Comp380 Report : PA4

Programming Assignment #4

Name: In-Young Cho (조인영)
ID : 20150720

● 일단 cpp파일에서 //edit ~ //end라고 묶여져 있는 부분은 모두 제가 수정한 코드입니다.
● PA2, PA3에서 더 수정한 것 만 설명하겠습니다.

```
#include <cstdlib> // for using rand function
#include <ctime>
-----------------------------------------------------------------------------------------------
//edit
//color에 관한 매크로
#define cowHex 0xFF54A7
#define betHex 0x9BEF05
#define stmHex 0x00B5EC
#define floorHex 0xB68B68
//end
-----------------------------------------------------------------------------------------------
//edit
// Variables for 'stormtrooper' object.
FrameXform stm2wld;
wavefront_obj_t *stm;
int stmID;
//end
-----------------------------------------------------------------------------------------------
float oldAngle = 45; // zoom기능을 구현하는 데 카메라의 fov 각을 저장하는데 쓰일 변수
int pdot = 0; // 0: nothing, 1: pan, 2: dolly, etc..
int selectOn = 0; // disable/enable selection mode
int selObjInx = 1; // 선택된 오브젝트의 인덱스. 이걸로 구별. 1: cow, 2: bet, 3: stm, 4:  current cam
int onX = 1; int onY = 0; int onZ = 0; // K키가 눌러지면 onK는 1, 아니라면 0
int tmpOnX = onX; int tmpOnY = onY; int tmpOnZ = onZ; // r 키가 눌러졌을 때 onK의 값을 tmpOnK에 임시저장하고 onK는 0으로. 그러면 회전도중 드래그를 해도 소는 움직이지 않음. 다시 r키가 눌러지면 저장했던 값을 onK에.
GLdouble angleX = 0; GLdouble angleY = 0; GLdouble angleZ = 0; // glRotated의 x,y,z 자리에 들어갈 글로벌 변수
int isRotate = 0; // 회전모드에서 1, 회전모드가 아닐 때 0
int modelOrView = 0; // 변환을 원하는 공간이 modelSpace면 0, viewSpace면 1
-----------------------------------------------------------------------------------------------
//edit
void drawStm() {
        if (frame == 0) {

                stm = new wavefront_obj_t("stormtrooper.obj");

                stmID = glGenLists(1);
                glNewList(stmID, GL_COMPILE);
                stm->draw();
                glEndList();
                glPushMatrix();
                glLoadIdentity();
                glTranslated(0, -stm->aabb.first[1], 0);
                glRotated(180, 0, 1, 0);
```

```
                glGetDoublev(GL_MODELVIEW_MATRIX, stm2wld.matrix());
                glPopMatrix();
        }

        glPushMatrix();
        glMultMatrixd(stm2wld.matrix());
        if (selectMode == 0) {
                drawFrame(8);
                //edit
                drawRotFrame(selObjInx == 3 ? 5 : 0); // 선택되지 않은 오브젝트의 축은 표현할 필요가 없다.
                //end
                float frontColor[] = { 0.8, 0.5, 0.5, 1.0 }; // 색상은 임의로 설정
                glEnable(GL_LIGHTING);
                glMaterialfv(GL_FRONT, GL_AMBIENT, frontColor);
                glMaterialfv(GL_FRONT, GL_DIFFUSE, frontColor);
        }
        else {
                glDisable(GL_LIGHTING);
                drawRotFrame(selObjInx == 3 ? 5 : 0);
                double3 color = munge(stmHex); // 색상설정
                glColor3dv(color.data());
        }
        glCallList(stmID);
        glPopMatrix();
}
//end
-----------------------------------------------------------------------------------------------
void onMouseButton( int button, int state, int x, int y ) {
    y = height - y - 1;
    if ( button == GLUT_LEFT_BUTTON ) {
        if ( state == GLUT_DOWN ) {
            printf( "Left mouse click at (%d, %d)\n", x, y );
            int tmp = selectMode; //edit 현재의 선택모드를 임시저장
            selectMode = 1;
            display();
            glReadBuffer( GL_BACK );
            unsigned char pixel[3];
            glReadPixels( x, y, 1, 1, GL_RGB, GL_UNSIGNED_BYTE, pixel );
            double3 pixel_d3 = { double(pixel[0]), double(pixel[1]), double(pixel[2]) };
            printf("pixel = %d\n", unmunge(pixel_d3));
            selectMode = tmp; //edit 백버퍼를 보여주는 상태에서 선택모드를 0으로 바꾸어버리면 안되므로 위에서 저장
해둔 tmp를 다시 할당.
