], v[1], v[2]);
}

void setup_PerMat() {
  float VRP[4] = { 0, 0, 0, 1};
  float VPN[4] = { 0, 0, 1, 1};
  float VUP[4] = { 0, 1, 0, 1};
  float PRP[4] = { 0, 0, 10, 1};
  float umax = 1;
  float umin = -1;
  float vmax = 1;
  float vmin = -1;
  float F = 0;
  float B = -1.2;
  float xmax = DEFAULT_WIDTH;
  float xmin = 0;
  float ymax = DEFAULT_HEIGHT;
  float ymin = 0;
  float zmax = 400;
  float zmin = 0;

  generate_N( VRP, VPN, VUP, PRP, umax, umin, vmax, vmin,
              B, F, Nper);
  generate_M( xmax, xmin, ymax, ymin, zmax, zmin, Mvv3dv);
  mat_mat_mul_4( Mvv3dv, Nper, PerMat);
  (void)up_fuck_transform( 0);

}

void update_transform( float clock_step) {
  float tmp_mat[4][4];
  float tmp_v[4] = {0,0,-1,0};
  generate_Rz( clock_step, TransMat);
  generate_Ry( clock_step, tmp_mat);
  mat_mat_mul_4( tmp_mat, TransMat, TransMat);
  generate_T( 1, tmp_v, tmp_mat);
  mat_mat_mul_4( tmp_mat, TransMat, TransMat);
}

void calculate_normals( real_polygon *scene_polys, int npolys) {
  int cpoly;
  float v1[4], v2[4];
  float points[3][4];
  float tmp;

  for( cpoly=0; cpoly < npolys; cpoly++) {
    mat_v_mul_4( TransMat, &(scene_polys[cpoly].points[0][0]), &points[0][0]);
    mat_v_mul_4( TransMat, &(scene_polys[cpoly].points[1][0]), &points[1][0]);
```

```
    mat_v_mul_4( TransMat, &(scene_polys[cpoly].points[2][0]), &points[2][0]);
    v1[0] = points[1][0]-points[0][0];
    v1[1] = points[1][1]-points[0][1];
    v1[2] = points[1][2]-points[0][2];
    v2[0] = points[2][0]-points[0][0];
    v2[1] = points[2][1]-points[0][1];
    v2[2] = points[2][2]-points[0][2];
    scene_polys[cpoly].normal[0] = v1[1]*v2[2]-v2[1]*v1[2];
    scene_polys[cpoly].normal[1] = v1[2]*v2[0]-v2[2]*v1[0];
    scene_polys[cpoly].normal[2] = v1[0]*v2[1]-v2[0]*v1[1];
    normalize_3( scene_polys[cpoly].normal);
  }
}

void calculate_lighting( real_polygon *scene_polys, int npolys,
                         point_light *scene_lights, int nlights) {
  int cpoly, clight;
  float tmp;

  for( cpoly=0; cpoly < npolys; cpoly++) {
    scene_polys[cpoly].intensity = AMBIENT_INTENSITY * AMBIENT_K;
    for( clight = 0; clight < nlights; clight++) {
      tmp = scene_lights[clight].vect[0] * scene_polys[cpoly].normal[0] +
            scene_lights[clight].vect[1] * scene_polys[cpoly].normal[1] +
            scene_lights[clight].vect[2] * scene_polys[cpoly].normal[2];
      if( tmp > 0)
            scene_polys[cpoly].intensity += tmp * DIFFUSE_K *
              scene_lights[clight].intensity;
    }
    if( scene_polys[cpoly].intensity > 1)
      scene_polys[cpoly].intensity = 1;
  }
}

