5.GLUT

5-1.GLUT

문자 기반의 콘솔창에 화려한 그래픽을 그릴 수는 없으므로 3차원이든 2차원이든 그래픽을 출력하려면 윈도우가 필요하다. 그러나 OpenGL은 윈도우 시스템과의 독립성 확보를 위해 윈도우 관리에 대해서는 어떠한 기능도 제공하지 않는다. 그래픽 코드를 윈도우 시스템과 완전히 분리해야 플랫폼 독립성을 확보할 수 있기 때문이다.

OpenGL의 의도된 빈틈을 메워주는 것이 각 운영체제별로 제공되는 AUX 라이브러리이다. 그러나 AUX는 거의 운영체제 전반의 지식을 요구하므로 처음 학습용으로는 부적합하다. OpenGL 예제를 만들기 위해 X 윈도우 시스템의 Xlib이나 Motif, 매킨토시의 GUI를 먼저 배워야 한다면 정말 맥 빠질 것이다. AUX는 기능은 섬세하지만 운영체제마다 판이하게 틀려 호환성에도 불리하다. Win32 API로 멋진 프로그램을 만들어 봤자 그 프로그램은 윈도우즈에서만 실행될 뿐이다.

이런 단점을 일거에 해소해 주는 것이 바로 GLUT이다. GLUT의 고수준 인터페이스는 기반 운영체제들을 완벽하게 추상화하여 플랫폼 독립성을 제공한다. GLUT이 제공하는 기능은 그래픽과는 거의 상관없는 기능들이며 쉽게 말해서 그래픽을 그리기 위한 껍데기를 만들어 주는 것이다. 그러나 이걸 대충이라도 알아야 창을 만들어 간단한 삼각형 쪼가리라도 뿌려 보고 입력을 받아 출력에 변화를 줄 수 있다.

그래서 OpenGL 학습을 위해 GLUT을 우선적으로 연구해 봐야 하는 것이다. 어디까지나 학습을 위한 도구일 뿐이므로 너무 상세하게 연구할 필요는 없다. 어차피 상용 프로그램을 만들어야 한다면 GLUT으로는 한계가 있어 타겟 플랫폼의 AUX를 사용해야 한다. 여기서는 이후의 강좌 실습을 위해 필요한 부분까지만 학습해 볼 것이다. 주요 함수들에 대해 간략하게 소개만 하고 간단한 예제로 동작만 확인해 본다.

void glutInit(int *argcp, char **argv);

이 함수는 GLUT 라이브러리를 초기화하고 기반 플랫폼의 윈도우 시스템과 연결한다. 인수는 main으로부터 전달받은 argc의 주소와 argv의 배열을 전달하는데 argc가 내부에서 변경될 수도 있으므로 반드시 참조로 전달해야 한다. 명령행 인수는 윈도우 시스템 초기화 방법에 대한 정보를 제공하는데 주로 X 윈도우를 위한 것들이며 다른 시스템에는 해당되지 않는다.

glutInit 함수가 하는 가장 중요한 일은 에러 처리이다. 윈도우 시스템과 연결할 수 없다거나 해당 운영체제가 그래픽 인터페이스를 제공하지 않는다면 에러 메시지를 출력하고 프로그램을 강제 종료하는 극단적인 방어가 주된 임무이다. 우리가 실습하고 있는 윈도우즈 환경은 확실한 그래픽 운영체제이고 명령행 인수를 받지도 않으므로 glutInit을 굳이 호출하지 않아도 상관없다.

GLUT의 가장 중요한 기능은 그래픽을 출력할 수 있는 윈도우를 생성하는 것이다. 윈도우를 생성하기 전에 윈도우의 여러 가지 옵션들을 먼저 설정해야 한다. 다음 두 함수는 윈도우의 위치나 크기를 지정한다. 윈도우를 생성하기 전에 미리 원하는 크기와 위치를 전달해야 한다.

void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);

인수로 폭과 높이 그리고 (x, y) 좌표를 전달한다. 윈도우의 좌표는 당연히 화면 좌상단을 원점으로 한다. 윈도우의 크기는 의미가 약간 다른데 창의 크기를 지정하는 것이 아니라 작업영역의 크기를 지정함을 주의하자. 창의 크기는 작업영역 크기에 타이틀 바와 경계선이 더해지므로 지정한 크기보다 조금 더 크게 생성된다.

예를 들어 크기를 100*100으로 지정했다면 윈도우의 크기가 100*100인 것이 아니라 윈도우 안쪽의 그래픽 출력 영역이 100*100이라는 뜻이다. 당연히 윈도우 크기는 이보다 조금 더 크다.

왜 이렇게 되어 있을까는 조금만 생각해 보면 짐작할 수 있는데 윈도우 시스템마다 타이틀 바 두께가 제각각이기 때문이다. 중요한 것은 그래픽이 출력될 작업 영역의 크기이지 윈도우 자체의 크기가 아니다. 플랫폼에 따른 이런 차이점들을 잘 추상화해주는 것이 GLUT의 주 임무이다. 크기와 위치를 지정한다고 해서 운영체제가 반드시 이대로 윈도우를 생성한다고 보장할 수는 없다. GLUT은 단순히 의도를 밝힐 뿐이며 이 부탁을 들어줄 것인가는 순전히 운영체제 마음이다.

위치와 크기를 생략하면 화면 좌상단에 300*300 정도의 적당한 크기로 윈도우를 생성한다. 이 디폴트 크기는 예제 작성에 아주 적합하므로 이 강좌에서는 이 두 함수를 굳이 호출하지 않는다. 다음 함수는 디스플레이 모드를 설정한다.

void glutInitDisplayMode(unsigned int mode);

디스플레이 모드는 그리기 표면의 주요 특징들을 결정한다. 윈도우를 생성한 후에는 변경할 수 없으므로 처음에 잘 결정해야 한다. 여러 개의 모드를 OR 연산자로 묶어서 지정하되 상호 배타적인 속성의 플래그들은 하나만 지정해야 한다. 예를 들어 싱글 버퍼와 더블 버퍼는 둘 중 하나를 선택하는 것이므로 두 플래그를 같이 쓸 수는 없다.