                        switch(unmunge(pixel_d3)*(selectOn)) { // selection이 disable되면 가장 최근에 선택된 오브
젝트를, enable되면 마우스로 클릭한 오브젝트를 인덱스에 저장.
                        case 0:
                                break;
                        case cowHex:
                                selObjInx = 1;
                                break;
                        case betHex:
```

```
                        selObjInx = 2;
                        break;
                case stmHex:
                        selObjInx = 3;
                        break;
                default:
                        selObjInx = 4;
                        break;
                }

                if (isRotate == 1) {
                        if (selectOn) {
                                if (unmunge(pixel_d3) == cowHex || unmunge(pixel_d3) == betHex ||
unmunge(pixel_d3) == stmHex) {
                                        setRotationAxis();
                                        if (modelOrView == 0)           glutIdleFunc(renderRotation);
//modeling 공간에서 회전을 원하면 IdleFunc실행
                                }
                                else {
                                        if (modelOrView == 0)           glutIdleFunc(NULL);  //  stop
rotating
                                }
                        }
                        else {
                                //nothing
                        }
                }
                //회전할 때, 물체 선택 기능이 켜져있으면, 바탕을 클릭하면 물체의 회전을 멈추고 다른 물체를
클릭하면 그 물체를 회전시킨다. 선택 기능이 꺼져있으면 그냥 계속돈다.
                //end
            // Save current clicked location of mouse here, and then use this on onMouseDrag function.
            oldX = x;
            oldY = y;
        }
    } else if ( button == GLUT_RIGHT_BUTTON ) {
        printf( "Right mouse click at (%d, %d)\n", x, y );
    }

    if (selectMode) {
        display();
        glutSwapBuffers();
    }


    glutPostRedisplay();
}
-----------------------------------------------------------------------------------------------
//edit
FrameXform* selObj2Wld() { // 선택된 물체의 오브젝트 공간 to 월드 공간 매트릭스의 주소를 반환
        switch (selObjInx) {
```

```
		case 1:
			return &cow2wld;
			break;
		case 2:
			return &bet2wld;
			break;
		case 3:
			return &stm2wld;
			break;
		case 4:
			return &cam2wld[cameraIndex];
			break;
		default: //이런 경우는 일어나지 않는다.
			printf("error\n");
		}
}
-----------------------------------------------------------------------------------------------
void setNormalized(double* a, double* b, double* c) { // 노멀 벡터 계산
		double norm = sqrt((*a)*(*a) + (*b)*(*b) + (*c)*(*c));
		*a /= norm; *b /= norm; *c /= norm;
}

void crossProduct(double * A, double * B, double * C) { // 외적 계산
		C[0] = A[1] * B[2] - A[2] * B[1];
		C[1] = A[2] * B[0] - A[0] * B[2];
		C[2] = A[0] * B[1] - A[1] * B[0];
		setNormalized(&(C[0]), &(C[1]), &(C[2]));
}

double getDotProduct(double *A, double *B) { // 내적 계산
		return A[0] * B[0] + A[1] * B[1] + A[2] * B[2];
}

double z(double x, double y) { // 트랙볼 구현을 위해 x,y에서 깊이 z 계산. 반지름은 임의 설정 가능하며, 반지름보다 큰 변화를 주었을 때를 대비하여 구간을 나눠야한다. (물론, 반지름을 화면을 꽉 채울 만큼 크게 줄 수도 있다) 하나는 구면, 하나는 분수함수
		double x2 = x*x;
		double y2 = y*y;
		double r2 = 1000000;  // 용도에 맞게 값 변경
		if (x2 + y2 <= r2 * 0.5) {
			return sqrt(r2 - (x2 + y2));
		}
		else {
			return r2 * 0.5 / sqrt(x2 + y2);
		}
}

void trackballProject(FrameXform* obj2wldPtr, double x, double y, double* vec) {// 물체의 원점과 화면에 선택된 점 x,y 간의 관계를 이용해 트랙볼 계산에 사용할 벡터를 계산한다. z는 위의 함수를 쓴다.
		double center[2] = { 0,0 };
```

```
        FrameXform tmp;
        glPushMatrix();
        glLoadIdentity();
        glMultMatrixd(wld2cam[cameraIndex].matrix());
        glMultMatrixd((*obj2wldPtr).matrix());
        glGetDoublev(GL_MODELVIEW_MATRIX, tmp.matrix());
        glPopMatrix(); // 물체 좌표상 원점을 캠 좌표로 옮긴다.