void render_image( real_polygon *scene_polys, int npolys, int width,
                   int height, unsigned short *image_ptr)
{
  unsigned short *scanning_image_ptr;
  float v1[4], v2[4], v3[4];
  int v1_int[3], v2_int[3], v3_int[3];
  float *ymax_vect, *ymid_vect, *ymin_vect;
  int scanline;
  int scanx;
  image_polygon image_polys[ NUM_POLYS];
  image_polygon_linked_list *(ysorted_polys[ DEFAULT_HEIGHT + 1]);
  image_polygon_linked_list *(xsorted_polys[ DEFAULT_WIDTH + 1]);
  image_polygon_linked_list *current_polys = NULL;
  image_polygon_linked_list *traverse_ptr;
  image_polygon_linked_list *tmp_ptr;
  image_polygon_linked_list *zsorted_polys = NULL;
  image_polygon_linked_list extra_linked_lists_y[ 2*NUM_POLYS];
  image_polygon_linked_list extra_linked_lists_x[ 2*NUM_POLYS];
  image_polygon_linked_list extra_linked_lists_z[ 2*NUM_POLYS];
  image_polygon_linked_list extra_linked_lists_current[ 2*NUM_POLYS];
  image_polygon_linked_list **linked_ptr;
  image_polygon_linked_list **tmp_linked_ptr;
  int elly = 0;
  int ellx = 0;
  int ellz = 0;
  int ellc = 0;
  int tmp_int;
  int cpoly;
  int count = 0;
  int ready_to_insert = 0;

  for( tmp_int = 0; tmp_int < height + 1; tmp_int++)
    ysorted_polys[ tmp_int] = NULL;
  for( tmp_int = 0; tmp_int < width + 1; tmp_int++)
    xsorted_polys[ tmp_int] = NULL;

  mat_mat_mul_4( PerMat, TransMat, FinalMat);
```

```
scanning_image_ptr = image_ptr;

for( cpoly = 0; cpoly < npolys; cpoly++) {
  mat_v_mul_4( FinalMat, &scene_polys[cpoly].points[0][0], v1);
  mat_v_mul_4( FinalMat, &scene_polys[cpoly].points[1][0], v2);
  mat_v_mul_4( FinalMat, &scene_polys[cpoly].points[2][0], v3);

  v1[0] = (int)(v1[0]/v1[3]); v1[1] = (int)(v1[1]/v1[3]); v1[2] /= v1[3];
  v2[0] = (int)(v2[0]/v2[3]); v2[1] = (int)(v2[1]/v2[3]); v2[2] /= v2[3];
  v3[0] = (int)(v3[0]/v3[3]); v3[1] = (int)(v3[1]/v3[3]); v3[2] /= v3[3];

  if( v1[1] <= v2[1]) {
    if( v2[1] <= v3[1]) {
        ymin_vect = v1;
        ymid_vect = v2;
        ymax_vect = v3;
    } else {
        if( v1[1] <= v3[1]) {
          ymin_vect = v1;
          ymid_vect = v3;
          ymax_vect = v2;
        } else {
          ymin_vect = v3;
          ymid_vect = v1;
          ymax_vect = v2;
        }
    }
  } else {
    if( v1[1] <= v3[1]) {
        ymin_vect = v2;
        ymid_vect = v1;
        ymax_vect = v3;
    } else {
        if( v2[1] <= v3[1]) {
          ymin_vect = v2;
          ymid_vect = v3;
          ymax_vect = v1;
        } else {
          ymin_vect = v3;
          ymid_vect = v2;
          ymax_vect = v1;
        }
    }
  }