모드	설명
GLUT_RGBA, GLUT_RGB	트루 컬러 모드이다. 이 값이 디폴트이다.
GLUT_INDEX	팔레트를 사용하는 인덱스 모드이다. 몇 가지 기능에 한계가 있어 거의 사용되지 않는다.
GLUT_SINGLE	싱글 버퍼를 사용한다. 이 값이 디폴트이다.
GLUT_DOUBLE	더블 버퍼를 사용한다.
GLUT_DEPTH	깊이 버퍼를 사용한다.
GLUT_ACCUM	누적 버퍼를 사용한다.
GLUT_ALPHA	색상에 알파 요소를 사용한다.
GLUT_STENCIL	스텐실 버퍼를 사용한다.
GLUT_MULTISAMPLE	멀티 샘플링을 지원한다. 이 기능을 사용하려면 클라이언트와 서버가 모두 지원해야 한다.
GLUT_STEREO	스테레오 윈도우를 생성한다.
GLUT_LUMINANCE	루미넌스(luminance) 색상 모델을 사용한다.

디스플레이 모드의 디폴트는 트루 컬러 모드에 싱글 버퍼를 사용하는 것이다. 그리고 깊어 버퍼는 디폴트로 사용하는 것으로 되어 있다. 즉, 별다른 지정이 없으면 다음과 같이 초기화하는 것과 동일하다. 디폴트대로 사용하려면 이 함수를 굳이 호출하지 않아도 상관없다.

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

물론 더블 버퍼링이 필요하면 원하는 모드로 변경해야 한다. 윈도우의 위치와 크기, 디스플레이 모드까지 결정했으면 다음 함수를 호출하여 윈도우를 생성한다. 그래픽을 출력하려면 필수적으로 호출해야 하는 함수이다.

int glutCreateWindow(char *name);

윈도우는 상단에 제목을 가지는데 제목 문자열을 인수로 지정한다. 단순한 문자열이므로 원하는대로 지정하면 된다. 윈도우를 지칭하는 유일한 ID가 리턴되는데 이 ID는 차후 윈도우를 관리할 때 사용된다. 윈도우는 탑 레벨로 생성되며 곧바로 보이고 활성 상태가 된다. 그러나 화면에 실제로 보이려면 그리기 메시지를 최소한 한번은 처리해야 한다. 다음 함수는 메시지 루프를 실행한다.

void glutMainLoop(void);

모든 윈도우 시스템은 이벤트 드리븐 방식으로 동작한다. 이 함수는 계속 실행되면서 사용자나 시스템에 의해 발생한 메시지를 받아 메시지 처리 함수를 호출하는 중요한 역할을 한다. 메시지를 처리하는 콜백 함수는 메시지 루프로 들어가기 전에 다음 함수들로 미리 등록해 두어야 한다.

void glutDisplayFunc(void (*func)(void));
void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
void glutReshapeFunc(void (*func)(int width, int height));

화면을 그릴 때, 키보드 입력을 받았을 때, 마우스 입력을 받았을 때, 윈도우 크기가 변경되었을 때 호출할 함수를 등록하는 것이다. 각 콜백의 원형은 등록 함수의 인수 목록에 나타나 있으므로 이 원형대로 함수를 생성하고 함수명을 등록함수의 인수로 전달하면 된다. glutMainLoop는 끊임없이 메시지 큐를 감시하며 메시지가 들어올 때마다 대응되는 콜백 함수를 계속 호출한다. 이미 등록한 콜백을 해제할 때는 인수로 NULL을 전달한다.

이 외에도 특수키, 마우스 이동, 팝업 메뉴 선택 등의 이벤트를 처리하는 여러 가지 콜백들이 있다. 관심있는 이벤트에 대해 콜백을 등록하되 OpenGL의 주 목적은 그래픽을 그리는 것이므로 그리기 콜백은 선택의 여지없이 반드시 등록해야 한다. 다음 함수는 윈도우의 캡션 문자열을 변경한다.

void glutSetWindowTitle(char *name);

캡션은 윈도우의 제목을 보여주는 역할을 하지만 실행중에 변수값을 찍어 보고 싶을 때도 아주 유용하다. OpenGL은 폰트 출력이 번거로와 캡션 외에는 정보를 실시간으로 출력해 볼만한 장치가 없다. 앞으로 예제에서 변수값 확인을 위해 이 함수를 종종 사용할 것이다. 지금까지의 예제에서는 WinMain에서 다음 세 줄의 코드만 호출한다.

```
glutCreateWindow("OpenGL");
glutDisplayFunc(DoDisplay);
glutMainLoop();
```

윈도우 만들고 출력 콜백을 등록한 후 메시지 루프를 돌리기만 한다. 명령행 인수를 해석할 필요도 없고 모드를 바꿀 필요도 없고 키보드 입력도 받지 않으므로 아주 간단하다. 앞으로의 예제는 조금 더 복잡해질 것이다. 위치나 크기에 대한 지정이 없으므로 화면 좌상단에 300 * 300 크기로 작게 열린다. 다음과 같이 수정하면 윈도우가 좀 더 커진다.

```
int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
        glutInitWindowSize(800, 600);
        glutInitWindowPosition(100,100);
        glutCreateWindow("OpenGL");
```

```
    glutDisplayFunc(DoDisplay);
    glutMainLoop();
    return 0;
}
```

사용자가 실행중에 창의 크기를 조정할 수 있고 위치도 마음대로 옮길 수 있으므로 초기 크기나 위치는 사실 큰 의미가 없다.