        center[0] = tmp.matrix()[12] * 1;
        center[1] = tmp.matrix()[13] * 1; // 원점은 (0,0,0,1)이므로

        x = x - width * 0.5 - center[0];
        y = y - height * 0.5 - center[1]; // 원점을 기준으로 다시 x,y 좌표를 계산한다. 원점이 회전의 중심이 되기 때문이다.
        vec[0] = x;         vec[1] = y;         vec[2] = z(x, y);
        setNormalized(&(vec[0]), &(vec[1]), &(vec[2]));
}

void trackballRotate(FrameXform* obj2wldPtr, double x1, double y1, double x2, double y2, double* arr) { // 슬라이드에 있는 내용 그대로. 외적으로 수직한 벡터(회전축)를 찾고, 내적으로 회전각을 찾는다.
        double v1[3], v2[3], normal[3];
        trackballProject(obj2wldPtr, x1, y1, v1);
        trackballProject(obj2wldPtr, x2, y2, v2);
        crossProduct(v1, v2, normal);
        double theta = acos(getDotProduct(v1, v2)); // dot product of v1 and v2, v1,v2의 크기는 이미 1
        arr[0] = theta; arr[1] = normal[0]; arr[2] = normal[1]; arr[3] = normal[2];
}
-----------------------------------------------------------------------------------------------
void pan(int x, int y) { // 화면에서 움직인 x- oldX, y - oldY 의 양만큼 eyex, eyey, eyez 와 centerx, centery, centerz를 옮기는 것이 pan이다. 캠 공간에서의 (x- oldX, y - oldY, 0, 0)가 월드 공간에서 어떤 벡터가 되는지 계산한다.
        auto &camera = cameras[cameraIndex];
        glPushMatrix();
        glLoadIdentity();

        double tmp0 = (cam2wld[cameraIndex].matrix()[0] * (x - oldX) * 0.05 + cam2wld[cameraIndex].matrix()[4] * (y - oldY) * 0.05);
        double tmp1 = (cam2wld[cameraIndex].matrix()[1] * (x - oldX) * 0.05 + cam2wld[cameraIndex].matrix()[5] * (y - oldY) * 0.05);
        double tmp2 = (cam2wld[cameraIndex].matrix()[2] * (x - oldX) * 0.05 + cam2wld[cameraIndex].matrix()[6] * (y - oldY) * 0.05); //  ((x - oldX) * 0.05, (y - oldY) * 0.05), 0) 만큼 translate

        cameras[cameraIndex][0] = camera[0] - tmp0; cameras[cameraIndex][1] = camera[1] - tmp1; cameras[cameraIndex][2] = camera[2] - tmp2;
        cameras[cameraIndex][3] = camera[3] - tmp0; cameras[cameraIndex][4] = camera[4] - tmp1; cameras[cameraIndex][5] = camera[5] - tmp2;

        gluLookAt(camera[0], camera[1], camera[2], camera[3], camera[4], camera[5], camera[6], camera[7], camera[8]);                // Setting the coordinate of camera.
        glGetDoublev(GL_MODELVIEW_MATRIX, wld2cam[cameraIndex].matrix());                // Read the world-to-camera matrix computed by gluLookAt.