  image_polys[ cpoly].pid = cpoly;
  image_polys[ cpoly].ystart = ymin_vect[1];
  image_polys[ cpoly].yend = ymax_vect[1];
  image_polys[ cpoly].ymiddle = ymid_vect[1];
  if( ((ymid_vect[0] - ymin_vect[0]) /
         (ymid_vect[1] - ymin_vect[1])) <
        ((ymax_vect[0] - ymin_vect[0]) /
         (ymax_vect[1] - ymin_vect[1]))) {
    if( ymid_vect[1] - ymin_vect[1] < .5) {
        image_polys[ cpoly].dx1[0] = ymid_vect[0] - ymin_vect[0];
        image_polys[ cpoly].dz1[0] = (ymid_vect[2] - ymin_vect[2]);
        image_polys[ cpoly].scanx[0] = ymid_vect[0];
        image_polys[ cpoly].scanx[1] = ymin_vect[0];
        image_polys[ cpoly].scanz[0] = ymid_vect[2];
        image_polys[ cpoly].scanz[1] = ymin_vect[2];
    } else {
        image_polys[ cpoly].dx1[0] = (ymid_vect[0] - ymin_vect[0]) /
          (int)(ymid_vect[1] - ymin_vect[1]);
        image_polys[ cpoly].dz1[0] = (ymid_vect[2] - ymin_vect[2]) /
          (int)(ymid_vect[1] - ymin_vect[1]);
        image_polys[ cpoly].scanx[0] = ymin_vect[0];
        image_polys[ cpoly].scanx[1] = ymin_vect[0];
        image_polys[ cpoly].scanz[0] = ymin_vect[2];
        image_polys[ cpoly].scanz[1] = ymin_vect[2];
    }
    if( ymax_vect[1] - ymid_vect[1] < .5) {
        image_polys[ cpoly].dx1[1] = ymax_vect[0] - ymid_vect[0];
```

```
            image_polys[ cpoly].dz1[1] = ymax_vect[2] - ymid_vect[2];
  } else {
            image_polys[ cpoly].dx1[1] = (ymax_vect[0] - ymid_vect[0]) /
              (int)(ymax_vect[1] - ymid_vect[1]);
            image_polys[ cpoly].dz1[1] = (ymax_vect[2] - ymid_vect[2]) /
              (int)(ymax_vect[1] - ymid_vect[1]);
  }
  if( ymax_vect[1] - ymin_vect[1] < .5) {
            image_polys[ cpoly].dx2[0] = ymax_vect[0] - ymin_vect[0];
            image_polys[ cpoly].dz2[0] = ymax_vect[2] - ymin_vect[2];
  } else {
            image_polys[ cpoly].dx2[0] = (ymax_vect[0] - ymin_vect[0]) /
              (int)(ymax_vect[1] - ymin_vect[1]);
            image_polys[ cpoly].dz2[0] = (ymax_vect[2] - ymin_vect[2]) /
              (int)(ymax_vect[1] - ymin_vect[1]);
  }
  image_polys[ cpoly].dx2[1] = image_polys[ cpoly].dx2[0];
  image_polys[ cpoly].dz2[1] = image_polys[ cpoly].dz2[0];
} else {
  if( ymid_vect[1] - ymin_vect[1] < .5) {
            image_polys[ cpoly].dx2[0] = ymid_vect[0] - ymin_vect[0];
            image_polys[ cpoly].dz2[0] = (ymid_vect[2] - ymin_vect[2]);
            image_polys[ cpoly].scanx[1] = ymid_vect[0];
            image_polys[ cpoly].scanx[0] = ymin_vect[0];
            image_polys[ cpoly].scanz[1] = ymid_vect[2];
            image_polys[ cpoly].scanz[0] = ymin_vect[2];
  } else {
            image_polys[ cpoly].dx2[0] = (ymid_vect[0] - ymin_vect[0]) /
              (int)(ymid_vect[1] - ymin_vect[1]);
            image_polys[ cpoly].dz2[0] = (ymid_vect[2] - ymin_vect[2]) /
              (int)(ymid_vect[1] - ymin_vect[1]);
            image_polys[ cpoly].scanx[0] = ymin_vect[0];
            image_polys[ cpoly].scanx[1] = ymin_vect[0];
            image_polys[ cpoly].scanz[0] = ymin_vect[2];
            image_polys[ cpoly].scanz[1] = ymin_vect[2];
  }
  if( ymax_vect[1] - ymid_vect[1] < .5) {
            image_polys[ cpoly].dx2[1] = ymax_vect[0] - ymid_vect[0];
            image_polys[ cpoly].dz2[1] = ymax_vect[2] - ymid_vect[2];
  } else {
            image_polys[ cpoly].dx2[1] = (ymax_vect[0] - ymid_vect[0]) /
              (int)(ymax_vect[1] - ymid_vect[1]);
            image_polys[ cpoly].dz2[1] = (ymax_vect[2] - ymid_vect[2]) /
              (int)(ymax_vect[1] - ymid_vect[1]);
  }
  if( ymax_vect[1] - ymin_vect[1] < .5) {
            image_polys[ cpoly].dx1[0] = ymax_vect[0] - ymin_vect[0];
            image_polys[ cpoly].dz1[0] = ymax_vect[2] - ymin_vect[2];
  } else {
            image_polys[ cpoly].dx1[0] = (ymax_vect[0] - ymin_vect[0]) /
              (int)(ymax_vect[1] - ymin_vect[1]);
            image_polys[ cpoly].dz1[0] = (ymax_vect[2] - ymin_vect[2]) /
              (int)(ymax_vect[1] - ymin_vect[1]);
  }
  image_polys[ cpoly].dx1[1] = image_polys[ cpoly].dx1[0];
  image_polys[ cpoly].dz1[1] = image_polys[ cpoly].dz1[0];
}
image_polys[ cpoly].color = (float)32*scene_polys[ cpoly].intensity;
/*    image_polys[ cpoly].color = image_polys[ cpoly].color<<10 +
  image_polys[ cpoly].color<<5 +
  image_polys[ cpoly].color; */