5-2. 입력 처리

그래픽 프로그램의 주 기능은 출력이므로 입력 기능은 사실 크게 필요치 않으며 그래서 OpenGL 자체는 입력 기능이 아예 없다. 하지만 사용자의 지시를 받아 그래픽 상태를 변경한다거나 변수값을 실행중에 바꿔 변화를 관찰해 보려면 입력 기능이 필수적이다. 매번 변수값을 바꿔 재 컴파일해 보기는 너무 번거롭다. 다행히 GLUT은 기본적인 입력 기능을 제공한다. 다음 함수들로 콜백을 지정하면 입력시 해당 함수가 호출된다.

```
void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
void glutSpecialFunc(void (*func)(int key, int x, int y));
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```

각 콜백이 등록하는 입력 함수의 원형은 인수 목록의 함수 포인터 타입에 명시되어 있다. 예를 들어 키보드 입력을 받는 함수는 다음 원형대로 작성해야 한다. 물론 함수의 이름은 명칭일 뿐이므로 사용자가 마음대로 정할 수 있다. 이 강좌에서는 Do + 이벤트명 형식을 일관되게 사용하고 있다.

```
void DoKeyboard(unsigned char key, int x, int y)
{
}
```

key 인수는 입력된 문자의 아스키 코드값이며 조합키까지 적용된 것이다. A 키를 Shift와 함께 눌렀으면 A가 전달되며 그냥 눌렀으면 a가 전달된다. Ctrl, Alt, Shift 조합키의 상태를 따로 조사하려면 glutGetModifiers 함수를 호출한다. x, y 인수는 키 입력시의 마우스 커서 위치이되 큰 실용성은 없다.

Keyboard 함수는 문자키만 입력받을 뿐 커서 이동키나 펑션키같은 특수키는 입력받을 수 없다. 특수키는 Special 함수로 입력받는다. 이 함수는 특수키만 입력받을 뿐 일반 문자키는 입력받지 못한다. 키의 종류에 따라 입력받는 콜백이 완전히 분리되어 있는 셈이다. key 인수로 눌러진 키의 코드가 전달되는데 각 키에 대해 다음과 같은 상수들이 정의되어 있다.

```
GLUT_KEY_F1~
GLUT_KEY_LEFT
GLUT_KEY_RIGHT
GLUT_KEY_UP
GLUT_KEY_DOWN
GLUT_KEY_PAGE_UP
GLUT_KEY_PAGE_DOWN
GLUT_KEY_HOME
GLUT_KEY_END
GLUT_KEY_INSERT
```

Mouse 함수는 마우스 버튼 입력을 받는다. button은 눌러진 마우스 버튼의 이름이며 GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON 셋 중 하나이다. 마우스 버튼이 하나밖에 없는 시스템에서는 왼쪽 버튼만 눌러진다. state는 버튼의 상태이며 GLUT_DOWN, GLUT_UP 둘 중 하나이다. x, y 인수는 마우스 버튼이 눌러진 윈도우상의 좌표이다. 이 함수는 마우스 버튼의 누름과 놓음만 인식할 뿐이며 이동시에는 호출되지 않는다. 마우스 이동시의 이벤트 콜백은 다음 두 함수로 지정한다.

void glutMotionFunc(void (*func)(int x, int y));
void glutPassiveMotionFunc(void (*func)(int x, int y));

glutMotionFunc의 콜백 함수는 마우스 버튼을 누른 채로 윈도우 내부에서 움직일 때에 호출되며 glutPassiveMotionFunc은 버튼을 누르지 않고 움직일 때 호출되는 콜백 함수를 지정한다. 드래그 처리를 하려면 이 두 이벤트를 처리해야 한다.

어느 윈도우 시스템에나 키보드와 마우스 입력을 처리하는 방식은 사실 비슷하므로 조금이라도 경험이 있다면 이 함수들은 쉽게 사용할 수 있을 것이다. 다음 예제는 키보드와 마우스 입력을 받아 도형의 색상이나 위치를 변경한다. 입력에 대해 그래픽 출력을 약간씩 바꿔 보는 전형적인 예제이다.

glutInput

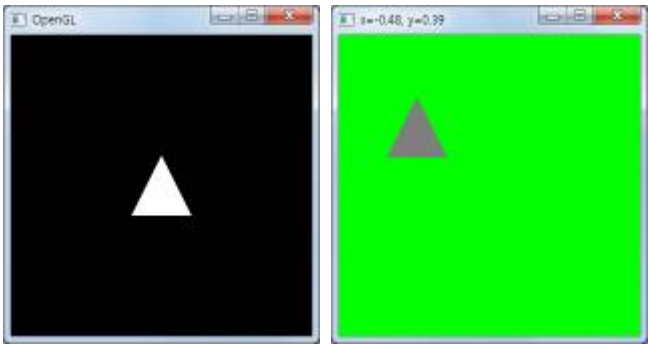
```
#include <windows.h>
#include <gl/glut.h>
#include <stdio.h>

void DoDisplay();
void DoKeyboard(unsigned char key, int x, int y);
void DoSpecial(int key, int x, int y);
void DoMouse(int button, int state, int x, int y);

const GLfloat size = 0.2;
const GLfloat step = 0.01;
GLfloat nx, ny;
GLboolean bGray = GL_FALSE;

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

void DoKeyboard(unsigned char key, int x, int y)
{
	switch(key) {
	case 'r':
	case 'R':
		glClearColor(1.0, 0.0, 0.0, 1.0);
		break;
	case 'g':
	case 'G':
		glClearColor(0.0, 1.0, 0.0, 1.0);
		break;
	case 'b':
	case 'B':
		glClearColor(0.0, 0.0, 1.0, 1.0);
		break;
	}
	glutPostRedisplay();
}
```

