Report of Project 2

1. Algorithms Summary

This project is to model the imaging process of a camera/eye and simulate the process. Here we concern about the geometric aspects of the imaging process and there are two commonly used projections: orthographic and perspective projections. To get the final images, transform matrix need to establish and apply them to objects's original coordinates.

Composed projection transformation matrix: It is the composed by several projection transform matrix to get the screen view of objects. We have

$$M = M_{VP} M_{Ortho} P\, M_{cam}$$

$$M_{cam} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

$$M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = M_{vp} M_{Ortho} M_{cam}$$

$$M_{cam} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix for translation, scaling, and rotation: We have

Translation:

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation:

$$\text{rotate-z}(\phi_z) = \begin{bmatrix} \cos\phi_z & -\sin\phi_z & 0 \\ \sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{rotate-y}(\phi_y) = \begin{bmatrix} \cos\phi_y & 0 & \sin\phi_y \\ 0 & 1 & 0 \\ -\sin\phi_y & 0 & \cos\phi_y \end{bmatrix} \qquad \text{rotate-x}(\phi_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & -\sin\phi_x \\ 0 & \sin\phi_x & \cos\phi_x \end{bmatrix}$$

The 3D overview rotation is given by:

$$\text{rotate-z}(\phi_z) \times \text{rotate-y}(\phi_y) \times \text{rotate-x}(\phi_x)$$

$$\text{shear-x}(d_y, d_z) = \begin{bmatrix} 1 & d_y & d_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{scale}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Finally: I also need to use file system to read the OFF file. Meanwhile, the algorithm to draw line, triangle is given by openGL.

2. The implementation

Draw floor and axis:

```
//draw floor function
void drawFloor(double xmin, double xmax, double ymin, double ymax, double nX, double nY, double floorEdge, double matrixFinal[][4],Line floor[])
{
	int i,j;
	for(i = 0; i < nX; i++){
		floor[i].sPoint[0] = xmin;
		floor[i].sPoint[1] = ymin + i * floorEdge;
		floor[i].sPoint[2] = 0;
		floor[i].sPoint[3] = 1;

		floor[i].ePoint[0] = xmax;
		floor[i].ePoint[1] = floor[i].sPoint[1];
		floor[i].ePoint[2] = 0;
		floor[i].ePoint[3] = 1;
	}

	for(i = i,j = 0; i < nX + nY; i++,j++){
		floor[i].sPoint[0] = xmin + j * floorEdge;
		floor[i].sPoint[1] = ymin;
		floor[i].sPoint[2] = 0;
		floor[i].sPoint[3] = 1;

		floor[i].ePoint[0] = floor[i].sPoint[0];
		floor[i].ePoint[1] = ymax;
		floor[i].ePoint[2] = 0;
		floor[i].ePoint[3] = 1;
	}
	for(i = 0; i < nX + nY; i++){
		matrixApply(matrixFinal,floor[i].sPoint);
		matrixApply(matrixFinal,floor[i].ePoint);
			glVertex2d(floor[i].sPoint[0]/floor[i].sPoint[3],floor[i].sPoint[1]/floor[i].sPoint[3]);
		glVertex2d(floor[i].ePoint[0]/floor[i].ePoint[3],floor[i].ePoint[1]/floor[i].ePoint[3]);
	}
}
```

In function Render_SSD()

```
/* Draw floor */
	double xmin,xmax,ymin,ymax,floorEdge;
	int nX,nY;

	xmin = ascene->floor.xmin;
	xmax = ascene->floor.xmax;
	ymin = ascene->floor.ymin;
	ymax = ascene->floor.ymax;
	floorEdge = ascene->floor.size;
```

```
        nY = ((xmax-xmin)/floorEdge) + 1;
        nX = ((ymax-ymin)/floorEdge) + 1;
        Line floor[nX + nY];

        glLineWidth(2);
        glBegin(GL_LINES);
        glColor3f(ascene->floor.color.rgba[0],ascene->floor.color.rgba[1],ascene->floor.color.rgba[2]);
        drawFloor(xmin,xmax,ymin,ymax,nX,nY,floorEdge,matrixFinal,floor);
        glEnd();

        /* draw axis*/
        glLineWidth(ascene->axis.width);
        glBegin(GL_LINES);
        if(ascene->isAxis == 1){
                double origin[4] = {0,0,0,1};
                double axisX[4] = {ascene->axis.length,0,0,1};
                double axisY[4] = {0,ascene->axis.length,0,1};
                double axisZ[4] = {0,0,ascene->axis.length,1};

                matrixApply(matrixFinal,origin);
                matrixApply(matrixFinal,axisX);
                matrixApply(matrixFinal,axisY);
                matrixApply(matrixFinal,axisZ);

                glColor3f(1,0,0);
                glVertex2d(origin[0]/origin[3],origin[1]/origin[3]);
                glVertex2d(axisX[0]/axisX[3],axisX[1]/axisX[3]);
                glColor3f(0,1,0);
                glVertex2d(origin[0]/origin[3],origin[1]/origin[3]);
                glVertex2d(axisY[0]/axisY[3],axisY[1]/axisY[3]);
                glColor3f(0,0,1);
                glVertex2d(origin[0]/origin[3],origin[1]/origin[3]);
                glVertex2d(axisZ[0]/axisZ[3],axisZ[1]/axisZ[3]);
        }
        glEnd();
```