```

```
	glPopMatrix();
		// Transfer the matrix that was pushed the stack to GL.
	cam2wld[cameraIndex] = wld2cam[cameraIndex].inverse();
	oldX = x; oldY = y;
	glutPostRedisplay();
}

void dolly(int x, int y) { 화면에서 움직인 y - oldY 의 양만큼 eyex, eyey, eyez 를 옮기는 것이 dolly이다. 캠 공간에서의 (0, 0, y - oldY, 0)가 월드 공간에서 어떤 벡터가 되는지 계산한다.
	auto &camera = cameras[cameraIndex];
	glPushMatrix();
	glLoadIdentity();

	double tmp0 = (cam2wld[cameraIndex].matrix()[8] * (y - oldY) * 0.05);
	double tmp1 = (cam2wld[cameraIndex].matrix()[9] * (y - oldY) * 0.05);
	double tmp2 = (cam2wld[cameraIndex].matrix()[10] * (y - oldY) * 0.05);

	cameras[cameraIndex][0] = camera[0] - tmp0; cameras[cameraIndex][1] = camera[1] - tmp1; cameras[cameraIndex][2] = camera[2] - tmp2;
	cameras[cameraIndex][3] = camera[3] - tmp0; cameras[cameraIndex][4] = camera[4] - tmp1; cameras[cameraIndex][5] = camera[5] - tmp2;

	gluLookAt(camera[0], camera[1], camera[2], camera[3], camera[4], camera[5], camera[6], camera[7], camera[8]);		// Setting the coordinate of camera.
	glGetDoublev(GL_MODELVIEW_MATRIX, wld2cam[cameraIndex].matrix());		// Read the world-to-camera matrix computed by gluLookAt.
	glPopMatrix();
		// Transfer the matrix that was pushed the stack to GL.
	cam2wld[cameraIndex] = wld2cam[cameraIndex].inverse();
	oldX = x; oldY = y;
	glutPostRedisplay();
}

void zoom(int x, int y) { // 카메라의 vertical fov를 늘리면 그만큼 한 눈에 들어오는 사물도 많아져서 zoom out
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	double aspect = width / double(height);
	gluPerspective(oldAngle - (y - oldY)*0.05, aspect, 1, 1024);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	oldAngle = oldAngle - (y - oldY)*0.05;
	oldX = x; oldY = y;
	glutPostRedisplay();
}

void trackball(int x, int y) {
// reference 에서는 world space의 원점을 중심으로 회전하지만 저는 카메라의 센터를 중심으로 회전하는게 더 유용하다 생각하여 그렇게 구현해보았습니다. 따라서 카메라의 센터에 물체를 놓으면 물체를 월드에 대해 움직이지 않아도 둘러볼 수 있습니다.
// center를 중심으로 하는 구면 위에서 eye_x, eye_y, eye_z를 움직인다. 이에 따라 upvector도 다시 계산하여야 하는데,
```

그 이유는 upvector를 새로 계산하지 않으면 위에서 바라보려 할 때 부드럽게 움직이지 못하는 문제가 생기기 때문. 새로 구해진 벡터 eye - center 과 upvector를 이용하여 둘에 수직인 축(cam's x axis)을 구하고 다시 새로운 upvector'를 구한다. 이로써 새로운 cam space가 만들어진다.

```
	double oldR, newR, t;
	auto &camera = cameras[cameraIndex];
	glPushMatrix();
	glLoadIdentity();

	oldR = sqrt(pow(camera[0]-camera[3], 2) + pow(camera[1] - camera[4], 2) + pow(camera[2] - camera[5], 2));

	double tmp0 = (cam2wld[cameraIndex].matrix()[0] * (x - oldX) * 0.05 + cam2wld[cameraIndex].matrix()[4] * (y - oldY) * 0.05);
	double tmp1 = (cam2wld[cameraIndex].matrix()[1] * (x - oldX) * 0.05 + cam2wld[cameraIndex].matrix()[5] * (y - oldY) * 0.05);
	double tmp2 = (cam2wld[cameraIndex].matrix()[2] * (x - oldX) * 0.05 + cam2wld[cameraIndex].matrix()[6] * (y - oldY) * 0.05); // translate like panning
	cameras[cameraIndex][0] = camera[0] - tmp0; cameras[cameraIndex][1] = camera[1] - tmp1; cameras[cameraIndex][2] = camera[2] - tmp2; // e_x, e_y, e_z

	newR = sqrt(pow(camera[0] - camera[3], 2) + pow(camera[1] - camera[4], 2) + pow(camera[2] - camera[5], 2));

	t = oldR / newR;

	cameras[cameraIndex][0] = t * camera[0] + (1 - t) * camera[3]; cameras[cameraIndex][1] = t * camera[1] + (1 - t) * camera[4]; cameras[cameraIndex][2] = t * camera[2] + (1 - t) * camera[5]; // Interpolating, 구면 위로의 사영.

