9.조명

9-1.조명

실세계의 사물은 고유의 색상을 가진다. 색상이란 물체 자체에서 나오는 것이 아니라 물체 표면이 주변에 존재하는 빛의 성분 중 어떤 것을 반사하는가로 결정된다. 빨간색을 잘 반사하면 빨간색 물체로 보이고 모든 빛을 잘 반사하면 흰색이며 모든 빛을 흡수하면 검정색이다. 눈이 인식하는 색상은 여러 빛의 혼합과 물체의 색상 반사 재질에 의해 결정된다.

한 물체의 색상은 굉장히 다양한 요소들에 의해 복잡하게 결정되므로 단일한 색인 경우보다 여러 색이 섞여 보이는 것이 일반적이다. 3D 그래픽은 실세계의 사물을 화면에 그려내는 기술이다. 사실적인 표현을 위해서는 주변빛에 따라 물체의 색상을 사실적으로 그려내야 한다. OpenGL은 이를 위해 조명을 성격에 따라 다음 세가지로 분류한다. 조명의 성질에 따라 물체의 색상에 끼치는 영향이 다르다.

- 주변광(Ambient) : 도처에 존재하는 빛이다. 광원에서 나온 빛이 여러 경로로 반사 및 재 반사되어 방향성을 잃어 버린 빛이다. 사방에서 물체의 모든 면에 골고루 비쳐지며 밝기도 일정하다.
- 분산광(Diffuse) : 한 방향으로 들어와서 물체의 표면에 반사되어 여러 방향으로 흩어지는 빛이다. 빛을 받는 부분이 그렇지 않은 부분에 비해 훨씬 더 밝게 보인다. 형광등 불빛이 대표적이다.
- 반사광(Specular) : 한 방향으로 들어와서 한 방향으로만 반사되는 빛이다. 강한 후레쉬 불빛이나 레이저 광선이 이에 해당한다.

빛을 내는 광원은 특정 한 조명에 속하지 않고 이 세가지 조명을 적절히 혼합한 상태이되 성질에 따라 특정 조명의 비율이 다르다. 예를 들여 야외의 햇빛은 난반사되어 도처에 있는 주변광의 성질을 띄지만 틈새로 비치는 직사 광선은 반사광의 성질을 띄기도 한다. 조명 효과를 사용하려면 조명 기능을 켜야 한다.

```
glEnable(GL_LIGHTING);
```

OpenGL은 보통 GL_LIGHT0 ~ GL_LIGHT7까지 8개의 동시 조명을 지원하며 구현에 따라서는 더 많은 수의 조명을 지원하는 것도 있다. 모든 조명은 디폴트로 꺼져 있으므로 사용하고자 하는 조명을 개별적으로 켜주어야 한다. 첫 번째 조명을 사용하려면 다음과 같이 호출한다.

```
glEnable(GL_LIGHT0);
```

즉, 조명을 사용하려면 GL_LIGHTING으로 OpenGL 자체의 전체 조명 기능을 켜고 사용하고자 하는 번호의 조명도 켜야 하므로 최소한 두번은 glEnable을 호출해야 한다. 각 조명에 대한 성질은 다음 함수로 지정한다.

void glLight[f, i](GLenum light, GLenum pname, GLfloat param);

첫번째 인수는 설정하고자 하는 조명의 번호이다. GL_LIGHT0 ~ GL_LIGHT7중의 하나이다. pname은 설정할 속성의 이름이며 param은 속성의 값이다. 함수 하나로 파라미터를 바꾸어 가며 조명의 여러 가지 성질을 지정한다.

속성	설명

GL_AMBIENT	주변광의 세기. (r, g, b, a) 배열로 각 색상 요소의 강도를 지정한다.
GL_DIFFUSE	분산광의 세기.
GL_SPECULAR	반사광의 세기.
GL_POSITION	조명의 (x, y, z, w) 위치
GL_SPOT_DIRECTION	조명의 (x, y, z) 방향
GL_SPOT_EXPONENT	스포트라이트 지수
GL_SPOT_CUTOFF	스포트라이트의 확산각도
GL_CONSTANT_ATTENUATION GL_LINEAR_ATTENUATION GL_QUADRATIC_ATTENUATION	빛이 점점 흐려지는 감쇠율

세가지 조명의 디폴트는 모두 (0,0,0,1) 즉, 검정색이다. 단 첫번째 조명인 GL_LIGHT0는 분산광과 조명광의 디폴트가 (1,1,1,1)로 되어 있다. 즉, 첫번째 조명만 디폴트로 하얀색이다. 다음 예제는 첫번째 조명을 배치하고 위치 및 주변광 설정을 조정한다.