```
void DoSpecial(int key, int x, int y)
{
	switch(key) {
	case GLUT_KEY_LEFT:
		nx -= step;
		break;
	case GLUT_KEY_RIGHT:
		nx += step;
		break;
	case GLUT_KEY_UP:
		ny += step;
		break;
	case GLUT_KEY_DOWN:
		ny -= step;
		break;
	}
	char info[128];
	sprintf(info, "x=%.2f, y=%.2f", nx, ny);
	glutSetWindowTitle(info);
	glutPostRedisplay();
}

void DoMouse(int button, int state, int x, int y)
{
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
		bGray = !bGray;
		glutPostRedisplay();
	}
}

void DoDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT);

	if (bGray) {
		glColor3f(0.5, 0.5, 0.5);
	} else {
		glColor3f(1.0, 1.0, 1.0);
	}
	glBegin(GL_POLYGON);
	glVertex2f(nx, ny + size);
	glVertex2f(nx - size, ny - size);
	glVertex2f(nx + size, ny - size);
	glEnd();
	glFlush();
}
```

검정 바탕에 흰색 삼각형을 화면 중앙에 그려 놓았다. 위치는 nx, ny 두 변수로 지정하고 삼각형의 색상은 bGray 변수로 통제한다. 키보드 이벤트에서 이 변수들의 값을 조작함으로써 그래픽을 바꿔 본다. 커서 이동키를 누르면 nx, ny 값을 증감시키고 마우스 왼쪽 버튼을 누르면 bGray 변수를 토글한다. 또 r, g, b 문자를 입력하면 배경색상도 바꾸어 본다. 그리기에 관련된 값을 변경한 후 다음 함수로 다시 그린다.

void glutPostRedisplay(void);

이 함수는 윈도우가 다시 그려져야 함을 표시하기만 할 뿐 즉시 그리기를 하지는 않는다. 다시 그릴 필요가 있음(전문 용어로 무효하다고 표현한다)을 표시만 해 놓으면 메시지 루프가 적당한 때에 Display 콜백을 호출하도록 되어 있다. 실행한 후 키보드나 마우스 등으로 조작해 보자.



특별히 어려운 코드는 없고 지극히 평이하다. 콜백 함수를 원형에 맞게 작성해 놓고 WinMain에서 콜백 함수를 등록해 놓았다. 키보드 입력 함수는 대문자와 소문자를 case 문에 나란히 적음으로써 둘 다 인정한다. 커서 이동키로 도형의 위치를 이동시킬 때마다 현재 좌표를 문자열로 조립하여 타이틀 바에 보여준다.

앞으로의 예제들도 이와 비슷한 방식으로 키보드나 마우스 입력을 받아 그리기에 사용되는 값들을 실행중에 바꿔볼 것이다. 3D 그래픽은 각도를 움직여 봐야 모양을 제대로 확인할 수 있으며 재컴파일하지 않고도 실행중에 함수의 인수들을 요모 조모 바꿔 봄으로써 이해력을 높일 수 있기 때문이다.

5-3.팝업 메뉴

키보드나 마우스는 손가락으로 콕콕 눌러 입력할 수 있으므로 조작이 쉽다는 장점이 있는 반면 어떤 키에 어떤 기능이 할당되어 있는지 알 수 없어 직관적이지 못하다. 반면 팝업 메뉴는 기능의 목록을 문자열 형태로 나열할 수 있으므로 사용법을 외우지 않아도 쉽게 이해할 수 있다는 이점이 있다. 그래서 이 강좌는 팝업 메뉴를 적극적으로 활용한다.

앞서 작성했던 Primitive 등의 예제에서 이미 팝업 메뉴를 사용한 적이 있다. GLUT은 이런 테스트 목적으로 팝업 메뉴 기능을 제공한다. 어느 윈도우 시스템이나 팝업 메뉴는 제공하므로 명령이나 옵션을 나열하기에는 최적이다. 팝업 메뉴는 다음 함수로 생성한다.

```
int glutCreateMenu(void (*func)(int value));
```

인수로 메뉴 항목을 선택했을 때 호출될 콜백 함수를 지정한다. 콜백 함수는 선택된 메뉴 항목의 ID를 인수로 전달받는다. 콜백은 메뉴 항목의 ID에 따라 대응되는 처리를 할 것이다. 이 함수는 메뉴 생성 후 메뉴의 ID를 리턴하는데 이 ID를 다음 함수로 전달함으로써 현재 메뉴로 지정한다.

```
void glutSetMenu(int menu);
int glutGetMenu(void);
```

새로 생성된 메뉴는 자동으로 현재 메뉴가 되므로 굳이 glutSetMenu를 호출하지 않아도 상관없다. 둘 이상의 메뉴를 생성해 놓고 번갈아 사용하고 싶을 때 glutSetMenu가 필요하다. 다음 함수는 메뉴를 파괴한다.