	double tmp3[3] = { camera[0] - camera[3], camera[1] - camera[4], camera[2] - camera[5] }; // eye - center vector
	double tmp4[3] = { camera[6], camera[7], camera[8] }; // upvector
	double tmp5[3] = {0,0,0};
	crossProduct(tmp4, tmp3, tmp5);
	crossProduct(tmp3, tmp5, tmp4);

	cameras[cameraIndex][6] = tmp4[0]; cameras[cameraIndex][7] = tmp4[1]; cameras[cameraIndex][8] = tmp4[2];

	gluLookAt(camera[0], camera[1], camera[2], camera[3], camera[4], camera[5], camera[6], camera[7], camera[8]);		// Setting the coordinate of camera.
	glGetDoublev(GL_MODELVIEW_MATRIX, wld2cam[cameraIndex].matrix());		// Read the world-to-camera matrix computed by gluLookAt.
	glPopMatrix();
		// Transfer the matrix that was pushed the stack to GL.
	cam2wld[cameraIndex] = wld2cam[cameraIndex].inverse();
	oldX = x; oldY = y;
	glutPostRedisplay();
}
```

```
void rotAxisCam2Obj(FrameXform* obj2wldPtr, double x, double y, double z) { // 카메라 좌표상의 임의의 회전축 (벡터)을 월드 좌표계의 벡터로 변환한다. v키를 누른 뒤 물체를 화면상 x 축에 맞춰 돌릴 때 등 사용 가능하다.
	FrameXform tmp;
	glPushMatrix();
	tmp = (*obj2wldPtr).inverse(); //wld2obj
	glLoadMatrixd(tmp.matrix());
	glMultMatrixd(cam2wld[cameraIndex].matrix()); // wld2obj * cam2wld = cam2obj
	glGetDoublev(GL_MODELVIEW_MATRIX, tmp.matrix()); // tmp에 위 행렬 저장
	glPopMatrix();
	//이제 tmp 행렬은 cam 좌표계 위의 점 (혹은 벡터)을 obj 좌표계 위로 보내는 아핀변환이다.

	angleX = tmp.matrix()[0] * x + tmp.matrix()[4] * y + tmp.matrix()[8] * z;
	angleY = tmp.matrix()[1] * x + tmp.matrix()[5] * y + tmp.matrix()[9] * z;
	angleZ = tmp.matrix()[2] * x + tmp.matrix()[6] * y + tmp.matrix()[10] * z;
	// that is, tmp * {x,y,z}^t
}
-----------------------------------------------------------------------------------------------
	if (selectMode) { // 백버퍼만을 보여주는 모드의 경우 새로운 그림을 그리고 디스플레이 후 버퍼를 스왑시켜서 항상 우리가 원하는 백버퍼만 화면에 보이도록 한다.
		display();
		glutSwapBuffers();
	}
-----------------------------------------------------------------------------------------------
void onMouseDrag( int x, int y ) {
// 조건의 계층(우선순위) 설정
p, d, o 눌러짐 -> 바탕화면 클릭 -> 바탕이 움직임
p, d, o 눌러짐 -> 물체 클릭 -> 물체는 m,v, 와 onX, onY, onZ, isRotate에 따라 움직임
t 눌러짐 -> 바탕클릭 -> 바탕 움직임
t 눌러짐 -> 물체클릭 -> 물체움직임
r, x, y, z 클릭 -> p,d,o,t 상태제거(pdot <- 0) 및 알맞은 변환

    y = height - y - 1;
    printf( "in drag (%d, %d)\n", x - oldX,  y - oldY );

    // (Project 2,3,4) TODO : Implement here to perform properly when drag the mouse on each case, respectively.
	//edited

	FrameXform* obj2wldPtr = selObj2Wld();

	glPushMatrix();

	if (pdot == 4) {
		if (selObjInx == 4 || selectOn == 0) { //바탕을 클릭했다.
			trackball(x, y);
		}
		else { //물체를 클릭했다.
			glLoadMatrixd((*obj2wldPtr).matrix());
			double arr[4];
			trackballRotate(obj2wldPtr, oldX, oldY, x, y, arr); // cam 공간 상에서 회전축과 각을 구한
```

```
다.