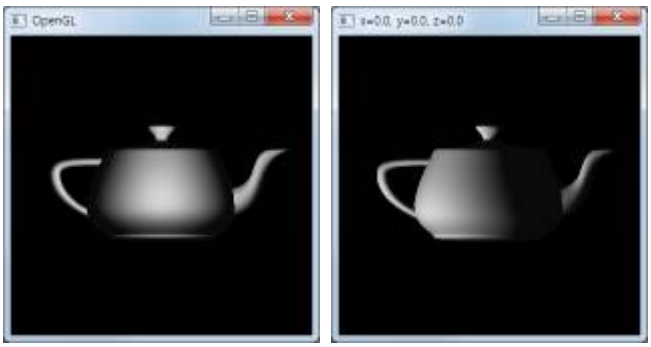
Lighting

```
#nclude <windows.h>
#nclude <gl/glut.h>
#nclude <stdio.h>

void DoDisplay();
void DoKeyboard(unsigned char key, int x, int y);
void DoMenu(int value);
GLfloat lx, ly, lz = -1.0;
GLfloat xAngle, yAngle, zAngle;
GLboolean bAmbient;
GLboolean bAttach;

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
      glutCreateWindow("OpenGL");
      glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
      glutDisplayFunc(DoDisplay);
      glutCreateMenu(DoMenu);
      glutAddMenuEntry("Ambient ON",1);
      glutAddMenuEntry("Ambient OFF",2);
      glutAddMenuEntry("Attach light",3);
      glutAddMenuEntry("Unattach light",4);
      glutAttachMenu(GLUT_RIGHT_BUTTON);
      glutKeyboardFunc(DoKeyboard);
      glutMainLoop();
      return 0;
}

void DoKeyboard(unsigned char key, int x, int y)
{
      switch(key) {
      case 'a':yAngle += 2;break;
      case 'd':yAngle -= 2;break;
      case 'w':xAngle += 2;break;
      case 's':xAngle -= 2;break;
      case 'q':zAngle += 2;break;
      case 'e':zAngle -= 2;break;
```

```
	case 'z':xAngle = yAngle = zAngle = 0.0;break;

	case 'j':lx -= 0.1;break;
	case 'l':lx += 0.1;break;
	case 'i':ly += 0.1;break;
	case 'k':ly -= 0.1;break;
	case 'u':lz += 0.1;break;
	case 'o':lz -= 0.1;break;
	case 'm':lx = 0, ly = 0, lz = -1.0;break;
	}
	char info[128];
	sprintf(info, "x=%.1f, y=%.1f, z=%.1f", xAngle, yAngle, zAngle);
	glutSetWindowTitle(info);
	glutPostRedisplay();
}

void DoMenu(int value)
{
	switch(value) {
	case 1:
		bAmbient = GL_TRUE;
		break;
	case 2:
		bAmbient = GL_FALSE;
		break;
	case 3:
		bAttach = GL_TRUE;
		break;
	case 4:
		bAttach = GL_FALSE;
		break;
	}
	glutPostRedisplay();
}

void DoDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glEnable(GL_DEPTH_TEST);
	glPushMatrix();

	if (bAttach) {
		glRotatef(xAngle, 1.0f, 0.0f, 0.0f);
		glRotatef(yAngle, 0.0f, 1.0f, 0.0f);
		glRotatef(zAngle, 0.0f, 0.0f, 1.0f);
	}

	// 0번 광원 배치.
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	GLfloat lightpos[] = {lx, ly, lz, 1};
	glLightfv(GL_LIGHT0, GL_POSITION, lightpos);

	// 주변광을 초록색으로 설정
	if (bAmbient) {
		GLfloat ambient[4]={0,1,0,1};
		glLightfv(GL_LIGHT0,GL_AMBIENT,ambient);
	} else {
		GLfloat ambient[4]={0,0,0,1};
		glLightfv(GL_LIGHT0,GL_AMBIENT,ambient);
```

```
        }

        if (bAttach == false) {
                glRotatef(xAngle, 1.0f, 0.0f, 0.0f);
                glRotatef(yAngle, 0.0f, 1.0f, 0.0f);
                glRotatef(zAngle, 0.0f, 0.0f, 1.0f);
        }

        glutSolidTeapot(0.5);

        glPopMatrix();
        glFlush();
}
```

전체적인 조명 기능을 켜고 0번 조명을 활성화했다. 조명의 위치는 디폴트로 0, 0, -1로 하되 실행중에 ijkluo 키로 옮길 수 있다. 최초 정면 앞에서 비추는데 조명을 옮겨 보면 주전자의 다른쪽을 비출 것이다.



GL_LIGHT0는 bAmbient 변수값에 따라 초록색 또는 검정색상을 가지되 디폴트는 검정색으로 해 두었다. 분산광과 반사광은 별도로 지정하지 않았으므로 디폴트인 흰색이 적용된다. 팝업 메뉴의 Ambient ON 명령으로 초록색 주변광을 켤 수 있다. 녹색광이 사방에서 비치므로 주전자의 전체 색상이 초록색으로 바뀔 것이다.

조명을 먼저 배치하고 회전 변환을 적용했으므로 주전자를 회전시키더라도 조명은 같이 이동하지 않는다. 팝업 메뉴에서 Attach light 옵션을 선택하면 회전 변환을 먼저한 후 조명을 배치하므로 조명도 같이 회전된다. 즉, 조명도 물체들과 마찬가지로 변환의 영향을 받는다.