```
void glutDestroyMenu(int menu);
```

메뉴는 껍데기일 뿐이므로 이 안에 실제 명령에 해당하는 내용물을 채워야 한다. 메뉴 내에 메뉴 항목이나 서브 메뉴는 다음 함수로 추가한다. 캡션 문자열과 항목의 ID를 지정한다.

void glutAddMenuEntry(char *name, int value);
void glutAddSubMenu(char *name, int menu);

다음 함수들은 메뉴를 마우스 버튼에 부착하거나 뗀다. 어떤 동작에 대해 메뉴를 호출할 것인지를 지정하는데 GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON 중 하나의 값을 지정하되 메뉴는 통상 마우스 오른쪽 버튼에 부착한다.

void glutAttachMenu(int button);
void glutDetachMenu(int button);

다음 함수는 메뉴 항목을 실행중에 변경한다. 순서값으로 변경 대상 항목을 지정하고 새로운 캡션, 새로운 ID를 지정한다. 순서값은 위쪽부터 순서대로 1이다. 사실상 ID를 바꿀 일은 거의 없고 캡션 정도를 바꾸는 경우가 종종 있다.

void glutChangeToMenuEntry(int entry, char *name, int value);

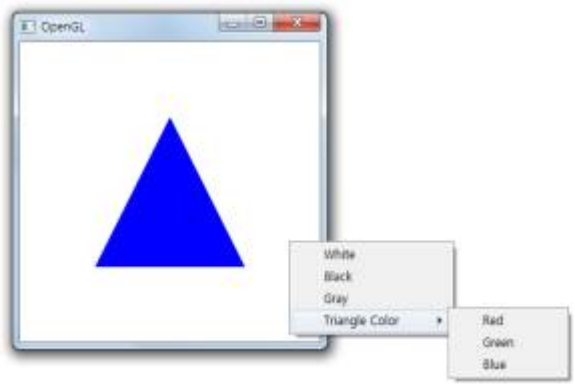
다음 예제는 팝업 메뉴로 도형의 색상과 배경 색상을 변경한다. 그리기에 사용할 변수들을 메뉴로 바꿔 보는 예이다.

glutMenu

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

void DoDisplay();
void DoMenu(int value);

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
        glutCreateWindow("OpenGL");

        // 서브 메뉴 미리 준비
        GLint SubMenu = glutCreateMenu(DoMenu);
        glutAddMenuEntry("Red",4);
        glutAddMenuEntry("Green",5);
        glutAddMenuEntry("Blue",6);

        // 메인 메뉴 생성
        glutCreateMenu(DoMenu);
        glutAddMenuEntry("White",1);
        glutAddMenuEntry("Black",2);
        glutAddMenuEntry("Gray",3);

        // 서브 메뉴를 메인 메뉴에 붙인다.
        glutAddSubMenu("Triangle Color",SubMenu);
        glutAttachMenu(GLUT_RIGHT_BUTTON);

        glutDisplayFunc(DoDisplay);
```

```
	glColor3f(1.0, 0.0, 0.0);
	glutMainLoop();
	return 0;
}

void DoMenu(int value)
{
	switch(value) {
	case 1:
		glClearColor(1.0, 1.0, 1.0, 1.0);
		break;
	case 2:
		glClearColor(0.0, 0.0, 0.0, 1.0);
		break;
	case 3:
		glClearColor(0.5, 0.5, 0.5, 1.0);
		break;
	case 4:
		glColor3f(1.0, 0.0, 0.0);
		break;
	case 5:
		glColor3f(0.0, 1.0, 0.0);
		break;
	case 6:
		glColor3f(0.0, 0.0, 1.0);
		break;
	}
	glutPostRedisplay();
}

void DoDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT);

	glBegin(GL_TRIANGLES);
	glVertex2f(0.0, 0.5);
	glVertex2f(-0.5, -0.5);
	glVertex2f(0.5, -0.5);
	glEnd();
	glFlush();
}
```

이 예제는 서브 메뉴를 포함하고 있다. 서브 메뉴를 미리 준비해 두고 메인 메뉴 생성 후 서브 메뉴를 메인 메뉴의 항목으로 붙여야 한다. 서브 메뉴는 필요한만큼 더 붙일 수 있다. 메뉴를 마우스 오른쪽 버튼에 부착했으므로 오른쪽 버튼을 누르면 팝업 메뉴가 나타난다.



메뉴를 선택했을 때의 동작은 DoMenu 함수에서 처리한다. 메인 메뉴 항목과 서브 메뉴 항목의 콜백 함수를 굳이 따로 만들 필요는 없다. ID만 확실하게 구분된다면 같은 콜백 함수를 공유할 수 있다. DoMenu는 glClearColor나 glColor 함수를 호출하여 선택된 메뉴 항목에 대응되는 색상을 변경한다. 메뉴 항목을 선택하면 배경색이 바뀌며 서브 메뉴 안의 메뉴를 선택하면 도형이 색상이 바뀔 것이다.

GLUT이 제공하는 메뉴는 별도의 리소스를 필요로 하지도 않고 코드로 간단하게 생성 및 처리할 수 있어 아주 편리하다. 그러나 메뉴 항목 사이에 구분선을 넣거나 체크 표시를 토글하는 고급 기능은 제공하지 않으므로 단순한 테스트용으로만 쓸 수 있는 정도이다. 메뉴 항목의 사용 여부를 지정하거나 실행중에 메뉴 항목을 첨삭하는 등의 고급 기능이 필요하다면 각 플랫폼별로 제공되는 메뉴 API를 직접 사용해야 한다.

이 강좌의 예제들은 팝업 메뉴를 아주 즐겨 사용한다. Action 변수로 DoDisplay 함수의 동작을 선택함으로써 한 예제에 여러 가지 그리기 코드를 통합적으로 작성할 수 있어서 좋고 재컴파일없이 실행중에 값을 바꿔볼 수 있다는 면에서 아주 편리하다. 대부분의 예제에 팝업 메뉴를 채용하고 있으므로 예제를 실행할 때마다 마우스 오른쪽 버튼을 한번씩 눌러보기 바란다.

5-4.애니메이션

GLUT은 반복적인 처리가 필요할 때 주로 사용되는 타이머 이벤트도 제공한다. 타이머 구현은 운영체제마다 다르지만 GLUT의 다음 함수를 호출하면 운영체제에 상관없이 타이머를 만들 수 있다. 다음 함수로 타이머 콜백 함수를 등록한다.

void glutTimerFunc(unsigned int millis, void (*func)(int value), int value);

millis 후에 func 함수를 호출하며 인수로 value를 전달한다. value는 타이머 콜백으로 전달되어 작업 거리를 지시하는데 타이머의 용도가 하나뿐이라면 아무 값이나 주어도 상관없다. 콜백을 등록해 놓으면 millis 후에 콜백 함수가 호출된다. 주의할 것은 일반적인 타이머와는 달리 주기적으로 호출되는 것이 아니라 딱 한번만 호출된다는 것이다.

주기적으로 계속 호출하려면 콜백 함수에서 자신을 다시 호출해야 한다. 지속적으로 호출되는 방식에 비해 약간 불편하지만 매 호출시마다 다음 주기를 가변적으로 결정할 수 있다는 면에서 활용성은 오히려 더 높다. 더 이상 타이머 호출이 필요없으면 콜백을 재등록하지 않으면 된다. WinMain에서는 다만 시동만 걸어줄 뿐이고 다음 타이머 호출 시점은 콜백 함수가 자체적으로 결정할 수 있다.