if( (image_polys[ cpoly].ystart < height) &&
            (image_polys[ cpoly].yend > 0)) {
  if( (int)image_polys[ cpoly].ystart >= 0) {
            linked_ptr = &ysorted_polys[ (int)image_polys[ cpoly].ystart];
            while( *linked_ptr != NULL) {
              linked_ptr = &(*linked_ptr)->cdr; }
            (*linked_ptr) = &extra_linked_lists_y[ elly++];
            (*linked_ptr)->car = &image_polys[ cpoly];
            (*linked_ptr)->cdr = NULL;
  } else {
```

```
              linked_ptr = &ysorted_polys[ 0];
              while( *linked_ptr != NULL) {
                linked_ptr = &(*linked_ptr)->cdr; }
              (*linked_ptr) = &extra_linked_lists_y[ elly++];
              (*linked_ptr)->car = &image_polys[ cpoly];
              (*linked_ptr)->cdr = NULL;
    }
    if( (int)image_polys[ cpoly].yend <= height) {
              linked_ptr = &ysorted_polys[ (int)image_polys[ cpoly].yend];
              while( *linked_ptr != NULL) {
                linked_ptr = &(*linked_ptr)->cdr; }
              (*linked_ptr) = &extra_linked_lists_y[ elly++];
              (*linked_ptr)->car = &image_polys[ cpoly];
              (*linked_ptr)->cdr = NULL;
    } else {
              linked_ptr = &ysorted_polys[ height + 1];
              while( *linked_ptr != NULL) {
                linked_ptr = &(*linked_ptr)->cdr; }
              (*linked_ptr) = &extra_linked_lists_y[ elly++];
              (*linked_ptr)->car = &image_polys[ cpoly];
              (*linked_ptr)->cdr = NULL;
    }
  }
}

for( scanline = 0; scanline < height; scanline++) {
  traverse_ptr = ysorted_polys[ scanline];
  while( traverse_ptr != NULL) {
    linked_ptr = &current_polys;
    while( 1) {
              if( *linked_ptr == NULL) {
                (*linked_ptr) = &extra_linked_lists_current[ ellc++];
                (*linked_ptr)->car = traverse_ptr->car;
                (*linked_ptr)->cdr = NULL;
                break;
              }
              if( (*linked_ptr)->car->pid == traverse_ptr->car->pid) {
                *linked_ptr = (*linked_ptr)->cdr;
                break;
              }
              linked_ptr = &(*linked_ptr)->cdr;
    }
    traverse_ptr = traverse_ptr->cdr;
  }
  traverse_ptr = current_polys;