Matrix and vector calculation

```
void vecCross(double firstVec[], double secondVec[], double result[])
{
        result[0] = firstVec[1] * secondVec[2] - firstVec[2] * secondVec[1];
        result[1] = firstVec[2] * secondVec[0] - firstVec[0] * secondVec[2];
        result[2] = firstVec[0] * secondVec[1] - firstVec[1] * secondVec[0];
}

void vecUnitization(double vec[],double result[])
{
        double vecSqrt = sqrt(vec[0]*vec[0]+vec[1]*vec[1]+vec[2]*vec[2]);
        result[0] = vec[0]/vecSqrt;
        result[1] = vec[1]/vecSqrt;
        result[2] = vec[2]/vecSqrt;
}

void matrixMultiply(double firstMatrix[][4], double secondMatrix[][4])
{
        int i,j;
        double result[4][4];
```

```
	for(i = 0; i < 4; i++){
		for(j = 0; j < 4; j++){
			result[i][j] = firstMatrix[i][0] * secondMatrix[0][j] + firstMatrix[i][1] * secondMatrix[1][j] + firstMatrix[i][2] *
secondMatrix[2][j] + firstMatrix[i][3] * secondMatrix[3][j];
		}
	}
	for(i = 0; i < 4; i++){
		for(j = 0; j < 4; j++){
			secondMatrix[i][j] = result[i][j];
		}
	}
}

void matrixApply(double matrix[][4], double coordinates[])
{
	int i;
	double tmp[4];
	for(i = 0; i < 4; i++){
		tmp[i] = matrix[i][0] * coordinates[0] + matrix[i][1] * coordinates[1] + matrix[i][2] * coordinates[2] + matrix[i][3] *
coordinates[3];
	}
	for(i = 0; i < 4; i++){
		coordinates[i] = tmp[i];
	}
}

void matrixInitial(double matrix[][4])
{
	int i,j;
	for(i = 0; i < 4; i++){
		for(j = 0; j < 4; j++){
			if(i == j){matrix[i][j] = 1;}
			else{matrix[i][j] = 0;}
		}
	}
}
```

M_{cam}(in Render_SSD())

```
/* Camera View */
	int ii;
	matrixInitial(intermediaM);
	matrixInitial(mCam);
	double u[3],v[3],w[3];
	double gaze[3] = {ascene->gaze.xyz[0],ascene->gaze.xyz[1],ascene->gaze.xyz[2]};
	double upVector[3] = {ascene->upVector.xyz[0],ascene->upVector.xyz[1],ascene->upVector.xyz[2]};

	vecUnitization(gaze,w);

	for(ii = 0; ii < 3; ii++){
		w[ii] = -w[ii];
	}
	vecCross(upVector,w,u);
	vecUnitization(u,u);
	vecCross(w,u,v);
	ii = 0;
	for(ii = 0; ii < 3; ii++){
		intermediaM[0][ii] = u[ii];
```

```
            intermediaM[1][ii] = v[ii];
            intermediaM[2][ii] = w[ii];
            mCam[ii][3] = -ascene->eye.xyz[ii];
        }
        matrixMultiply(intermediaM,mCam);
```

Perspective and Orthographic projection matrix

```
void getFinalTransformMatrix(double anglePers, double nearPers, double farPers, double rightOrtho, double topOrtho, double farOrtho, double nearOrtho, double pjType, double screenWidth, double screenHeight, double mCam[][4], double matrixFinal[][4])
{
        /*Persective(1) and Orthographic(0) Projection matrix*/
        /*The part to get perspective projection matrix*/
        double mPersp[4][4],mOrtho[4][4],mVp[4][4];
        matrixInitial(mPersp);
        matrixInitial(mOrtho);
        matrixInitial(mVp);
        if(pjType == 1){
            double Pi = 3.141592653;
            double radian = (anglePers/(double)180) * Pi;
            double top = tan(radian/(double)2) * (-nearPers);
            double right = (double)screenWidth/(double)screenHeight * top;
            mPersp[0][0] = nearPers/right;
            mPersp[1][1] = nearPers/top;
            mPersp[2][2] = (farPers + nearPers)/(nearPers - farPers);
            mPersp[2][3] = (2 * nearPers * farPers)/(farPers- nearPers);
            mPersp[3][2] = 1;
            mPersp[3][3] = 0;

        }
        /* The part to get orthographic projection atrix*/
        else{
            mOrtho[0][0] = (double)1/rightOrtho; //r=-l
            mOrtho[1][1] = (double)1/topOrtho;
            mOrtho[2][2] = (double)2/(farOrtho - nearOrtho);
            mOrtho[2][3] = -(farOrtho + nearOrtho)/(farOrtho - nearOrtho);
        }

        /*View point matrix*/
        mVp[0][0] = screenWidth/(double)2;
        mVp[0][3] = (screenWidth - 1)/(double)2;
        mVp[1][1] = screenHeight/(double)2;
        mVp[1][3] = (screenHeight - 1)/(double)2;
        /*Final transform matrix*/
        matrixMultiply(mCam,matrixFinal);
        matrixMultiply(mPersp,matrixFinal);
        matrixMultiply(mOrtho,matrixFinal);
        matrixMultiply(mVp,matrixFinal);
}
```

In Render_SSD()(call function above)

```
        double anglePers, nearPers, farPers, rightOrtho, topOrtho, farOrtho, nearOrtho,pjType;
        double screenWidth,screenHeight;
        screenWidth = ascene->screen_w;
        screenHeight = ascene->screen_h;
        anglePers = ascene->persp.angle;
        nearPers = ascene->persp.near;
        farPers = ascene->persp.far;
        rightOrtho = ascene->ortho.right;
```

```
	topOrtho = ascene->ortho.top;
	farOrtho = ascene->ortho.far;
	nearOrtho = ascene->ortho.near;
	pjType = ascene->pjType;

	getFinalTransformMatrix(anglePers, nearPers, farPers, rightOrtho, topOrtho, farOrtho, nearOrtho, pjType, screenWidth, screenHeight, mCam, matrixFinal);
```