9-2.조명 모델

다음 함수는 조명의 전역적인 성질을 지정한다. 특정 번호의 조명이 아니라 전역 설정이므로 장면 전체에 적용된다.

void glLightModel[f, i][v](GLenum pname, const GLfloat *params);

pname은 지정할 속성이며 params는 속성의 값이다. 설정 가능한 속성은 다음 4가지가 있다.

속성	설명
GL_LIGHT_MODEL_AMBIENT	특정 광원에 소속되지 않은 전역 주변광을 설정한다.
GL_LIGHT_MODEL_COLOR_CONTROL	반사광을 계산하는 것과 적용하는 것을 분리한다.
GL_LIGHT_MODEL_LOCAL_VIEWER	반사광의 각도를 어떻게 계산할 것인가를 지정한다. 이 값이 0이면 조명이 평행하게 비치고 0이 아니면 정점의 방향에 따라 비친다.
GL_LIGHT_MODEL_TWO_SIDE	물체의 뒷면에도 조명을 적용한다. 디폴트는 한쪽면에만 조명이 적용된다.

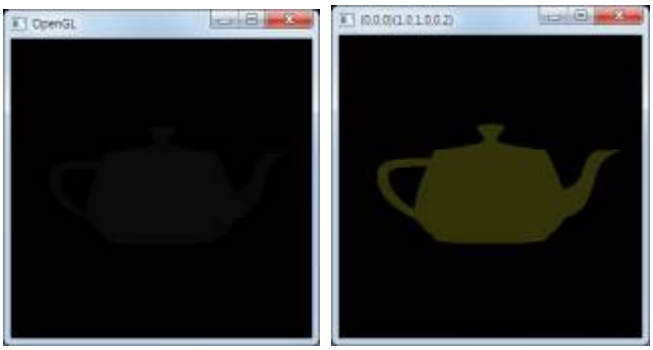
다른 속성은 텍스처 맵핑 등의 특수한 기능과 연관이 있어 여기서 당장 테스트해 보기는 어렵다. 가장 쉽고 자주 사용되는 전역 주변광 옵션만 테스트해 보자. 이 속성의 디폴트는 (0.2, 0.2, 0.2, 1.0)이며 어두운 회색이다. 그래서 조명을 전혀 켜지 않아도 물체가 희미하게 보인다. 물론 더 밝게 하면 좀 더 또렷해질 것이다. 다음 예제는 주전자를 배치해 놓고 전역 주변광만 비춘다.

LightModel

```
#include <windows.h>
#include <gl/glut.h>
#include <stdio.h>

void DoDisplay();
void DoKeyboard(unsigned char key, int x, int y);
GLfloat xAngle, yAngle, zAngle;
GLfloat red=0.2, green=0.2, blue=0.2;

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
	glutCreateWindow("OpenGL");
	glutDisplayFunc(DoDisplay);
	glutKeyboardFunc(DoKeyboard);
	glutMainLoop();
	return 0;
}

void DoKeyboard(unsigned char key, int x, int y)
{
	switch(key) {
	case 'a':yAngle += 2;break;
	case 'd':yAngle -= 2;break;
	case 'w':xAngle += 2;break;
	case 's':xAngle -= 2;break;
	case 'q':zAngle += 2;break;
	case 'e':zAngle -= 2;break;
	case 'z':xAngle = yAngle = zAngle = 0.0;break;

	case 'u':red += 0.1;break;
	case 'j':red -= 0.1;break;
	case 'i':green += 0.1;break;
	case 'k':green -= 0.1;break;
	case 'o':blue += 0.1;break;
	case 'l':blue -= 0.1;break;
```

```
        case 'm':red=0.5, green=0.5, blue=0.5;break;
        }
        char info[128];
        sprintf(info, "(%.0f,%.0f,%.0f)"
                "(%.1f,%.1f,%.1f)",
                xAngle, yAngle, zAngle,
                red, green, blue);
        glutSetWindowTitle(info);
        glutPostRedisplay();
}

void DoDisplay()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glShadeModel(GL_FLAT);
        glEnable(GL_DEPTH_TEST);

        // 조명 모델 설정
        glEnable(GL_LIGHTING);
        GLfloat arLight[]={red, green, blue, 1.0};
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, arLight);

        glPushMatrix();
        glRotatef(xAngle, 1.0f, 0.0f, 0.0f);
        glRotatef(yAngle, 0.0f, 1.0f, 0.0f);
        glRotatef(zAngle, 0.0f, 0.0f, 1.0f);

        glutSolidTeapot(0.5);

        glPopMatrix();
        glFlush();
}
```

조명의 밝기는 일단 디폴트로 맞추었다. 최초 실행해 보면 주전자가 보일듯 말듯 희미하게 보인다. u,i,o키를 눌러 조명을 점점 더 밝게 해 보면 주전자가 서서히 보일 것이다. u키와 i키를 8번씩 눌러 빨간색과 초록색 강도를 높여 보면 주전자가 흐릿한 노란색이 된다. jkl 키는 조명을 점점 어둡게 만들고 m키는 디폴트로 리셋한다.