타이머의 용도는 여러 가지가 있지만 대표적인 예가 애니메이션이다. 일정한 간격으로 출력을 계속 바꾸면 그림이 움직이는 것처럼 보인다. 애니메이션을 할 때는 더블 버퍼링을 사용하는 것이 좋다. 한 버퍼에서 지웠다 그리기를 계속 반복하면 깜박거림이 발생하여 눈에 거슬리므로 두 개의 버퍼를 교대로 사용한다. 초기화시에 GLUT_DOUBLE 플래그를 전달하면 OpenGL은 두 개의 버퍼를 준비한다.

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

두 버퍼는 각각 전면(Front buffer 또는 On Screen Buffer), 후면(Back Buffer 또는 Off Screen Buffer)라고 부른다. 전면 버퍼는 현재 화면에 출력된 버퍼이며 백 버퍼는 안쪽에 숨겨진 버퍼이다. 더블 버퍼링 옵션이 켜지면 그래픽 카드는 항상 전면 버퍼를 모니터에 뿌리지만 모든 그리기 동작은 백 버퍼에서 수행된다. 그래서 그리는 중간 과정이 사용자 눈에 보이지 않으며 깜박거림도 없다. 백 버퍼에 그림을 다 그렸으면 다음 함수로 버퍼를 통째로 교체한다.

glutSwapBuffers();

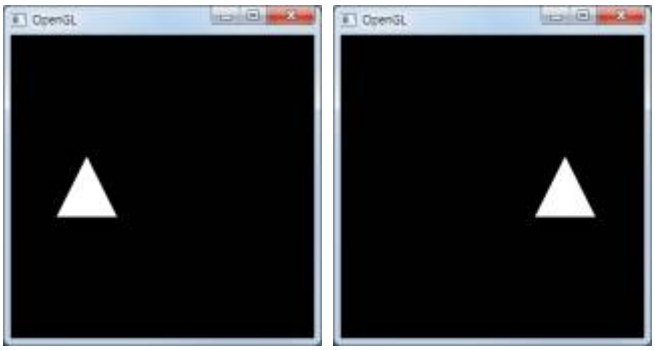
전면 버퍼와 후면 버퍼가 일시에 교체됨으로써 백 버퍼에 미리 준비해 놓은 그림이 뿅 하고 나타난다. 순간적으로 교체되므로 깜박거림은 전혀 느낄 수 없다. 이 과정을 계속 반복하면 애니메이션이 되는 것이다. 버퍼를 교체하는 것 자체가 출력이므로 glFlush는 호출하지 않아도 상관없다. 다음 예제는 삼각형 도형을 좌우로 이동시키는 간단한 애니메이션을 보여준다.

glutAni

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

void DoDisplay();
void DoTimer(int value);

const GLfloat size = 0.2;
GLfloat x;
GLfloat dx = 0.02;

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
      glutCreateWindow("OpenGL");
      glutDisplayFunc(DoDisplay);
      glutTimerFunc(30, DoTimer, 1);
      glutMainLoop();
      return 0;
}

void DoTimer(int value)
{
      x += dx;
      if (x + size > 1 || x - size < -1) {
            dx *= -1;
      }
      glutPostRedisplay();
      glutTimerFunc(30, DoTimer, 1);
}

void DoDisplay()
{
      glClear(GL_COLOR_BUFFER_BIT);

      glBegin(GL_POLYGON);
      glVertex2f(x, size);
      glVertex2f(x - size, -size);
      glVertex2f(x + size, -size);
      glEnd();
      glutSwapBuffers();
}
```

코드의 구조는 아주 간단하다. WinMain에서 DoTimer 함수를 0.03초후에 호출하도록 등록했다. 0.03초 후에 DoTimer가 호출되면 x값을 dx만큼 증감시키되 화면 가장자리에 닿으면 dx의 부호를 바꾸어 반대 방향으로 움직이도록 한다. dx가 0.02로 정의되어 있으므로 타이머 이벤트가 한번 발생할 때마다 삼각형이 0.02만큼 이동한다.

DoTimer의 마지막줄에서 자기 자신을 0.03초 후에 다시 호출함으로써 이동이 계속 반복된다. 이 처리가 생략되면 타이머는 딱 한번만 호출되므로 애니메이션이 실행되지 않을 것이다. 실행해 보면 삼각형이 좌우로 왔다리 갔다리 하면서 방황할 것이다.



이 예제는 더블 버퍼링과 타이머 콜백을 설명하기 위한 예제이며 도형의 x 좌표를 직접 조작하는 단순한 방법을 사용했다. OpenGL에는 애니메이션을 위한 더 좋은 방법들이 많이 준비되어 있다. 변환을 사용하면 이동 뿐만 아니라 회전, 확대 등의 다양한 애니메이션을 훨씬 더 빠른 속도로 구현할 수 있다.

5-5. Win32 OpenGL 예제

지금 우리는 윈도우즈 환경에서 GLUT으로 OpenGL 예제를 만들고 있다. GLUT은 어디까지나 플랫폼 추상층을 제공하는 도우미일 뿐 OpenGL의 필수 도구는 아니다. 이말은 즉 GLUT을 사용하지 않고도 AUX 라이브러리를 제공하는 임의의 플랫폼에서 동작하는 완벽한 OpenGL 프로그램을 만들 수 있다는 얘기이다.

윈도우즈는 OpenGL 지원을 위해 운영체제 차원에서 관련 wgl 함수들을 제공한다. 이 함수들을 사용하면 OpenGL을 위한 출력 표면을 만들 수 있고 이 표면에 대해 모든 OpenGL 함수가 출력을 내 보낼 수 있다. 그 외에 모든 윈도우즈 기능을 100% 활용할 수 있으므로 그래픽이나 입출력 모두 원하는 형태로 만들 수 있다.

과연 그것이 가능한지 GLUT을 사용하지 않고 순수한 Win32 API만으로 OpenGL 예제를 만들어 보자. 다음 예제는 일반적인 윈도우즈 응용 프로그램이다. 앞서 작성했던 예제에 비해 거대한 구조체가 등장하고 뭔가 복잡해 보이는 메시지 처리 함수도 사용되어 어지러워 보인다. 소스가 길어 보이지만 사실 Win32를 잘 하는 사람에게는 아주 쉬운 소스이다.

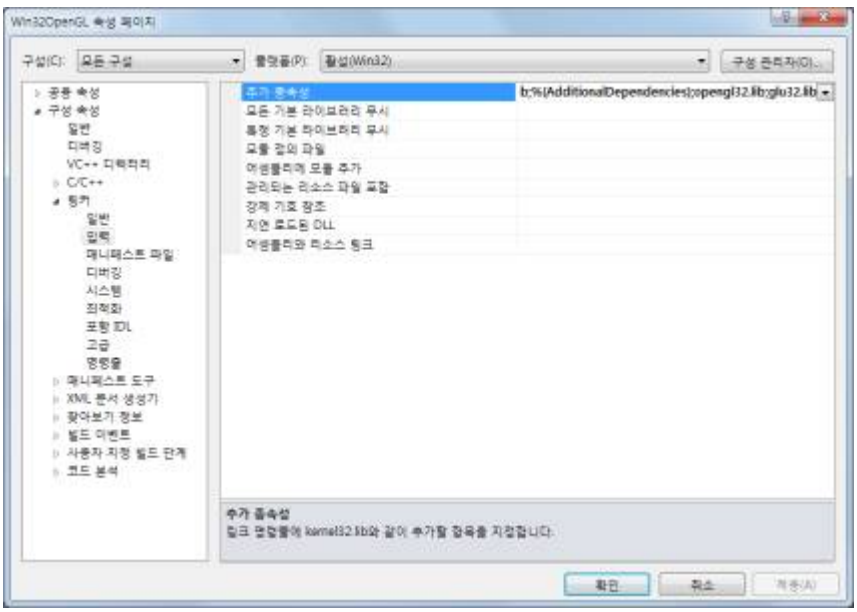
Win32OpenGL