Scale,Translate,Shear,Rotation and composition:

```
//functions that will be called in Render_SSD
void rotateMatrix(double axis[], double mRotate[][4], double mTransform[][4], double radian)
{
	int i,j;
	double u[3],v[3],t[3];
	double tmp[4][4];
	vecUnitization(axis,axis);

	t[0] = axis[0];
	t[1] = axis[1]+1;
	t[2] = axis[2];
	vecCross(t,axis,u);
	vecUnitization(u,u);
	vecCross(axis,u,v);
	vecUnitization(v,v);

	matrixInitial(tmp);
	matrixInitial(mRotate);
	tmp[0][0] = cos(radian);
	tmp[0][1] = -sin(radian);
	tmp[1][0] = sin(radian);
	tmp[1][1] = cos(radian);
	for(i = 0; i < 3; i++){
		mRotate[0][i] = u[i];
		mRotate[1][i] = v[i];
		mRotate[2][i] = axis[i];

	}
	matrixMultiply(tmp,mRotate);

	matrixInitial(tmp);
	for(i = 0; i < 3; i++){
		tmp[i][0] = u[i];
		tmp[i][1] = v[i];
		tmp[i][2] = axis[i];

	}
	matrixMultiply(tmp,mRotate);
	matrixMultiply(mRotate,mTransform);
}

void objectsDraw(POINT points[], int nvertices, char arg0[], char arg1[], char arg2[], char arg3[], char arg4[], char arg5[])
{
	int i;
	for(i=0;i<nvertices;i++){
		switch(i){
			case 0:
	glColor3f(points[atoi(arg1)].rgba[0],points[atoi(arg1)].rgba[1],points[atoi(arg1)].rgba[2]);
```

```
        glVertex2d(points[atoi(arg1)].xyz[0]/points[atoi(arg1)].xyz[3],points[atoi(arg1)].xyz[1]/points[atoi(arg1)].xyz[3]);break;
                case 1:
        glColor3f(points[atoi(arg2)].rgba[0],points[atoi(arg2)].rgba[1],points[atoi(arg2)].rgba[2]);
        glVertex2d(points[atoi(arg2)].xyz[0]/points[atoi(arg2)].xyz[3],points[atoi(arg2)].xyz[1]/points[atoi(arg2)].xyz[3]);break;
                case 2:
        glColor3f(points[atoi(arg3)].rgba[0],points[atoi(arg3)].rgba[1],points[atoi(arg3)].rgba[2]);
        glVertex2d(points[atoi(arg3)].xyz[0]/points[atoi(arg3)].xyz[3],points[atoi(arg3)].xyz[1]/points[atoi(arg3)].xyz[3]);break;
                case 3:
        glColor3f(points[atoi(arg4)].rgba[0],points[atoi(arg4)].rgba[1],points[atoi(arg4)].rgba[2]);
        glVertex2d(points[atoi(arg4)].xyz[0]/points[atoi(arg4)].xyz[3],points[atoi(arg4)].xyz[1]/points[atoi(arg4)].xyz[3]);break;
                case 4:
        glColor3f(points[atoi(arg5)].rgba[0],points[atoi(arg5)].rgba[1],points[atoi(arg5)].rgba[2]);
        glVertex2d(points[atoi(arg5)].xyz[0]/points[atoi(arg5)].xyz[3],points[atoi(arg5)].xyz[1]/points[atoi(arg5)].xyz[1]);break;
                }
        }
}
```

//in Render_SSD()

```
int nT = 0;
        int nR = 0;
        int nS = 0;
        int nM = 0;
        for(i = 0; i < ascene->nidentities; i++){
                matrixInitial(mTransform);
                for(j = 0; j < ascene->identities[i].inStr_num; j++){
                        if(ascene->identities[i].instr[j] == TRANSLATE_KEY){
                                matrixInitial(mTranslate);
                                mTranslate[0][3] = ascene->translate[nT].xyz[0];
                                mTranslate[1][3] = ascene->translate[nT].xyz[1];
                                mTranslate[2][3] = ascene->translate[nT].xyz[2];
                                matrixMultiply(mTranslate,mTransform);
                                nT++;
                        }
                        else if(ascene->identities[i].instr[j] == ROTATE_KEY){
                                double axis[3];
                                axis[0] = ascene->rotate[nR].xyz[0];
                                axis[1] = ascene->rotate[nR].xyz[1];
                                axis[2] = ascene->rotate[nR].xyz[2];
                                double Pi = 3.141592653;
                                double radian = (ascene->rotate[nR].angle/(double)180) * Pi;
                                rotateMatrix(axis,mRotate,mTransform,radian);
                                nR++;

                        }
                        else if(ascene->identities[i].instr[j] == SCALE_KEY){
                                matrixInitial(mScale);
                                mScale[0][0] = ascene->scale[nS].xyz[0];
                                mScale[1][1] = ascene->scale[nS].xyz[1];
                                mScale[2][2] = ascene->scale[nS].xyz[2];
                                matrixMultiply(mScale,mTransform);
                                nS++;
                        }
                        else if(ascene->identities[i].instr[j] == MESH_KEY){
                                char arg0[MAXLINELENGTH],arg1[MAXLINELENGTH],arg2[MAXLINELENGTH],
                                        arg3[MAXLINELENGTH],arg4[MAXLINELENGTH],arg5[MAXLINELENGTH];
                                char line[MAXLINELENGTH];
                                char *offname = ascene->mesh[nM].offname;
```

```
                double tM[4][4];
                matrixInitial(tM);
                matrixMultiply(mTransform,tM);
                matrixMultiply(matrixFinal,tM);

                arg5[0] = 'N';
                FILE *fp = fopen(offname,"rb");
                /*OFF*/
                if(fgets(line,MAXLINELENGTH,fp) == NULL)
                        exit(-1);

                /*total*/
                if(fgets(line,MAXLINELENGTH,fp) == NULL)
                        exit(-1);

                sscanf(line,"%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]",arg0,arg1,arg2);

                int npoints = atoi(arg0);
                int npolygons    = atoi(arg1);
                int k;
                POINT points[npoints];

                for(k = 0; k < npoints; k++){
                        if(fgets(line,MAXLINELENGTH,fp) == NULL)
                                exit(-1);

                        if(line[0] == 32)

sscanf(line,"%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]",arg0,arg1,arg2,arg3,arg4,arg5);
                        else

sscanf(line,"%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]",arg0,arg1,arg2,arg3,arg4,arg5);

                        points[k].xyz[0] = atof(arg0);
                        points[k].xyz[1] = atof(arg1);
                        points[k].xyz[2] = atof(arg2);
                        points[k].xyz[3] = 1;

                        if(arg5[0] == 'N'){
                                points[k].rgba[0] = ascene->mesh[nM].color.rgba[0];
                                points[k].rgba[1] = ascene->mesh[nM].color.rgba[1];
                                points[k].rgba[2] = ascene->mesh[nM].color.rgba[2];
                        }
                        else{
                                points[k].rgba[0] = atof(arg3);
                                points[k].rgba[1] = atof(arg4);
                                points[k].rgba[2] = atof(arg5);
                        }
                        matrixApply(tM,points[k].xyz);
                }
                for(k = 0; k < npolygons; k++){
                        if(fgets(line,MAXLINELENGTH,fp) == NULL)
                                exit(-1);

                        if(line[0] == 32)

sscanf(line,"%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]",arg0,arg1,arg2,arg3,arg4,arg5);
                        else
```

```
        sscanf(line,"%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]%*[ ]%[^ ]",arg0,arg1,arg2,arg3,arg4,arg5);

                                        int nvertices = atoi(arg0);
                                        glLineWidth(ascene->mesh[nM].width);
                                        glBegin(GL_LINE_LOOP);
                                        objectsDraw(points, nvertices, arg0, arg1, arg2, arg3, arg4, arg5);
                                        glEnd();
                                }
                                fclose(fp);
                        nM++;
                }
        }
}
```