나머지 설정을 테스트해 보려면 복잡한 상황이 설정되어야 하므로 간단하게 테스트해 보기 어렵다. 다음에 관련 부분에서 알아 보도록 하자.

9-3. 재질

물체의 최종 색상을 결정하는데는 조명보다 물체 자체의 성질이 더 중요하다. 조명은 단지 빛일 뿐이며 이 빛이 물체에 반사되어 망막으로 들어온 빛이 실제 색상으로 인식되기 때문이다. 똑같은 빛이라도 종이에 비춘

것과 유리에 비춘 것, 금속에 비춘 것 등의 효과가 다르게 나타난다. 각 조명의 어떤 색상 요소를 얼마나 반사할 것인가를 지정하는 성질을 물체의 재질이라고 한다. 다음 함수로 각 다각형에 대해 재질을 지정한다.

void glMaterial[f, i][v](GLenum face, GLenum pname, GLfloat* param);

face는 물체의 어떤 면에 대한 재질인지를 지정하는데 GL_FRONT, GL_BACK, GL_FRONT_AND_BACK 중 하나의 값을 전달한다. 앞뒷면에 대해 서로 다른 재질을 부여할 수도 있다. pname은 설정하고자 하는 속성이며 param은 속성의 값이다.

속성	설명
GL_AMBIENT	주변광의 어느 색상 요소를 얼마만큼 반사할 것인가를 지정한다. 각 색상 요소에 대해 -1 ~ 1 사이의 범위를 가지며 디폴트는 (0.2, 0.2, 0.2, 1.0)이다.
GL_DIFFUSE	분산광에 대한 반사도를 지정한다. 디폴트는 (0.8, 0.8, 0.8, 1.0)이다.
GL_AMBIENT_AND_DIFFUSE	위 두 속성을 한꺼번에 조정한다.
GL_SPECULAR	반사광에 대한 반사도를 지정한다. 디폴트는 (0,0,0,1)이다.
GL_EMISSION	방출 속성을 지정한다. 디폴트는 (0,0,0,1)이다.
GL_SHININESS	반사광에 대해 반짝거리는 정도를 지정한다. 0 ~ 128 사이의 범위를 지정할 수 있으며 디폴트는 0이다.
GL_COLOR_INDEXES	주변광, 분산광, 반사광에 대한 색상 인덱스이다. 디폴트는 (0,1,1)이다.

각 종류의 조명에 대해 또 각각의 색상 요소에 대한 반사도를 개별적으로 지정할 수 있다. 물체에 비치는 조명은 물론이고 3가지 조명에 대해 rgba 요소에 대한 반사도와 방출, 반짝임 등의 물체 재질까지 고려되므로 조명에 의한 색상을 결정하는데 20개 이상의 변수가 개입하는 셈이다. 다음 예제는 피라미드의 각 면에 대해 주변광에 대한 반사도를 지정했다.

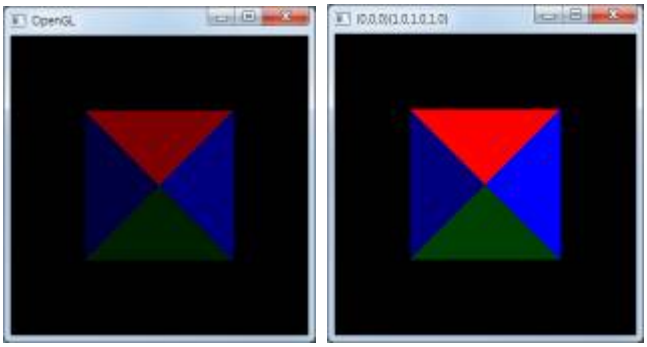
Material

```
#nclude <windows.h>
#nclude <gl/glut.h>
#nclude <stdio.h>

void DoDisplay();
void DoKeyboard(unsigned char key, int x, int y);
void DoMenu(int value);
GLfloat xAngle, yAngle, zAngle;
GLfloat red=0.5, green=0.5, blue=0.5;
GLboolean bColorTrack = GL_FALSE;

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
      glutCreateWindow("OpenGL");
      glutDisplayFunc(DoDisplay);
      glutKeyboardFunc(DoKeyboard);
      glutCreateMenu(DoMenu);
      glutAddMenuEntry("Color Tracking ON",1);
      glutAddMenuEntry("Color Tracking OFF",2);
      glutAttachMenu(GLUT_RIGHT_BUTTON);
      glutMainLoop();
      return 0;
```