  while( traverse_ptr != NULL) {
    if( traverse_ptr->car->ymiddle > scanline) {
              traverse_ptr->car->scanx[0] += traverse_ptr->car->dx1[0];
              traverse_ptr->car->scanx[1] += traverse_ptr->car->dx2[0];
              traverse_ptr->car->scanz[0] += traverse_ptr->car->dz1[0];
              traverse_ptr->car->scanz[1] += traverse_ptr->car->dz2[0];
    } else {
              traverse_ptr->car->scanx[0] += traverse_ptr->car->dx1[1];
              traverse_ptr->car->scanx[1] += traverse_ptr->car->dx2[1];
              traverse_ptr->car->scanz[0] += traverse_ptr->car->dz1[1];
              traverse_ptr->car->scanz[1] += traverse_ptr->car->dz2[1];
    }
    for( tmp_int = 0; tmp_int < 2; tmp_int++) {
              if( (int)traverse_ptr->car->scanx[ tmp_int] >= 0) {
                if( (int)traverse_ptr->car->scanx[ tmp_int] <= width) {

                  linked_ptr = &xsorted_polys[ (int)traverse_ptr->car->scanx[tmp_int]];
                  while( 1) {
                    if( *linked_ptr == NULL) {
                              (*linked_ptr) = &extra_linked_lists_x[ ellx++];
                              (*linked_ptr)->car = traverse_ptr->car;
                              (*linked_ptr)->cdr = NULL;
                              break;
                    }
                    if( (*linked_ptr)->car->pid == traverse_ptr->car->pid) {
                              *linked_ptr = (*linked_ptr)->cdr;
```

```
                break;
          }
          linked_ptr = &(*linked_ptr)->cdr;
        }
      } else {

        linked_ptr = &xsorted_polys[ width];
        while( 1) {
          if( *linked_ptr == NULL) {
                (*linked_ptr) = &extra_linked_lists_x[ ellx++];
                (*linked_ptr)->car = traverse_ptr->car;
                (*linked_ptr)->cdr = NULL;
                break;
          }
          if( (*linked_ptr)->car->pid == traverse_ptr->car->pid) {
                *linked_ptr = (*linked_ptr)->cdr;
                break;
          }
          linked_ptr = &(*linked_ptr)->cdr;
        }
      }
    } else {

      linked_ptr = &xsorted_polys[ 0];
      while( 1) {
        if( *linked_ptr == NULL) {
          (*linked_ptr) = &extra_linked_lists_x[ ellx++];
          (*linked_ptr)->car = traverse_ptr->car;
          (*linked_ptr)->cdr = NULL;
          break;
        }
        if( (*linked_ptr)->car->pid == traverse_ptr->car->pid) {
          *linked_ptr = (*linked_ptr)->cdr;
          break;
        }
        linked_ptr = &(*linked_ptr)->cdr;
      }
    }
  }
  traverse_ptr = traverse_ptr->cdr;

}

for( scanx = 0; scanx < width; scanx++) {
  traverse_ptr = xsorted_polys[ scanx];