SSD file read:(mostly in red)

SSD_util.c

```
#include <stdio.h>
#define SSD_UTIL_SOURCE_CODE
#include "SSD_util.h"
#define MAXLINELENGTH 1000
#define MAXLABELLEN          16

struct ssd_keyword keyword_table[] =
  {{SCREEN_KEY, "screen", 5},
   {COLOR_KEY, "color", 3},
   {LINE_KEY, "line", 1},
   {VERTEX_KEY, "vertex", 3},
   {POLYLINE_KEY, "polyline", 2},
   {CIRCLE_KEY, "circle", 3},
   {ARC_KEY, "arc", 5},
   {SAVE_KEY, "save", 1},
   {TRIANGLE_KEY, "triangle",0},
   {EYE_KEY, "eye",3},
   {GAZE_KEY, "gaze",3},
   {UPVECTOR_KEY, "upvector",3},
   {ORTHO_KEY, "ortho",4},
   {PERSP_KEY, "perspective",3},
   {FLOOR_KEY, "floor",5},
   {AXIS_KEY, "axis",2},
   {IDENTITY_KEY, "identity",0},
   {TRANSLATE_KEY, "translate",3},
   {ROTATE_KEY, "rotate",3},
   {SCALE_KEY, "scale",3},
   {MESH_KEY, "mesh",2},
   {-1,"unknown", 0}};

int match_Keyword(char *keyword, int *npara)
{
  int i;
  *npara = 0;
  for (i=0; keyword_table[i].key_id != -1; i++) {
    if (strcmp(keyword_table[i].name, keyword) == 0) {
      *npara = keyword_table[i].npara;
      return keyword_table[i].key_id;
    }
  }
```

```
    return -1;
}

int readAndParse(FILE *inFilePtr, char *keyword, char *arg0,
        char *arg1, char *arg2, char *arg3, char *arg4)
{
    static char line[MAXLINELENGTH];
    char *ptr = line;

    /* delete prior values */
    keyword[0] = arg0[0] = arg1[0] = arg2[0] = arg3[0] = arg4[0] = '\0';
    if (feof(inFilePtr)) return(0);
    /* read the line from the SSD file */
    while (1) {
        if (fgets(line, MAXLINELENGTH, inFilePtr) == NULL) {
            /* reached end of file */
            return(0);
        }

        if (feof(inFilePtr)==0) {
            /* check for line too long (by looking for a \n) */
            if (strchr(line, '\n') == NULL) {
            /* line too long */
            printf("error: line too long\n");
            exit(1);
            }
        }
        if (line[0] != '#') break;
    }
    /*
     * Parse the line.
     */
    sscanf(line,
    "%[^\t\n ]%*[\t\n ]%[^\t\n ]%*[\t\n ]%[^\t\n ]%*[\t\n ]%[^\t\n ]%*[\t\n ]%[^\t\n ]%*[\t\n ]%[^\t\n ]%*[\t\n ]",
            keyword, arg0, arg1, arg2, arg3, arg4);
    return strlen(line);
}

int Read_SSD_Scene(char *fname, SCENE *ascene, char *saved_fname)
{
    char keyword[MAXLINELENGTH],    arg0[MAXLINELENGTH],
        arg1[MAXLINELENGTH], arg2[MAXLINELENGTH], arg3[MAXLINELENGTH];
    char arg4[MAXLINELENGTH];
            int ident = 0;
    FILE *fp;

    /* We first set all the default values */
    RGB_COLOR fcolor, vcolor;
    int ind, ii, num_ver, key_id, key_id1, npara;

    ascene->screen_w = 600;
    ascene->screen_h = 400;
    /* The default color is white */
    ascene->bcolor.rgba[0] = 1.0;
    ascene->bcolor.rgba[1] = 1.0;
    ascene->bcolor.rgba[2] = 1.0;
    ascene->bcolor.rgba[3] = 1.0;
    fcolor.rgba[0] = 0.0; fcolor.rgba[1] = 0.0;
```

```
fcolor.rgba[2] = 0.0; fcolor.rgba[3] = 1.0;
ascene->nlines = 0;
ascene->npolylines = 0;
ascene->ntriangles = 0;
//
ascene->nidentities = 0;
ascene->pjType = 0; //0:orthographic 1:perspective
ascene->isAxis = 0; //0 represents noAxis
int ntranslate = 0;
int nrotate = 0;
int nscale = 0;
int nmesh = 0;
fp = fopen(fname,"rb");
if (fp == NULL) {
   fprintf(stderr,"%s:%d: Can not open SSD file <%s>.\n",
                __FILE__, __LINE__, fname);
   return -1;
}
while (readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4) > 0) {
   if (keyword[0] == '\0') {
      /* We simply all blank lines */
      continue;
   }
   key_id = match_Keyword(keyword, &npara);
   switch(key_id) {
   case LINE_KEY:
      ascene->nlines++;
      break;
   case POLYLINE_KEY:
      ascene->npolylines++;
      break;
   case TRIANGLE_KEY:
      ascene->ntriangles++;
      break;

   case IDENTITY_KEY:
      ascene->nidentities++;
      break;
   case TRANSLATE_KEY:
      ntranslate++;
      break;
   case SCALE_KEY:
      nscale++;
      break;
   case ROTATE_KEY:
      nrotate++;
      break;
   case MESH_KEY:
      nmesh++;
      break;
   }
}
printf("There are %d lines, %d polylines, and %d triangles in %s.\n",
       ascene->nlines, ascene->npolylines, ascene->ntriangles,
       fname);
/* We rewind the file to the very beginning to read the file again */
rewind(fp);
ascene->lines = (LINE *)malloc(sizeof(LINE) * ascene->nlines);
```

```
ascene->polylines = (POLYLINE *)malloc(sizeof(POLYLINE) *
                                        ascene->npolylines);
ascene->triangles = (TRIANGLE *)malloc(sizeof(TRIANGLE) *
                                        ascene->ntriangles);
ascene->identities = (IDENTITY *)malloc(sizeof(IDENTITY) * ascene->nidentities);
ascene->translate = (TRANSLATE *)malloc(sizeof(TRANSLATE) * ntranslate);
ascene->scale = (SCALE *)malloc(sizeof(SCALE) * nscale);
ascene->rotate = (ROTATE *)malloc(sizeof(ROTATE) * nrotate);
ascene->mesh = (MESH *)malloc(sizeof(MESH) * nmesh);

ascene->nlines = 0;
ascene->npolylines = 0;
ascene->ntriangles = 0;
ascene->nidentities = 0;
ntranslate = 0;
nscale = 0;
nrotate = 0;
nmesh = 0;