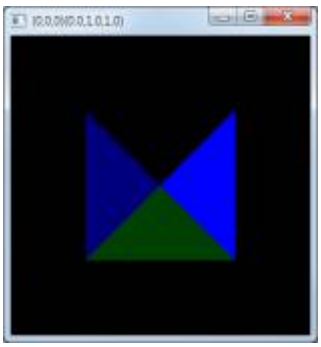
```
}

void DoKeyboard(unsigned char key, int x, int y)
{
	switch(key) {
	case 'a':yAngle += 2;break;
	case 'd':yAngle -= 2;break;
	case 'w':xAngle += 2;break;
	case 's':xAngle -= 2;break;
	case 'q':zAngle += 2;break;
	case 'e':zAngle -= 2;break;
	case 'z':xAngle = yAngle = zAngle = 0.0;break;

	case 'u':red += 0.1;break;
	case 'j':red -= 0.1;break;
	case 'i':green += 0.1;break;
	case 'k':green -= 0.1;break;
	case 'o':blue += 0.1;break;
	case 'l':blue -= 0.1;break;
	case 'm':red=0.5, green=0.5, blue=0.5;break;
	}
	char info[128];
	sprintf(info, "(%.0f,%.0f,%.0f)"
		"(%.1f,%.1f,%.1f)",
		xAngle, yAngle, zAngle,
		red, green, blue);
	glutSetWindowTitle(info);
	glutPostRedisplay();
}

void DoMenu(int value)
{
	switch(value) {
	case 1:
		bColorTrack = GL_TRUE;
		break;
	case 2:
		bColorTrack = GL_FALSE;
		break;
	}
	glutPostRedisplay();
}

void DoDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glShadeModel(GL_FLAT);
	glEnable(GL_DEPTH_TEST);

	// 주변광 설정
	glEnable(GL_LIGHTING);
	GLfloat arLight[]={red, green, blue, 1.0};
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT, arLight);

	if (bColorTrack) {
		glEnable(GL_COLOR_MATERIAL);
		glColorMaterial(GL_FRONT,GL_AMBIENT);
	} else {
		glDisable(GL_COLOR_MATERIAL);
	}
```

```
        glPushMatrix();
        glRotatef(xAngle, 1.0f, 0.0f, 0.0f);
        glRotatef(yAngle, 0.0f, 1.0f, 0.0f);
        glRotatef(zAngle, 0.0f, 0.0f, 1.0f);

        // 아랫면 흰 바닥
        GLfloat arbottom[]={1.0, 1.0, 1.0, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, arbottom);
        glColor3f(1,1,1);
        glBegin(GL_QUADS);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
        glVertex2f(-0.5, -0.5);
        glEnd();

        // 위쪽 빨간 변
        GLfloat arMat1[]={1.0, 0.0, 0.0, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, arMat1);
        glBegin(GL_TRIANGLE_FAN);
        glColor3f(1,1,1);
        glVertex3f(0.0, 0.0, -0.8);
        glColor3f(1,0,0);
        glVertex2f(0.5, 0.5);
        glVertex2f(-0.5, 0.5);

        // 왼쪽 노란 변
        GLfloat arMat2[]={0.0, 0.0, 0.5, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, arMat2);
        glColor3f(1,1,0);
        glVertex2f(-0.5, -0.5);

        // 아래쪽 초록 변
        GLfloat arMat3[]={0.0, 0.25, 0.0, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, arMat3);
        glColor3f(0,1,0);
        glVertex2f(0.5, -0.5);

        // 오른쪽 파란 변
        GLfloat arMat4[]={0.0, 0.0, 1.0, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, arMat4);
        glColor3f(0,0,1);
        glVertex2f(0.5, 0.5);
        glEnd();

        glPopMatrix();
        glFlush();
    }
```

전역 주변광은 일단 회색으로 지정하되 ujikol 키로 각 색상 요소를 증감하도록 했다. 피라미드의 4면에는 각각 다른 재질 속성을 부여했다. 최초 실행하면 주변광이 50%밖에 안되므로 흐릿한 색상으로 보이지만 전역 주변광의 광도를 높이면 점점 밝아진다.



위쪽 면은 빨간색만 100% 반사하고 오른쪽면은 파란색만 100% 반사하도록 설정했다. 그래서 빨간색과 파란색으로 보이되 주변광의 광도에 따라 밝기가 달라진다. 주변광의 R 요소가 1.00이면 완전한 빨간색으로 보이고 0.5이면 밝기가 떨어짐으로 절반만큼만 빨간색으로 보일 것이다. u, j키로 빨간색의 강도를 조정해 보자. R 요소가 0이 되면 반사할 빛이 전혀 없으므로 검정색으로 보인다.



만약 조명에 의해 특성 색상 요소가 1.0을 초과하면 이때는 내림 처리된다. 주변광의 R 요소를 1.0보다 더 크게 지정해도 빨간색이 그 이상 밝아지지는 않는다. 주변광의 B 요소와 파란색 면의 관계도 동일하다. o, l 키로 B 요소를 증감시켜 보아라.

위쪽, 오른쪽 두 면은 빨간색과 파란색만 반사하므로 주변광의 다른 색상 요소에 대해서는 전혀 영향을 받지 않는다. i, k 키로 주변광의 초록색 요소를 증감시켜 봐도 빨간면과 파란면의 색상은 전혀 변하지 않는다. 왜냐하면 이 면의 재질은 초록색 조명을 완전히 흡수하도록 설정되어 있기 때문이다.