                        rotAxisCam2Obj(obj2wldPtr, arr[1], arr[2], arr[3]); // cam 공간 상 회전축(벡터)을 obj공간의 벡터로 변환한다.
                        glRotated(100 * arr[0], angleX, angleY, angleZ); // 100은 속력 가중치

                        glGetDoublev(GL_MODELVIEW_MATRIX, (*obj2wldPtr).matrix());
                        oldX = x; // 위치를 저장한다
                        oldY = y;
                }
        }
        else if (pdot != 0 && (selObjInx == 4 || selectOn == 0)) { // p,d,o 중 하나를 클릭했다.
                glPopMatrix();
                switch (pdot) {
                case 1:
                        pan(x, y);
                        break;
                case 2:
                        dolly(x, y);
                        break;
                case 3:
                        zoom(x, y);
                        break;
                }
                if (selectMode) {
                        display();
                        glutSwapBuffers();
                }
                return;
        }
        else {
                if (modelOrView == 0 && isRotate == 0) { //만약 modeling space에서 이동을 원한다면,

                        glLoadMatrixd((*obj2wldPtr).matrix()); // 소의 변환 행렬을 로드한다
                        glTranslated((x - oldX)*onX*0.05, (x - oldX)*onY*0.05, (x - oldX)*onZ*0.05);// x - oldX는 드래그 보폭, onK는 어떤 방향으로 움직여야 하는지. 0.1은 임의의 속력 상수

                                                                //obj2wld 행렬을 로드했으므로, 위 arguments 의 값은 소 좌표계 기준이다. 이 translation은 world에 반영된다.
                        glGetDoublev(GL_MODELVIEW_MATRIX, (*obj2wldPtr).matrix());
                }
                else if (modelOrView == 1 && isRotate == 0) { // viewing space에서 이동을 원한다면,
                        glLoadMatrixd(cam2wld[cameraIndex].matrix());
                        glTranslated((x - oldX)*(onX || onY)*0.05, (y - oldY)*(onX || onY)*0.05, (x - oldX)*onZ*0.05); // 'x'또는 'y'를 눌렀다면 x-y plane 상에서 이동. 'z'를 눌렀다면 (캠 좌표상) z축 방향으로 이동.


//위와 마찬가지로, cam2wld[cameraIndex] 행렬을 로드했으므로, 위 arguments 의 값은 캠 좌표계 기준이다.
                        glMultMatrixd(wld2cam[cameraIndex].matrix());
                        glMultMatrixd((*obj2wldPtr).matrix());
                        //위 두 행렬의 곱은 결국 obj2cam 과 동치
```

```
                        //위 두 행렬을 current matrix에 곱해줌으로써 캠 좌표계 기준의 변환을 소 좌표계 기준으로 환산한다. translation을 world에 반영한다.
                        glGetDoublev(GL_MODELVIEW_MATRIX, (*obj2wldPtr).matrix());
                }
                if (isRotate == 1 && modelOrView == 1) { // 돌 수 있는 상태이고, viewing space에서 이동을 원한다면,
                        tmpOnX = onX; tmpOnY = onY; tmpOnZ = onZ; // tmpOnK <-swap-> onK
                        onX = 0; onY = 0; onZ = 0; // 0으로 만들면 평행이동이 작동하지 않는다.
                        setRotationAxis();
                        glLoadMatrixd((*obj2wldPtr).matrix());
                        glRotated((x - oldX), angleX, angleY, angleZ);
                        //renderRotaion() 에서도 썼던 기본적인 modeling space상 회전. 축은 v상태에서 r키를 누를 때 결정된다.
                        glGetDoublev(GL_MODELVIEW_MATRIX, (*obj2wldPtr).matrix());
                }
                oldX = x; // 위치를 저장한다
                oldY = y;
        }

        glPopMatrix();

        if (selectMode) {
                display();
                glutSwapBuffers();
        }
        //end
    glutPostRedisplay();

}


/*****************************************************************************
* Call this part whenever user types keyboard.
* This part is called in main() function by registering on glutKeyboardFunc(onKeyPress).
*****************************************************************************/
void onKeyPress( unsigned char key, int x, int y ) {
    // If 'c' or space bar are pressed, alter the camera.
