BLG444E

Computer Graphics

2014 Spring

Report of Project 2

Date of Submission : 29.02.2014

040100118 Volkan İlbeyli
040100043 Mert Tepe

Instructor : Uluğ Bayazıt

Since this is the very basic 3D hello world program of OpenGL, two of us did the entire project on our own and then in the end, combined our work into one project for the purpose of either of us having learnt the basics of OpenGL and 3D graphics.

Data Structures & Initial Setup

In this stage, we simply added a second cube to the right and moved the initial cube a to the left. The second line specifies the cubes' color vectors.

```
static Matrix4 g_objectRbt[2] = {
	Matrix4::makeTranslation(Cvec3(-.85,0,0)),
	Matrix4::makeTranslation(Cvec3(.85,0,0))
};  // currently only 1 obj is defined

static Cvec3f g_objectColors[2] = {Cvec3f(1, 0, 0), Cvec3f(.051,.75,.137)};
```

The corresponding drawing code is added to `drawStuff()` function's body, right after the first draw call for red cube.

```
  MVM = invEyeRbt * g_objectRbt[1];
  NMVM = normalMatrix(MVM);
  sendModelViewNormalMatrix(curSS, MVM, NMVM);
  safe_glUniform3f(curSS.h_uColor, g_objectColors[1][0], g_objectColors[1][1],
g_objectColors[1][2]);
  g_cube->draw(curSS);
```

LinFact() and TransFact() Functions

TransFact() function takes the upper-left-3x3 part of a Matrix4, while LinFact() function takes the first 3 values from the rightmost column, representing the translational and rotational factors of a translation, respectively. Both functions override these values onto the corresponding row/columns on an identity matrix of 4x4.

```
inline Matrix4 transFact(const Matrix4& m) {
	Matrix4 transM = Matrix4();	// identity matrix
	for(int i=0; i<3 ; ++i)
		transM(i, 3) = m(i, 3);		// override the last column only
	return transM;
}

inline Matrix4 linFact(const Matrix4& m) {
	Matrix4 linM = Matrix4();  // identity matrix
	for(int i=0; i<3 ; ++i){   // override the upper left 3x3 matrix area
		for(int j=0; j<3; ++j){
			linM(i,j) = m(i,j);
		}
	}

	return linM;
}
```

Views

A global variable is declared for maintaining the views.

```
// states: 0=eye, 1=redcube, 2=greencube
static unsigned viewState = 0;
```

Whenever ‘v’ key is pressed, the stated is incremented. Note that the `viewState` variable can have 3 values: 0, 1 and 2.

```
case 'v':
       viewState++;
       viewState %= 3;
       break;
```

The eye is assigned to either of the cubes or the sky camera according to the value of the `viewState` varible when executing a `drawStuff()`call.

```
  // assign eye rigid body matrix;
  Matrix4 eyeRbt;
  switch(viewState){
  case 0:
         eyeRbt = g_skyRbt;
         break;
  case 1:
         eyeRbt = g_objectRbt[0];
         break;
  case 2:
         eyeRbt = g_objectRbt[1];
         break;
  }
```

Object Manipulation

To keep track of which object is manipulated and the manipulation modes (for sky camera), two global variables are introduced.

```
// states: 0=eye, 1=redcube, 2=greencube
static unsigned manipState = 1;
static bool worldSkyFrame = true;
```

The function `doMtoOwrtA()` added for manipulation for different frames and objects.

```
static void doMtoOwrtA(Matrix4 m, Matrix4& o, Matrix4 a){
       Matrix4 A = transFact(o)*linFact(a);
       o = A * m * inv(A) * o;
}
```

Note that this function is **not** called when manipulating the sky camera while the view is the sky camera (case 0 below), since the manipulation criteria doesn’t comply to the function we defined above. *The code segments below are from the `void motion(const int x, const int y)` function.*

```
if (g_mouseClickDown) {
	switch(manipState){ // manipulation state: 0=sky, 1=redCube, 2=greenCube
	case 0:			// sky camera is manipulated
		if(viewState == 0){		// sky camera is the eye
			Matrix4 A;
			if(worldSkyFrame)	// if world-sky frame
				A = transFact(m)*linFact(g_skyRbt);
			else	// if sky-sky frame
				A = transFact(g_skyRbt)*linFact(g_skyRbt);

			g_skyRbt = A*m*inv(A)*g_skyRbt;	// apply the transform
		}
		break;
	case 1:			// red cube is manipulated
		if(viewState == 0)	// if sky camera is the eye
			doMtoOwrtA(m, g_objectRbt[0], g_skyRbt); // cube-sky frame

		else if(viewState == 2)	//if green cube is the eye
			doMtoOwrtA(m, g_objectRbt[0], g_objectRbt[1]);  //cube i - cube j

		break;
	case 2:			// green cube is manipulated
		if(viewState == 0)		// if sky camera is the eye
			doMtoOwrtA(m, g_objectRbt[1], g_skyRbt); // cube-sky frame

		else if(viewState == 1){	//if red cube is the eye
			doMtoOwrtA(m, g_objectRbt[1], g_objectRbt[0]);  //cube j - cube i

		}
		break;
	}
```

The sign inversions for sky camera manipulation are done when assigning the m matrix as follows.

```
Matrix4 m;
if (g_mouseLClickButton && !g_mouseRClickButton) { // left button down?
	m = Matrix4::makeXRotation(-dy) * Matrix4::makeYRotation(dx);

	// if sky camera is manipulated override the m matrix as signs inverted
	if(manipState == 0)
		m = Matrix4::makeXRotation(dy) * Matrix4::makeYRotation(-dx);

}
else if (g_mouseRClickButton && !g_mouseLClickButton) { // right button down?
	m = Matrix4::makeTranslation(Cvec3(dx, dy, 0) * 0.01);

	//if we are manipulating sky camera, override the m matrix
	if(manipState == 0 && worldSkyFrame == true){
		m = Matrix4::makeTranslation(Cvec3(-dx, -dy, 0) * 0.01);
	}
}
else if (g_mouseMClickButton || (g_mouseLClickButton && g_mouseRClickButton)) {  //
middle or (left and right) button down?
	m = Matrix4::makeTranslation(Cvec3(0, 0, -dy) * 0.01);

	//if we are manipulating sky camera, override the m matrix
	if(manipState == 0 && worldSkyFrame == true){
		m = Matrix4::makeTranslation(Cvec3(0, 0, dy) * 0.01);
	}
}
```