색상 요소에 대한 반사도는 명도에 영향을 미친다. 왼쪽면은 파란색을 반사하되 50%만 반사하도록 되어 있어 그래서 오른쪽 면과 색상은 같되 훨씬 더 어둡게 보인다. 똑같은 조명을 비춰도 반사하는 정도가 다르기 때문이다. 아래쪽면은 초록색을 25%반 반사하므로 다른 면보다 훨씬 더 어두운 색을 띈다.

물체의 재질은 각 조명에 대해 또 각 색상 요소에 대해 반사도를 개별적으로 지정할 수 있어 자유도는 높지만 잔손이 굉장히 많이 간다. 또한 반사도로 원하는 색상을 만들어내기도 쉽지 않고 재질 설정 함수가 배열 버전밖에 없어 각 조명에 대한 재질을 지정할 때마다 배열을 하나씩 선언하는 것도 무척이나 귀찮은 일이다.

재질은 조명에 대한 반사도로 다각형의 색상을 지정하는 것이므로 glColor로 지정한 고유한 색상을 완전히 무시한다. 왼쪽면은 노란색으로 지정되었지만 재질에서 파란색만 반사하도록 설정했으므로 파란색으로 보인다. glColor로 지정한 색상을 재질 대신 사용하려면 색상 트래킹(color tracking) 기능을 사용한다. 기능을 사용하려면 다음 두 함수를 호출한다.

```
glEnable(GL_COLOR_MATERIAL);
void glColorMaterial(GLenum face, GLenum mode);
```

이 기능의 이름이 의미하듯이 색상을 곧 재질로 사용하는 것이다. glColorMaterial 함수는 어떤 면에 어떤 조명을 적용할 것인가를 지정한다. face는 조명이 적용될 면이고 mode는 면에 적용할 조명의 종류이다.

팝업 메뉴에서 색상 트래킹 기능을 켜 보자. 이 옵션을 켜면 물체의 앞면에 대해 주변광을 적용한다. 각 면에 대해 재질을 밝힐 필요가 없으며 대신 glColor 함수로 색상을 밝혀야 한다. 각 면의 색상에 따라 재질이 결정되며 주변광의 밝기만 조정하면 전체적인 명도를 쉽게 조정할 수 있다.

9-4. 법선

방향성이 있는 조명은 각도에 따라 물체에 닿는 빛의 강도가 달라진다. 후레쉬로 물체를 비출 때 수직으로 비추는 것과 비스듬하게 비출 때의 색상이 다르게 보이는 것과 같은 이치이다. 그래서 조명에 의해 반사되는 빛의 양을 정확하게 계산하기 위해서는 조명과 물체면과의 각도를 알아야 한다.

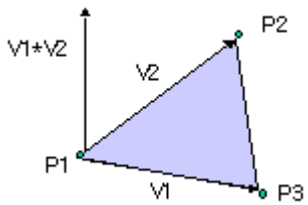
이 각도를 계산하는데 중요한 문제가 있다. 다각형을 구성하는 정점은 3차원 공간상의 위치만 가질 뿐 평면도 아니고 직선도 아니다. 빛은 직진하는 성질이 있으므로 직선이지만 정점은 그냥 위치일 뿐이다. 직선과 점 사이의 각도를 구하는 것은 수학적으로 불가능하다. 각도라는 개념은 최소한 직선과 직선이 만나야만 측정 가능한 것이다. 그래서 정점의 좌표만으로는 조명의 강도를 결정할 수 없으며 추가 정보가 더 필요하다.

정점의 조명 계산을 위한 이 추가 정보가 바로 법선 벡터(Normal Vector)이다. 법선 벡터는 정점에서 수직으로 뻗어나가는 가상의 선이며 정점의 방향을 나타난다. 대개의 경우 정점이 속한 평면에 대해 위쪽으로 수직선을 그으면 되지만 문제가 그리 간단하지 않다. 곡면을 구성하는 다각형에 속한 정점들은 법선의 각도가 다 달라야 부드러운 곡면을 표현할 수 있다. 그래서 OpenGL은 법선에 대한 자동 계산을 지원하지 않으며 전적으로 개발자가 지정해야 한다. 법선은 다음 함수로 배치한다.

void glNormal3[f,d,i,s,b][v](GLfloat nx, GLfloat ny, GLfloat nz);

정점 하나에 대해 법선을 지정하므로 이 함수는 주로 glVertex 함수와 쌍으로 호출된다. 여러 개의 정점이 하나의 법선을 공유한다면 glNormal을 한번만 호출하고 glVertex로 정점을 계속 배치하면 된다. 즉, 법선도 한번 배치하면 다른 값으로 바꾸기전에는 계속 유효하다.