    // If a number is pressed, alter the camera corresponding the number.
    if ( ( key == ' ' ) || ( key == 'c' ) ) {
        printf( "Toggle camera %d\n", cameraIndex );
        cameraIndex += 1;
                //edit
                if ( isRotate == 1 && modelOrView == 1 ) setRotationAxis(); // viewing space에서 회전하는 중, 카메라를 변경할 때 회전축도 같이 바꿔준다.
                //end
        }
        else if ((key >= '0') && (key <= '9')) {
                cameraIndex = key - '0';
                //edit
                if (isRotate == 1 && modelOrView == 1) setRotationAxis(); // viewing space에서 회전하는 중, 카메
```

```
라를 변경할 때 회전축도 같이 바꿔준다.
                //end
        }

        if (cameraIndex >= (int)wld2cam.size())
                cameraIndex = 0;

    // (Project 2,3,4) TODO : Implement here to handle keyboard input.
        //edited
        if (key == 'p' || key == 'd' || key == 'o' || key == 't') {
                pdot = 1 * (key == 'p') + 2 * (key == 'd') + 3 * (key == 'o') + 4 * (key == 't');  // key가 어떤
값이냐에 따라 pdot을 할당
                if (isRotate && modelOrView == 0) {  // modelSpace에서 Rotate중이면 IdleFunc 중단
                        glutIdleFunc(NULL);
                        isRotate = 0;
                }
        }

        if (key == 'h') {
                printf("========================================\n");
                printf("Key Map\n");
                printf("========================================\n");
                printf("s: selection mode toggle\n"
                        "x, y, z: translate along each axis\n"
                                "r: rotate\n"
                                "v: viewing space\n"
                                "m: modeling space\n"
                                "p: pan\n"
                                "d: dolly\n"
                                "o: zoom\n"
                                "t: trackball\n"
                                "b: toggle show back buffer\n");
                printf("========================================\n");
        }

        if (key == 's') selectOn = 1 - selectOn; // object selection enable/disable

        if (key == 'b') {
                selectMode = 1 - selectMode; // back buffer mode on/off
        }

        if (key == 'v') {
                if (isRotate == 1 && modelOrView == 0) { // modeling space에서 회전 중 이면,
                        glutIdleFunc(NULL); // stop rotating
                        modelOrView = 1;
                        setRotationAxis();
                        //일단 회전을 멈추고 viewing space에서 축을 설정하고 기다린다.
                }
                modelOrView = 1;
        }
```

```
	if (key == 'm') {
		if (isRotate == 1 && modelOrView == 1) { // viewing space에서 회전 중 이면,
			modelOrView = 0;
			setRotationAxis();
			glutIdleFunc(renderRotation);
			//modeling space에서 축을 설정한 뒤 계속 회전한다.
		}
		modelOrView = 0;
	}

	if (key == 'x' || key == 'y' || key == 'z'){
		pdot = 0;
		if (isRotate == 1) { //회전 중이면 회전을 멈춰라.
			if (modelOrView == 0)		glutIdleFunc(NULL); // stop rotating
			isRotate = 0;
		}
		onX = (key == 'x');	onY = (key == 'y'); onZ = (key == 'z'); // 각 키가 눌러지면, 해당하는 onK 를 1로, 아니면 0으로
	}

	if (key == 'r') {
		pdot = 0;
		if (isRotate == 1) { //회전 중이면 회전을 멈춰라. 이전 translation 방향을 유지하라
			if (modelOrView == 0)		glutIdleFunc(NULL); // stop rotating
			isRotate = 0;
			onX = tmpOnX; onY = tmpOnY; onZ = tmpOnZ; // re-assignment
		}
		else {
			tmpOnX = onX; tmpOnY = onY; tmpOnZ = onZ; // tmpOnK <-swap-> onK
			onX = 0; onY = 0; onZ = 0; // 0으로 만들면 평행이동이 작동하지 않는다.
			isRotate = 1;
			setRotationAxis();
			if (modelOrView == 0)		glutIdleFunc(renderRotation); //modeling 공간에서 회전을 원하면 IdleFunc실행
		}
	}
	//end

	if (selectMode) {
		display();
		glutSwapBuffers();
	}

    glutPostRedisplay();
}
```