  while( traverse_ptr != NULL) {
    ready_to_insert = 0;
    linked_ptr = &zsorted_polys;
    while( 1) {
      if( *linked_ptr == NULL) {
        if( ready_to_insert) {
          tmp_ptr = *tmp_linked_ptr;
          *tmp_linked_ptr = &extra_linked_lists_z[ ellz++];
          (*tmp_linked_ptr)->car = traverse_ptr->car;
          (*tmp_linked_ptr)->cdr = tmp_ptr;
        } else {
          (*linked_ptr) = &extra_linked_lists_z[ ellz++];
          (*linked_ptr)->car = traverse_ptr->car;
          (*linked_ptr)->cdr = NULL;
        }
        break;
      }
      if( (*linked_ptr)->car->pid == traverse_ptr->car->pid) {
        *linked_ptr = (*linked_ptr)->cdr;
        break;
      }
      if( !((int)traverse_ptr->car->scanx[0] ==
                (int)(*linked_ptr)->car->scanx[1])) {
        if( ((int)traverse_ptr->car->scanx[0] ==
                 (int)(*linked_ptr)->car->scanx[0]) &&
                ((int)traverse_ptr->car->scanz[0] ==
```

```
                 (int)(*linked_ptr)->car->scanz[0])) {
          if( ((traverse_ptr->car->scanz[1] -
                   traverse_ptr->car->scanz[0]) /
                  (traverse_ptr->car->scanx[1] -
                   traverse_ptr->car->scanx[0])) >
                 (((*linked_ptr)->car->scanz[1] -
                   (*linked_ptr)->car->scanz[0]) /
                  ((*linked_ptr)->car->scanx[1] -
                   (*linked_ptr)->car->scanx[0]))) {
                tmp_linked_ptr = linked_ptr;
                ready_to_insert = 1;
          }
         } else {
          if( traverse_ptr->car->scanx[0] < traverse_ptr->car->scanx[1]) {
                if( ((*linked_ptr)->car->scanz[0]) -
                   ((*linked_ptr)->car->scanx[0] - traverse_ptr->car->scanx[0])
                   * (((*linked_ptr)->car->scanz[0] - (*linked_ptr)->car->scanz[1]) /
                     ((*linked_ptr)->car->scanx[0] - (*linked_ptr)->car->scanx[1]))
                   < traverse_ptr->car->scanz[0] && !ready_to_insert) {
                 tmp_linked_ptr = linked_ptr;
                 ready_to_insert = 1;
                }
          } else {
                if( ((*linked_ptr)->car->scanz[0]) -
                   ((*linked_ptr)->car->scanx[0] - traverse_ptr->car->scanx[1])
                   * (((*linked_ptr)->car->scanz[0] - (*linked_ptr)->car->scanz[1]) /
                     ((*linked_ptr)->car->scanx[0] - (*linked_ptr)->car->scanx[1]))
                   < traverse_ptr->car->scanz[1] && !ready_to_insert) {
                 tmp_linked_ptr = linked_ptr;
                 ready_to_insert = 1;
                }
          }
         }
        }
        linked_ptr = &(*linked_ptr)->cdr;
       }

       traverse_ptr = traverse_ptr->cdr;
   }

   xsorted_polys[ scanx] = NULL;

   if( zsorted_polys == NULL) {
       *scanning_image_ptr = 0;
       scanning_image_ptr++;
   } else {
       *scanning_image_ptr =
        zsorted_polys->car->color;
       scanning_image_ptr++;
   }
  }

  ellx = 0;
  ellz = 0;
  zsorted_polys = NULL;