```
#nclude <windows.h>
#nclude <gl/GL.h>
#nclude <gl/GLU.h>

LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
void DoDisplay();
HINSTANCE g_hInst;
HWND hWndMain;
LPCTSTR lpszClass=TEXT("OpenGL");
HDC hdc;
HGLRC hrc;

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
        ,LPSTR lpszCmdParam,int nCmdShow)
{
      HWND hWnd;
      MSG Message;
      WNDCLASS WndClass;
      g_hInst=hInstance;
```

```
	WndClass.cbClsExtra=0;
	WndClass.cbWndExtra=0;
	WndClass.hbrBackground=(HBRUSH)GetStockObject(BLACK_BRUSH);
	WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
	WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
	WndClass.hInstance=hInstance;
	WndClass.lpfnWndProc=WndProc;
	WndClass.lpszClassName=lpszClass;
	WndClass.lpszMenuName=NULL;
	WndClass.style=CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
	RegisterClass(&WndClass);

	hWnd=CreateWindow(lpszClass,lpszClass,
		WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_CLIPSIBLINGS,
		CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
		NULL,(HMENU)NULL,hInstance,NULL);
	ShowWindow(hWnd,nCmdShow);

	while (GetMessage(&Message,NULL,0,0)) {
		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}
	return (int)Message.wParam;
}

void DoDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT);

	glBegin(GL_TRIANGLES);
	glVertex2f(0.0, 0.5);
	glVertex2f(-0.5, -0.5);
	glVertex2f(0.5, -0.5);
	glEnd();
	glFinish();
}

LRESULT CALLBACK WndProc(HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM lParam)
{
	switch (iMessage) {
	case WM_CREATE:
		hWndMain=hWnd;
		PIXELFORMATDESCRIPTOR pfd;
		int nPixelFormat;

		hdc=GetDC(hWnd);
		memset(&pfd, 0, sizeof(pfd));

		pfd.nSize = sizeof(pfd);
		pfd.nVersion = 1;
		pfd.dwFlags = PFD_DRAW_TO_WINDOW |
				PFD_SUPPORT_OPENGL;
		pfd.iPixelType = PFD_TYPE_RGBA;
		pfd.cColorBits = 32;

		nPixelFormat = ChoosePixelFormat(hdc, &pfd);
		SetPixelFormat(hdc, nPixelFormat, &pfd);

		hrc=wglCreateContext(hdc);
		wglMakeCurrent(hdc, hrc);
```

```
            return 0;
        case WM_PAINT:
            DoDisplay();
            SetTextColor(hdc, RGB(255,255,0));
            SetBkMode(hdc,TRANSPARENT);
            TextOutA(hdc,10,10,"Win32 OpenGL",12);
            ValidateRect(hWnd, NULL);
            return 0;
        case WM_SIZE:
            glViewport( 0, 0, LOWORD(lParam), HIWORD(lParam));
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();

            glOrtho(-1, 1, -1, 1, 1, -1);