int instr_ind = 0;
int brk = 0;
int readAndParseResult = 0;

while (readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4) > 0) {
   if (keyword[0] == '\0') {
      /* We simply all blank lines */
      continue;
   }
   key_id = match_Keyword(keyword, &npara);
   switch(key_id) {
   case SCREEN_KEY:
     ascene->screen_w = atoi(arg0);
     ascene->screen_h = atoi(arg1);
     ascene->bcolor.rgba[0] = atoi(arg2)/255.0;
     ascene->bcolor.rgba[1]    = atoi(arg3)/255.0;
     ascene->bcolor.rgba[2]    = atoi(arg4)/255.0;
     ascene->bcolor.rgba[3] = 1.0;
     break;

   case COLOR_KEY:
      /* We read the color */
      fcolor.rgba[0] = atoi(arg0)/255.0;    fcolor.rgba[1] = atoi(arg1)/255.0;
      fcolor.rgba[2] = atoi(arg2)/255.0;
      break;
   case LINE_KEY:
      ind = ascene->nlines;
      ascene->lines[ind].width = atof(arg0);
      /* We set the default colors */
      vcolor = fcolor;
      num_ver = 0;
      while (readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4) > 0) {
         key_id1 = match_Keyword(keyword, &npara);
         switch(key_id1) {
         case VERTEX_KEY:
            ascene->lines[ind].vertices[num_ver].xyzw[0] = atof(arg0);
        ascene->lines[ind].vertices[num_ver].xyzw[1] = atof(arg1);
            ascene->lines[ind].vertices[num_ver].xyzw[2] = atof(arg2);
        memcpy(ascene->lines[ind].vertices[num_ver].rgba,
               vcolor.rgba, sizeof(vcolor.rgba));
```

```
#if defined(DEBUG_FLAG)
                printf("Point %d %d with color %6.4f %6.4f %6.4f\n",
                        (int)ascene->lines[ind].vertices[num_ver].xyzw[0],
                    (int)ascene->lines[ind].vertices[num_ver].xyzw[1],
                    ascene->lines[ind].vertices[num_ver].rgba[0],
                    ascene->lines[ind].vertices[num_ver].rgba[1],
                    ascene->lines[ind].vertices[num_ver].rgba[2]);
#endif
                num_ver ++;
                break;
              case COLOR_KEY:
              vcolor.rgba[0] = atoi(arg0)/255.0; vcolor.rgba[1] = atoi(arg1)/255.0;
                vcolor.rgba[2] = atoi(arg2)/255.0;
                break;
              default:
                printf("%s:%d Line (%s %s %s %s %s %s) ignored.\n",
                        __FILE__, __LINE__, keyword, arg0, arg1, arg2,
                        arg3, arg4);
              }
              if (num_ver == 2) {
                break;
              }
            }
            ascene->nlines++;
            break;
          case POLYLINE_KEY:
            ind = ascene->npolylines;
            ascene->polylines[ind].nvertices = atoi(arg0);
            ascene->polylines[ind].width = atof(arg1);
            ascene->polylines[ind].vertices =
            (COLOR_VERTEX *)malloc(sizeof(COLOR_VERTEX) *
                                    ascene->polylines[ind].nvertices);
            vcolor = fcolor;
            num_ver = 0;
            while (readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4) > 0) {
              key_id1 = match_Keyword(keyword, &npara);
              switch(key_id1) {
              case VERTEX_KEY:
                ascene->polylines[ind].vertices[num_ver].xyzw[0] = atof(arg0);
              ascene->polylines[ind].vertices[num_ver].xyzw[1] = atof(arg1);
                ascene->polylines[ind].vertices[num_ver].xyzw[2] = atof(arg2);
              memcpy(ascene->polylines[ind].vertices[num_ver].rgba,
                    vcolor.rgba, sizeof(vcolor.rgba));

#if defined(DEBUG_FLAG)
                printf("Point %d %d with color %6.4f %6.4f %6.4f\n",
                        (int)ascene->polylines[ind].vertices[num_ver].xyzw[0],
                    (int)ascene->polylines[ind].vertices[num_ver].xyzw[1],
                    ascene->polylines[ind].vertices[num_ver].rgba[0],
                    ascene->polylines[ind].vertices[num_ver].rgba[1],
                    ascene->polylines[ind].vertices[num_ver].rgba[2]);
#endif
                num_ver ++;
                break;
              case COLOR_KEY:
              vcolor.rgba[0] = atoi(arg0)/255.0; vcolor.rgba[1] = atoi(arg1)/255.0;
                vcolor.rgba[2] = atoi(arg2)/255.0;
```

```
                break;
            default:
                printf("%s:%d Line (%s %s %s %s %s %s) ignored.\n",
                        __FILE__, __LINE__, keyword, arg0, arg1, arg2,
                        arg3, arg4);
            }
            if (num_ver >= ascene->polylines[ind].nvertices) {
                break;
            }
        }
        ascene->npolylines++;
        break;

    case TRIANGLE_KEY:
        ind = ascene->ntriangles;
        vcolor = fcolor;
        num_ver = 0;
        while (readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4) > 0) {
            key_id1 = match_Keyword(keyword, &npara);
            switch(key_id1) {
            case VERTEX_KEY:
                ascene->triangles[ind].vertices[num_ver].xyzw[0] = atof(arg0);
            ascene->triangles[ind].vertices[num_ver].xyzw[1] = atof(arg1);
                ascene->triangles[ind].vertices[num_ver].xyzw[2] = atof(arg2);