 }
}
```

Appendix 1.B - X Code

```
/****
  xrend.c
  Version .03
  Copyright (C) 1997 Colin Bulthaup
****/

#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

```
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <X11/extensions/XShm.h>

#include <stdio.h>
#include <stdlib.h>

#include "rend.h"

#include "icon_bitmap"

#define BITMAPDEPTH 1
#define TOO_SMALL 0
#define BIG_ENOUGH 1
#define NUM_POLYS 100
#define NUM_LIGHTS 2

#define COLRGB(r,g,b) (((r)<<11)+((g)<<5)+((b)))

Display *display;
int screen_num;
XImage *Screen1;

static char *progname;

float Nper[4][4];
float Mvv3dv[4][4];
float PerMat[4][4];
float TransMat[4][4];
float FinalMat[4][4];

#include "rend.006.c"

void main( int argc, char **argv)
{
  Window win;
  unsigned int width, height;
  int x, y;
  unsigned int border_width = 4;
  unsigned int display_width, display_height;
  unsigned int icon_width, icon_height;
  char *window_name = "Foobar";
  char *icon_name = "Tiny Foobar";
  Pixmap icon_pixmap;
  XSizeHints *size_hints;
  XIconSize *size_list;
  XWMHints *wm_hints;
  XClassHint *class_hints;
  XTextProperty windowName, iconName;
  int count;
  XEvent report;
  GC gc, clear_gc;
  XFontStruct *font_info;
  char *display_name = NULL;
  int window_size = 0;
  XShmSegmentInfo ScreenShmInfo;

  float p1[4] = { 1, 0, 0, 1};
  float clock_step = 0;
  float tmp;
  int i, j, k;
  real_polygon bunch_o_polygons[NUM_POLYS];
  point_light light_sources[ NUM_LIGHTS] = { { { 1, 1, 2}, 1},
                                             { { -1, 0, 0}, .5} };

  for( i = 0; i < NUM_LIGHTS; i++)
    normalize_3( light_sources[i].vect);

  for( i = 0; i < NUM_POLYS; i++) {
    for( j = 0; j < 3; j++) {
```

```
    for( k = 0; k < 3; k++) {
            bunch_o_polygons[i].points[j][k] = (float)(random() * -1) /
              (float)RAND_MAX;
    }
    bunch_o_polygons[i].points[j][3] = 1;
  }
}

progname = argv[0];

setup_PerMat();

if( !(size_hints = XAllocSizeHints())) {
  fprintf( stderr, "%s: failure allocating memory\n", progname);
  exit(0);
}

if( !(wm_hints = XAllocWMHints())) {
  fprintf( stderr, "%s: failure allocating memory\n", progname);
  exit(0);
}

if( !(class_hints = XAllocClassHint())) {
  fprintf( stderr, "%s: failure allocating memory\n", progname);
  exit(0);
}

if( ( display=XOpenDisplay( display_name)) == NULL) {
  fprintf( stderr, "%s: cannot connect to X server %s\n",
            progname, XDisplayName( display_name));
  exit( -1);
}

screen_num = DefaultScreen( display);
display_width = DisplayWidth( display, screen_num);
display_height = DisplayHeight( display, screen_num);

x = y = 0;

width = DEFAULT_WIDTH;
height = DEFAULT_HEIGHT;

win = XCreateSimpleWindow( display, RootWindow( display, screen_num),
                           x, y, width, height, border_width,
                           BlackPixel( display, screen_num),
                           WhitePixel( display, screen_num));

icon_pixmap = XCreateBitmapFromData( display, win, icon_bitmap_bits,
                                     icon_bitmap_width,
                                     icon_bitmap_height);

size_hints->flags = PPosition | PSize | PMinSize;
size_hints->min_width = 300;
size_hints->min_height = 200;

if( XStringListToTextProperty( &window_name, 1, &windowName) == 0) {
  fprintf( stderr, "%s: structure allocation for windowName failed.\n",
            progname);
  exit( -1);
}

if( XStringListToTextProperty( &icon_name, 1, &iconName) == 0) {
  fprintf( stderr, "%s: structure allocation for iconName failed.\n",
            progname);
  exit( -1);
}

wm_hints->initial_state = NormalState;
wm_hints->input = True;