            glMatrixMode(GL_MODELVIEW );
            glLoadIdentity();
            return 0;
        case WM_DESTROY:
            wglMakeCurrent(hdc, NULL);
            wglDeleteContext(hrc);
            ReleaseDC(hWnd,hdc);
            PostQuitMessage(0);
            return 0;
        }
        return(DefWindowProc(hWnd,iMessage,wParam,lParam));
}
```

프로젝트를 만드는 방법은 기존의 예제와 동일하다. 그러나 glut.h가 빠짐으로 인해 몇 가지 추가 처리가 필요하다. 먼저 gl.h와 glu.h 헤더파일을 직접 인클루드해야 한다. 뿐만 아니라 라이브러리 링크란에 opengl32.lib와 glu32.lib도 연결해 주어야 한다. glut.h에 작성되어 있는 #pragma comment 문이 없어졌으므로 임포트 라이브러리 지정은 수동으로 바뀌었다.

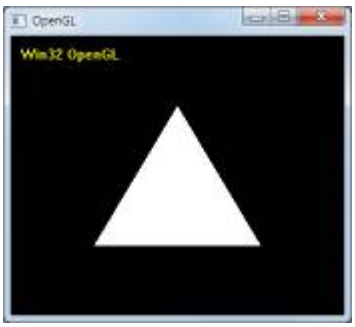


윈도우를 생성하는 것은 Win32의 고유 코드로 쉽게 처리할 수 있다. 하지만 일반적인 윈도우에 비해 몇가지 차이점이 있다. OpenGL은 자주 화면을 바꾸므로 반드시 CS_OWNDC 스타일을 지정하여 전용 DC를 할당해 주어야 빠른 그리기를 할 수 있다. 또 자식 윈도우와 형제 윈도우에 대한 클리핑을 지정하여 OpenGL이 다른 창 위에 그리지 않도록 해야 한다.

이렇게 만들어진 윈도우는 일반적인 윈도우일 뿐 OpenGL이 그리기에 사용할 수 있는 윈도우는 아니다. OpenGL이 창에 그리기를 하려면 랜더링 컨텍스트를 생성하여 윈도우의 DC와 연결해야 한다. 이 처리를 WM_CREATE에서 하고 있다. 랜더링 컨텍스트를 생성하기 위해 그리기 표면의 특성을 픽셀 포맷으로 작성하고 그래픽 카드가 지원하는 모드 중의 하나를 선택해야 한다. 이 작업은 PIXELFORMATDESCRIPTOR 구조체와 ChoosePixelFormat 함수로 수행하며 선택한 픽셀 포맷을 SetPixelFormat 함수로 결정한다. wglCreateContext 함수로 OpenGL 그리기를 위한 랜더링 컨텍스트를 생성하고 wglMakeCurrent 함수로 현재 컨텍스트를 활성화하면 이후부터 윈도우에 그리기를 수행할 수 있다.

윈도우의 크기가 결정되는 WM_SIZE 메시지에서 뷰 포트를 윈도우 크기에 맞게 설정하고 직교 투영 함수로 클리핑 영역을 설정한다. 이 예제의 WM_SIZE 처리는 지금까지의 예제에서는 없었던 것인데 그 이유는 GLUT이 무난하게 디폴트 처리를 해 주었기 때문이다. 지금은 그런 서비스가 없으므로 모든 것을 직접해야 한다. 뷰포트는 윈도우 크기에 맞추었고 클리핑 영역은 GLUT의 디폴트와 맞추었다. 물론 원하는대로 변경 가능하다.

실제 그리기를 수행하는 WM_PAINT에서는 도형을 그린다. 기존 예제와 비슷함을 강조하기 위해 DoDisplay 함수로 그리기 코드를 분리해 두었다. 랜더링 컨텍스트만 제대로 생성되어 있다면 OpenGL의 그리기 코드를 그대로 활용할 수 있다. 뿐만 아니라 덤으로 Win32의 GDI 출력도 같이 내 보낼 수 있다. 실행해 보면 삼각형과 함께 왼쪽 위에 문자열이 하나 출력될 것이다.



이 문자열은 Win32의 TextOut 함수로 수행하되 프로젝트 설정이 유니코드로 되어 있으므로 TextOutA 함수를 호출했다. 문자열은 OpenGL이 출력한 그림 위에 배치되므로 OpenGL이 완전히 그린 후에 별도로 그려야 한다. 그러기 위해 DoDisplay 함수의 끝에서 glFlush 대신 glFinish 함수를 호출했다. glFinish는 완전히 그릴 때까지 대기하는 특성이 있어 OpenGL 출력과 Win32 출력을 분리하는 역할을 한다. glFlush를 쓰면 다시 그리기를 즉시 하지 않으므로 GDI 출력이 OpenGL 출력에 덮여 버린다.

그까짓 문자열 하나 출력하는 게 뭐 대단하냐고 하겠지만 결코 그렇지 않다. OpenGL은 입체 도형은 그럴싸하게 잘 그려내지만 문자열 출력 기능은 따로 제공하지 않는다. 본연의 임무가 아니기 하고 그래픽 화면에서는 문자조차도 그림으로 그려야 하기 때문이다. 정 문자열을 출력하려면 일일이 비트맵 폰트를 만들어 그려야 한다. Win32는 간단한 함수 호출 하나만으로 문자열을 출력할 수 있고 색상, 모드 등을 자유로이 선택할 수 있으며 트루타입의 모든 이점을 활용할 수 있다.

이 예제의 모든 코드를 다 이해할 필요는 없다. 다만 이 실습을 통해 GLUT이 어떤 역할을 하는지, 장단점이 무엇인지만 파악하면 된다. 보다시피 동일한 예제를 만드는데도 Win32로는 굉장히 많은 코드가 필요하다. 모든 것을 일일이 직접 처리해야 한다. 더구나 이렇게 만든 프로그램은 윈도우즈 전용이므로 호환성에 지극히 불리하다. Win32 뿐만 아니라 매킨토시나 리눅스용 AUX도 동일한 문제점이 있다.

그러나 GLUT을 사용하지 않을 때의 이점 또한 만만치 않다. 위 예에서 보다시피 트루타입 글꼴을 편리하게 출력할 수 있으며 팝업 메뉴에 구분선이나 체크 표시도 달 수 있다. 키보드 키의 동시 입력도 처리할 수 있으며 멀티 스레드와 동기화, 파일 입출력, 네트워크, 데이터베이스 지원 등 해당 플랫폼이 제공하는 모든 이점을 완벽하게 누릴 수 있다. 그래서 상용 프로그램을 만들 때는 타겟 플랫폼의 AUX를 사용하지 않을 수 없다.

GLUT은 어디까지나 실습용, 테스트용의 라이브러리임을 잊지 말자. 물론 GLUT도 계속 업그레이드되고 있으므로 이것만으로도 상당한 수준의 프로그램을 만들 수 있지만 최종 제품을 위한 해결책이라 보기는 어렵

다. 학습에 필요한만큼만 연구해 두고 더 필요하면 그때 연구해도 늦지 않다. 그래서 여기서도 예제 제작에 딱 필요한만큼만 소개했다.