            memcpy(ascene->triangles[ind].vertices[num_ver].rgba,
                    vcolor.rgba, sizeof(vcolor.rgba));
#if defined(DEBUG_FLAG)
                printf("Point %d %d with color %6.4f %6.4f %6.4f\n",
                        (int)ascene->triangles[ind].vertices[num_ver].xyzw[0],
                    (int)ascene->triangles[ind].vertices[num_ver].xyzw[1],
                    ascene->triangles[ind].vertices[num_ver].rgba[0],
                    ascene->triangles[ind].vertices[num_ver].rgba[1],
                    ascene->triangles[ind].vertices[num_ver].rgba[2]);
#endif
                num_ver ++;
                break;
            case COLOR_KEY:
            vcolor.rgba[0] = atoi(arg0)/255.0; vcolor.rgba[1] = atoi(arg1)/255.0;
                vcolor.rgba[2] = atoi(arg2)/255.0;
                break;
            default:
                printf("%s:%d Line (%s %s %s %s %s %s) ignored.\n",
                        __FILE__, __LINE__, keyword, arg0, arg1, arg2,
                        arg3, arg4);
            }
            if (num_ver >= 3) {
                break;
            }
        }
        ascene->ntriangles++;
        break;


    case EYE_KEY:
        ascene->eye.xyz[0] = atof(arg0);
        ascene->eye.xyz[1] = atof(arg1);
        ascene->eye.xyz[2] = atof(arg2);
        break;
```

```
case GAZE_KEY:
    ascene->gaze.xyz[0] = atof(arg0);
    ascene->gaze.xyz[1] = atof(arg1);
    ascene->gaze.xyz[2] = atof(arg2);
    break;

case UPVECTOR_KEY:
    ascene->upVector.xyz[0] = atof(arg0);
    ascene->upVector.xyz[1] = atof(arg1);
    ascene->upVector.xyz[2] = atof(arg2);
    break;

case ORTHO_KEY:
    ascene->ortho.right = atof(arg0);
    ascene->ortho.top = atof(arg1);
    ascene->ortho.near = atof(arg2);
    ascene->ortho.far = atof(arg3);
    break;

case PERSP_KEY:
    ascene->pjType = 1;
    ascene->persp.near = atof(arg1);
    ascene->persp.far = atof(arg2);
    ascene->persp.angle = atof(arg0);
    break;

case FLOOR_KEY:
    ascene->floor.size = atof(arg0);
    ascene->floor.xmin = atof(arg1);
    ascene->floor.xmax = atof(arg2);
    ascene->floor.ymin = atof(arg3);
    ascene->floor.ymax = atof(arg4);
    ascene->floor.color = fcolor;
    break;

case AXIS_KEY:
    ascene->isAxis = 1;
    ascene->axis.width = atof(arg0);
    ascene->axis.length = atof(arg1);
    break;

case IDENTITY_KEY:
    readAndParseResult = readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4);
        while (readAndParseResult > 0) {
            key_id1 = match_Keyword(keyword, &npara);
            switch(key_id1) {
        case TRANSLATE_KEY:
            ascene->translate[ntranslate].xyz[0] = atof(arg0);
            ascene->translate[ntranslate].xyz[1] = atof(arg1);
            ascene->translate[ntranslate++].xyz[2] = atof(arg2);
            ascene->identities[ascene->nidentities].instr[instr_ind++] = TRANSLATE_KEY;
            break;
        case ROTATE_KEY:
            ascene->rotate[nrotate].angle = atof(arg0);
            ascene->rotate[nrotate].xyz[0] = atof(arg1);
            ascene->rotate[nrotate].xyz[1] = atof(arg2);
            ascene->rotate[nrotate++].xyz[2] = atof(arg3);
```

```
            ascene->identities[ascene->nidentities].instr[instr_ind++] = ROTATE_KEY;
            break;
        case SCALE_KEY:
            ascene->scale[nscale].xyz[0] = atof(arg0);
            ascene->scale[nscale].xyz[1] = atof(arg1);
            ascene->scale[nscale++].xyz[2] = atof(arg2);
            ascene->identities[ascene->nidentities].instr[instr_ind++] = SCALE_KEY;
            break;
        case MESH_KEY:
            strncpy(ascene->mesh[nmesh].offname,arg0,sizeof(arg0));
            ascene->mesh[nmesh].width = atof(arg1);
            ascene->mesh[nmesh++].color = fcolor;
            ascene->identities[ascene->nidentities].instr[instr_ind++] = MESH_KEY;
            break;
        case COLOR_KEY:
            fcolor.rgba[0] = atoi(arg0)/255.0;    fcolor.rgba[1] = atoi(arg1)/255.0;
            fcolor.rgba[2] = atoi(arg2)/255.0;
            break;
        case IDENTITY_KEY:
             case SAVE_KEY:
             case LINE_KEY:
                 brk = 1;
                 fseek(fp,-readAndParseResult,SEEK_CUR);
                 break;
             default:
                 printf("%s:%d Line (%s %s %s %s %s %s) ignored.\n",
                     __FILE__, __LINE__, keyword, arg0, arg1, arg2,
                     arg3, arg4);
                         }
          if(brk == 1)
          break;
          readAndParseResult = readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4);
                  }
      ascene->identities[ascene->nidentities].inStr_num = instr_ind;
      ascene->nidentities++;
      instr_ind = 0;
      brk = 0;
      break;

    case SAVE_KEY:
      strcpy(saved_fname, arg0);
      break;
    default:
      printf("%s:%d Keyword (%s) and the line (%s %s %s %s %s) ignored.\n",
              __FILE__, __LINE__, keyword, arg0, arg1, arg2, arg3, arg4);

    }
  }
  fclose(fp);
  return 0;
}
```