다음은 삼각형을 구성하는 정점의 법선을 구하는 공식이다. 세 개의 정점으로 구성된 삼각형의 경우 한 점에서 나머지 두 점으로 두 개의 벡터를 그릴 수 있다. 이 두 벡터의 외적(Cross Product)을 구하면 두 벡터에 수직인 벡터가 구해진다.



V1 벡터가 (x1, y1, z1), V2 벡터가 (x2, y2, z2)일 때 두 벡터의 외적은 다음 공식으로 구할 수 있다. 이 공식은 수학책에 나와 있으므로 나한테 묻지 말고 홍성대 선생님께 문의하기 바란다. 또 벡터의 외적은 어째서 항상 두 벡터에 수직인지는 교양 물리학을 참고하도록 하자.

(y1z2 - z2y1, z1x2-x1z2, x1y2-y1x1)

OpenGL은 광원의 각도 계산을 단순화하기 위해 법선 벡터의 길이를 1로 지정할 것을 요구한다. 어차피 법선은 각도 계산을 위해 지정하는 것이고 각도 계산에 길이는 무의미하다. 길이가 1인 법선 벡터를 단위 법선 벡터라고 하며 길이를 1로 줄이는 것을 정규화라고 한다. 길이를 1로 줄이려면 먼저 벡터의 길이를 구해야 한다. 벡터의 길이는 피타고라스의 정리로 쉽게 구할 수 있다.

$$\sqrt{x^2+y^2+z^2}$$

법선 벡터의 좌표를 길이로 나누어 (x/len, y/len, z/len)을 취하면 단위 법선 벡터가 된다. 모든 법선을 일일이 정규화하는 것이 귀찮다면 GL_NORMALIZE 기능을 켜 OpenGL이 법선을 정규화하도록 할 수도 있다. 자동 정규화 기능을 사용하면 편리하지만 성능이 저하되는 단점이 있다. 다음 예제는 피라미드의 각 정점에 대해 이 공식대로 법선을 계산하여 지정한다.