wm_hints->icon_pixmap = icon_pixmap;

wm_hints->flags = StateHint | IconPixmapHint | InputHint;
```

```
class_hints->res_name = progname;
class_hints->res_class = "Foobar";

XSetWMProperties( display, win, &windowName, &iconName,
                  argv, argc, size_hints, wm_hints,
                  class_hints);

XSelectInput( display, win, ExposureMask | KeyPressMask |
              ButtonPressMask | StructureNotifyMask);

load_font( &font_info);

getGC( win, &gc, font_info);
get_clear_GC( win, &clear_gc);

Screen1 = XShmCreateImage( display, DefaultVisual( display, screen_num),
                           16, ZPixmap, NULL, &ScreenShmInfo,
                           DEFAULT_WIDTH, DEFAULT_HEIGHT);
ScreenShmInfo.shmid = shmget( IPC_PRIVATE,
                              2*Screen1->bytes_per_line*Screen1->height,
                              IPC_CREAT|077);

printf( "height = %d, width = %d\n", Screen1->height, Screen1->bytes_per_line);

if( ScreenShmInfo.shmid == -1) {
 perror( "shmget()");
 exit( 1);
}

ScreenShmInfo.shmaddr = Screen1->data = shmat( ScreenShmInfo.shmid,0,0);
if( ScreenShmInfo.shmaddr == -1) {
 perror( "shmat()");
 exit(1);
}

shmctl( ScreenShmInfo.shmid, IPC_RMID, NULL);
ScreenShmInfo.readOnly = False;

XShmAttach( display, &ScreenShmInfo);

XMapWindow( display, win);

calculate_normals( bunch_o_polygons, NUM_POLYS);
calculate_lighting( bunch_o_polygons, NUM_POLYS,
                    light_sources, NUM_LIGHTS);
render_image( bunch_o_polygons, NUM_POLYS, width, height,
              (unsigned short *)Screen1->data);

while( 1) {
 XNextEvent( display, &report);
 switch (report.type) {
 case Expose:
  if( report.xexpose.count != 0)
      break;

  XShmPutImage( display, win, gc, Screen1, 0, 0, 0, 0, width, height,
                False);
  XFlush( display);
  break;
 case ConfigureNotify:
  width = report.xconfigure.width;
  height = report.xconfigure.height;
  generate_M( width, 0, height, 0, 22, 0, Mvv3dv);
  mat_mat_mul_4( Mvv3dv, Nper, PerMat);
  break;
 case ButtonPress:
  for( clock_step = 0; clock_step < 31.4; clock_step += .1) {
      update_transform( clock_step);
      calculate_normals( bunch_o_polygons, NUM_POLYS);
      calculate_lighting( bunch_o_polygons, NUM_POLYS,
                          light_sources, NUM_LIGHTS);
```

```
                render_image( bunch_o_polygons, NUM_POLYS, width, height,
                              (unsigned short *)Screen1->data);
                XShmPutImage( display, win, gc, Screen1, 0, 0, 0, 0, width, height,
                              False);
                XFlush( display);
      }
      break;
    case KeyPress:
      XShmDetach( display, &ScreenShmInfo);
      shmctl( ScreenShmInfo.shmid, IPC_RMID, NULL);
      shmdt( ScreenShmInfo.shmaddr);
      XUnloadFont( display, font_info->fid);
      XFreeGC( display, gc);
      XCloseDisplay( display);
      exit( 1);
    default:
      break;
    }
  }
}

getGC( Window win, GC *gc, XFontStruct *font_info)
{
  unsigned long valuemask = 0;
  XGCValues values;
  unsigned int line_width = 6;
  int line_style = LineOnOffDash;
  int cap_style = CapRound;
  int join_style = JoinRound;
  int dash_offset = 0;
  static char dash_list[] = {12, 24};
  int list_length = 2;

  *gc = XCreateGC( display, win, valuemask, &values);

  XSetFont( display, *gc, font_info->fid);

  XSetForeground( display, *gc, BlackPixel( display, screen_num));
}

load_font( XFontStruct **font_info)
{
  char *fontname = "9x15";

  if( ( *font_info = XLoadQueryFont( display, fontname)) == NULL) {
    fprintf( stderr, "%s: Cannot open %s font\n", progname, fontname);
    exit( -1);
  }
}
```