SSD_util.h(in red)

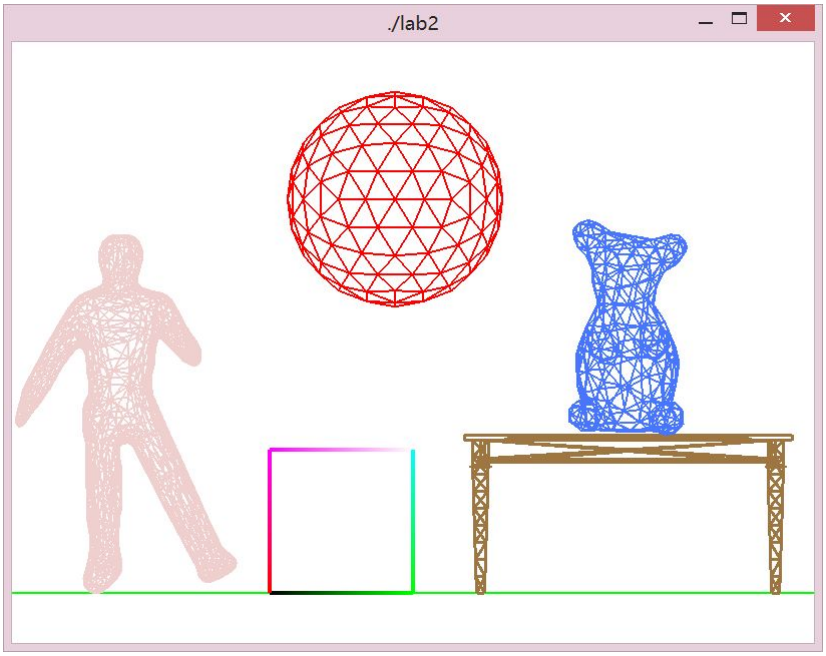
```
#if !defined(SSD_UTIL_H_H)
#define SSD_UTIL_H_H
#include <stdlib.h>
#include <string.h>
```

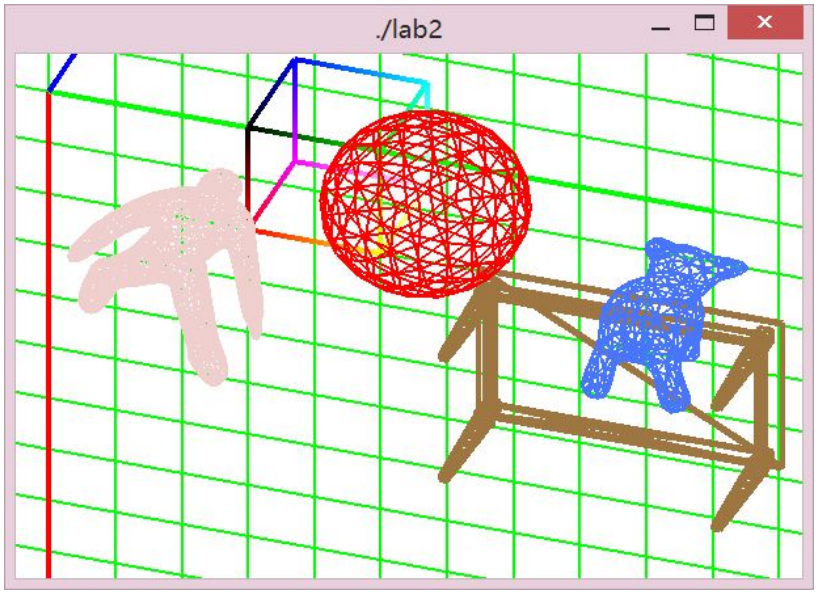
```
#define SCREEN_KEY      0
#define COLOR_KEY       1
#define LINE_KEY        2
#define VERTEX_KEY      3
#define POLYLINE_KEY 4
#define CIRCLE_KEY      5
#define ARC_KEY         6
#define SAVE_KEY        7
#define TRIANGLE_KEY 8
#define EYE_KEY         9
#define GAZE_KEY       10
#define UPVECTOR_KEY           11
#define ORTHO_KEY      12
#define PERSP_KEY       13
#define FLOOR_KEY      14
#define AXIS_KEY       15
#define IDENTITY_KEY     16
#define TRANSLATE_KEY    17
#define ROTATE_KEY         18
#define SCALE_KEY          19
#define MESH_KEY           20

struct ssd_keyword {
   /* Keyword table entry to be used for reading SSD */
   int key_id;
   char name[32];
   int npara;
};

typedef struct {
   double xyzw[4];
} VERTEX;

typedef struct {
   float rgba[4];
} RGB_COLOR;

typedef struct {
   double xyzw[4];
   float    rgba[4];
} COLOR_VERTEX;

typedef struct {
   double width;
   COLOR_VERTEX vertices[2];
} LINE;

typedef struct {
   double width;
   int        nvertices;
   COLOR_VERTEX *vertices;
} POLYLINE;

typedef struct {
   COLOR_VERTEX vertices[3];
} TRIANGLE;
```