Normal

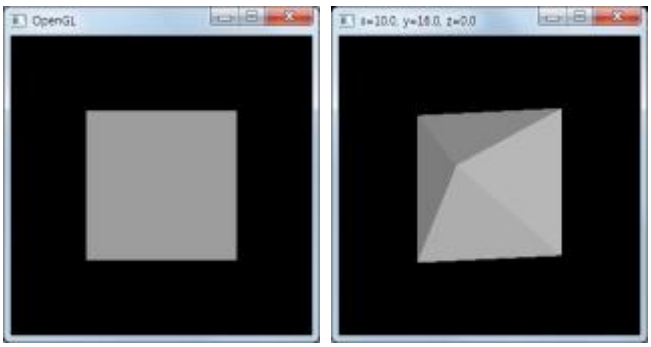
```
#include <windows.h>
#include <gl/glut.h>
#include <stdio.h>
#include <math.h>

void DoDisplay();
void DoKeyboard(unsigned char key, int x, int y);
void DoMenu(int value);

GLfloat xAngle, yAngle, zAngle;
GLboolean bNormal = GL_TRUE;

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
	glutCreateWindow("OpenGL");
	glutDisplayFunc(DoDisplay);
	glutKeyboardFunc(DoKeyboard);
	glutCreateMenu(DoMenu);
	glutAddMenuEntry("Normal ON",1);
	glutAddMenuEntry("Normal OFF",2);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutMainLoop();
	return 0;
}

void DoKeyboard(unsigned char key, int x, int y)
{
	switch(key) {
	case 'a':yAngle += 2;break;
	case 'd':yAngle -= 2;break;
	case 'w':xAngle += 2;break;
	case 's':xAngle -= 2;break;
	case 'q':zAngle += 2;break;
	case 'e':zAngle -= 2;break;
	case 'z':xAngle = yAngle = zAngle = 0.0;break;
	}
	char info[128];
	sprintf(info, "x=%.1f, y=%.1f, z=%.1f", xAngle, yAngle, zAngle);
	glutSetWindowTitle(info);
	glutPostRedisplay();
}

void DoMenu(int value)
{
	switch(value) {
	case 1:
		bNormal = GL_TRUE;
		break;
```

```
		case 2:
			bNormal = GL_FALSE;
			break;
		}
		glutPostRedisplay();
}

void GetNormal(GLfloat a[3], GLfloat b[3], GLfloat c[3], GLfloat normal[3])
{
		GLfloat ba[3];
		GLfloat ca[3];
		GLfloat n[3];

		// 두 정점간의 벡터 계산
		ba[0]=b[0]-a[0];ba[1]=b[1]-a[1];ba[2]=b[2]-a[2];
		ca[0]=c[0]-a[0];ca[1]=c[1]-a[1];ca[2]=c[2]-a[2];

		// 외적 구함
		n[0]=ba[1]*ca[2]-ca[1]*ba[2];
		n[1]=ca[0]*ba[2]-ba[0]*ca[2];
		n[2]=ba[0]*ca[1]-ca[0]*ba[1];

		// 정규화
		GLfloat l=sqrt(n[0]*n[0] + n[1]*n[1] + n[2]*n[2]);
		normal[0]=n[0]/l;normal[1]=n[1]/l;normal[2]=n[2]/l;
}

void DoDisplay()
{
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		glShadeModel(GL_FLAT);
		glEnable(GL_DEPTH_TEST);

		// 조명을 켠다.
	    glEnable(GL_LIGHTING);
	    glEnable(GL_LIGHT0);
	    GLfloat ambient[] = { 0.5, 0.5, 0.5, 1.0 };
	    glLightfv(GL_LIGHT0,GL_AMBIENT,ambient);
	    GLfloat diffuse[] = { 0.5, 0.5, 0.5, 1.0 };
	    glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuse);
	    GLfloat spec[] = { 1.0, 1.0, 1.0, 1.0 };
	    glLightfv(GL_LIGHT0,GL_SPECULAR,ambient);

	    glEnable(GL_COLOR_MATERIAL);
	    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

		 glPushMatrix();
		 glRotatef(xAngle, 1.0f, 0.0f, 0.0f);
		 glRotatef(yAngle, 0.0f, 1.0f, 0.0f);
		 glRotatef(zAngle, 0.0f, 0.0f, 1.0f);

		// 아랫면 흰 바닥
		glBegin(GL_QUADS);
		glVertex2f(-0.5, 0.5);
		glVertex2f(0.5, 0.5);
		glVertex2f(0.5, -0.5);
		glVertex2f(-0.5, -0.5);
		glEnd();

		GLfloat normal[3];
```

```
	glColor3ub(128, 128, 128);

	// 위
	glBegin(GL_TRIANGLES);
	GLfloat up[3][3]={
		{0.0, 0.0, -0.8},
		{0.5, 0.5, 0.0,},
		{-0.5, 0.5, 0.0},
	};
	GetNormal(up[0], up[1], up[2], normal);
	if (bNormal) glNormal3fv(normal);
	glVertex3fv(up[0]);
	glVertex3fv(up[1]);
	glVertex3fv(up[2]);

	// 왼쪽
	GLfloat left[3][3]={
		{0.0, 0.0, -0.8},
		{-0.5, 0.5, 0.0},
		{-0.5, -0.5, 0.0},
	};
	GetNormal(left[0], left[1], left[2], normal);
	if (bNormal) glNormal3fv(normal);
	glVertex3fv(left[0]);
	glVertex3fv(left[1]);
	glVertex3fv(left[2]);

	// 아래
	GLfloat down[3][3]={
		{0.0, 0.0, -0.8},
		{-0.5, -0.5, 0.0},
		{0.5, -0.5, 0.0},
	};
	GetNormal(down[0], down[1], down[2], normal);
	if (bNormal) glNormal3fv(normal);
	glVertex3fv(down[0]);
	glVertex3fv(down[1]);
	glVertex3fv(down[2]);

	// 오른쪽
	GLfloat right[3][3]={
		{0.0, 0.0, -0.8},
		{0.5, -0.5, 0.0},
		{0.5, 0.5, 0.0},
	};
	GetNormal(right[0], right[1], right[2], normal);
	if (bNormal) glNormal3fv(normal);
	glVertex3fv(right[0]);
	glVertex3fv(right[1]);
	glVertex3fv(right[2]);

	glEnd();
	glPopMatrix();
	glFlush();
}
```

이 예제의 GetNormal 함수의 코드가 다소 복잡해 보이는데 위에서 소개한 공식대로 두 벡터의 외적을 구하고 길이로 나누어 정규화한다. 이 공식을 사용하려면 각 정점의 좌표를 전달해야 하므로 좌표들을 모두 배

열로 정의한 후 사용했다.



최초 피라미드는 정면을 바라보고 있으며 이 상태에서는 삼각형 4면의 각도가 일치한다. 그러다 보니 조명이 4면에 골고루 비쳐져 면간의 구분이 되지 않는다. 그러나 조금이라도 피라미드를 회전시켜 보면 각 면의 각도가 달라짐으로써 조명의 강도가 달라지고 결과적으로 각 면의 색상이 달라진다.

이런 계산이 가능한 이유는 각 정점에 대해 법선이 정의되어 있기 때문이다. 법선에 의해 조명이 면을 비추는 각도를 알 수 있고 따라서 색상을 결정할 수 있다. 법선이 없으면 어떻게 되는지 팝업 메뉴에서 법선 기능을 꺼 보자. 각 면에 대해 조명을 어떤 강도로 비춰야 할지를 결정할 수 없으므로 모든 면에 균일하게 비춰지며 그러다 보니 면이 전혀 구분되지 않는다.

이 예제는 공식대로 법선을 지정하여 한 다각형에 속한 정점의 법선을 모두 동일하게 처리했다. 좀 더 부드러운 처리를 위해서는 모든 정점마다 최적의 법선을 지정해야 한다. 특히 곡면을 구성하는 다각형은 법선을 제대로 지정해야 다각형의 각이 완화되며 조명이 부드럽게 입혀진다.