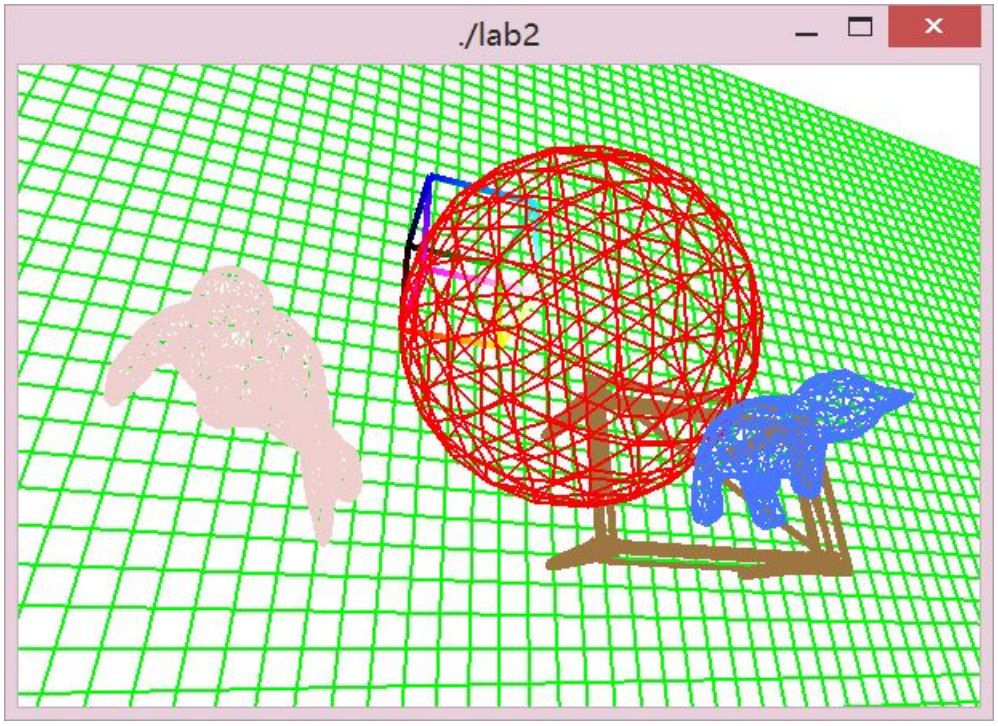
```
typedef struct {
   double xyz[3];
} Vector;

typedef struct {
   double near;
   double far;
   double angle;
} PERSP;

typedef struct {
   double right;
   double top;
   double near;
   double far;
} ORTHO;

typedef struct {
   double xmin;
   double xmax;
   double ymin;
   double ymax;
   double size;
   RGB_COLOR color;
} FLOOR;

typedef struct {
   double width;
   double length;
} AXIS;

typedef struct {
   int inStr_num;
   int instr[50];
} IDENTITY;

typedef struct {
   double xyz[3];
} TRANSLATE;

typedef struct {
   double angle;
   double xyz[3];
} ROTATE;

typedef struct {
   double xyz[3];
} SCALE;

typedef struct {
   char offname[1000];
   double width;
   RGB_COLOR color;
} MESH;

typedef struct {
   int screen_w, screen_h;
   RGB_COLOR bcolor; /* The background color for the window */
```

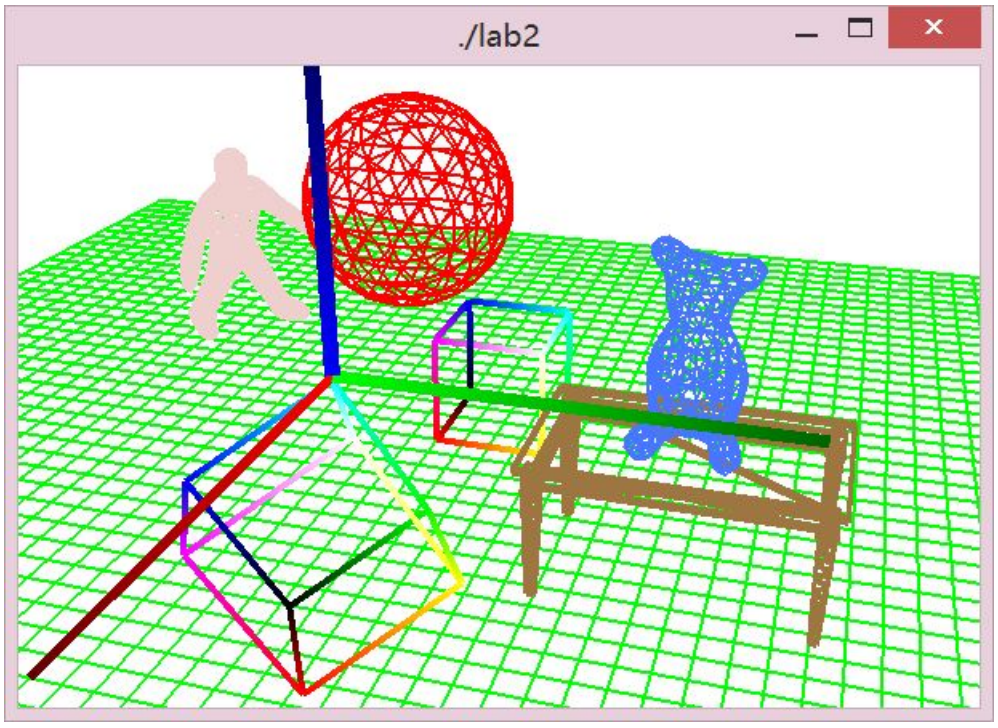
```
    int    nlines; /* Number of lines */
    LINE *lines;
    int    npolylines; /* Number of the polylines */
    POLYLINE *polylines;
    int    ntriangles;
    TRIANGLE *triangles;

    Vector eye;
    Vector gaze;
    Vector upVector;
    int pjType; /* which projection to be done*/
    ORTHO ortho;
    PERSP persp;
    FLOOR floor;
    int isAxis;
    AXIS axis;
    int nidentities;
    IDENTITY *identities;
    TRANSLATE *translate;
    ROTATE *rotate;
    SCALE *scale;
    MESH *mesh;

} SCENE;

typedef struct {
    VERTEX position;
} CAMERA;

#if defined(SSD_UTIL_SOURCE_CODE)
#define EXTERN_FLAG
#else
#define EXTERN_FLAG extern
extern struct ssd_keyword keyword_table[];
#endif

EXTERN_FLAG
int match_Keyword(char *keyword, int *npara);

EXTERN_FLAG
int readAndParse(FILE *inFilePtr, char *keyword, char *arg0,
                 char *arg1, char *arg2, char *arg3, char *arg4);
EXTERN_FLAG
int Read_SSD_Scene(char *fname, SCENE *ascene, char *saved_fname);
#undef    EXTERN_FLAG
#endif
```

3. Result(screen shots)


./lab2


./lab2





4. Conclude

It is my first time to modify the ssd file. At the beginning, it hard for me to read those data. I t took me a long time to know what I should do. And it is a complex work to get all things done, especially the part of matrix. It is easy to get confused for so many variables. But finally, I did it. From this project I get familiar with transformation matrix -scale, rotation, shear, translate- and also the the process of projection. I got improved from this project.