PROGRAM-PROGRAM CONTOH DARI REDBOOK OPENGL

Pengantar: Program-program di bawah ini diambil dari buku OpenGL Programming Guide (Redbook) Addison-Wesley Publishing Company). Versi e-booknya serta file programnya dapat didownload pada situs kuliah ini.

01: Program Menampilkan Poligon (Example 1-2)

```
/* Prog-01: Menampilkan POLIGON */
#include <gl/glut.h>
void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT);
	glColor3f (1.0, 1.0, 1.0);
	glBegin(GL_POLYGON);
		glVertex3f (0.25, 0.25, 0.0);
		glVertex3f (0.75, 0.25, 0.0);
		glVertex3f (0.75, 0.75, 0.0);
		glVertex3f (0.25, 0.75, 0.0);
	glEnd();
	glFlush ();
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize (250, 250);
	glutInitWindowPosition (100, 100);
	glutCreateWindow ("Program Pertama OpenGL");
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
	glutDisplayFunc(display);
	glutMainLoop();
	return 0;
}
```

02: Program Double-buffer untuk Animasi Poligon (Example 1-3)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
static GLfloat spin = 0.0;
void init(void)
{
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_FLAT);
}

void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glPushMatrix();
	glRotatef(spin, 0.0, 0.0, 1.0);
	glColor3f(1.0, 1.0, 1.0);
	glRectf(-25.0, -25.0, 25.0, 25.0);
	glPopMatrix();
	glutSwapBuffers();
}
```

```
void spinDisplay(void)
{
	spin = spin + 2.0;
	if (spin > 360.0)
	spin = spin - 360.0;
	glutPostRedisplay();
}
void reshape(int w, int h)
{
	glViewport (0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
	switch (button) {
	case GLUT_LEFT_BUTTON:
		if (state == GLUT_DOWN)
			glutIdleFunc(spinDisplay);
		break;
	case GLUT_MIDDLE_BUTTON:
		if (state == GLUT_DOWN)
			glutIdleFunc(NULL);
		break;
	default:
		break;
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize (250, 250);
	glutInitWindowPosition (100, 100);
	glutCreateWindow (argv[0]);
	init ();
	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutMouseFunc(mouse);
	glutMainLoop();
	return 0;
}
```

03: Menggambar Kubus (Example 3-1)

```
/* Prog-03: Menggambar Kubus */
#include <windows.h>
#include <GL/glut.h>
void init(void)
{
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_FLAT);
}
void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT);
	glColor3f (1.0, 1.0, 1.0);
	glLoadIdentity (); /* clear the matrix */
	/* viewing transformation */
	gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

```
        glScalef (1.0, 2.0, 1.0); /* modeling transformation */
        glutWireCube (1.0);
        glFlush ();
}
void reshape (int w, int h)
{
        glViewport (0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
        glMatrixMode (GL_MODELVIEW);
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize (500, 500);
        glutInitWindowPosition (100, 100);
        glutCreateWindow (argv[0]);
        init ();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```

04: Line Patterns (Example 2-5)

```
#include <windows.h>
#include <GL/glut.h>
#define drawOneLine(x1,y1,x2,y2) glBegin(GL_LINES); \
glVertex2f ((x1),(y1)); glVertex2f ((x2),(y2)); glEnd();
void init(void)
{
        glClearColor (0.0, 0.0, 0.0, 0.0);
        glShadeModel (GL_FLAT);
}
void display(void)
{
        int i;
        glClear (GL_COLOR_BUFFER_BIT);
        /* select white for all lines */
        glColor3f (1.0, 1.0, 1.0);
        /* in 1st row, 3 lines, each with a different stipple */
        glEnable (GL_LINE_STIPPLE);
        glLineStipple (1, 0x0101); /* dotted */
        drawOneLine (50.0, 125.0, 150.0, 125.0);
        glLineStipple (1, 0x00FF); /* dashed */
        drawOneLine (150.0, 125.0, 250.0, 125.0);
        glLineStipple (1, 0x1C47); /* dash/dot/dash */
        drawOneLine (250.0, 125.0, 350.0, 125.0);
        /* in 2nd row, 3 wide lines, each with different stipple */
        glLineWidth (5.0);
        glLineStipple (1, 0x0101); /* dotted */
        drawOneLine (50.0, 100.0, 150.0, 100.0);
        glLineStipple (1, 0x00FF); /* dashed */
        drawOneLine (150.0, 100.0, 250.0, 100.0);
        glLineStipple (1, 0x1C47); /* dash/dot/dash */
        drawOneLine (250.0, 100.0, 350.0, 100.0);
        glLineWidth (1.0);
        /* in 3rd row, 6 lines, with dash/dot/dash stipple */
        /* as part of a single connected line strip */
        glLineStipple (1, 0x1C47); /* dash/dot/dash */
```

```
        glBegin (GL_LINE_STRIP);
                for (i = 0; i < 7; i++)
                        glVertex2f (50.0 + ((GLfloat) i * 50.0), 75.0);
        glEnd ();
        /* in 4th row, 6 independent lines with same stipple */
        for (i = 0; i < 6; i++) {
                drawOneLine (50.0 + ((GLfloat) i * 50.0), 50.0,
                50.0 + ((GLfloat)(i+1) * 50.0), 50.0);
}
        /* in 5th row, 1 line, with dash/dot/dash stipple */
        /* and a stipple repeat factor of 5 */
        glLineStipple (5, 0x1C47); /* dash/dot/dash */
        drawOneLine (50.0, 25.0, 350.0, 25.0);
        glDisable (GL_LINE_STIPPLE);
        glFlush ();
}
void reshape (int w, int h)
{
        glViewport (0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize (400, 150);
        glutInitWindowPosition (100, 100);
        glutCreateWindow (argv[0]);
        init ();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```

05: Polygon Patterns (Example 2-6)

```
#include <windows.h>
#include <GL/glut.h>
void display(void)
{
        GLubyte fly[] = {
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
                0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,
                0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,
                0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,
                0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
                0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
                0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
                0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,
                0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,
                0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,
                0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,
                0x06, 0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30,
                0x18, 0xcc, 0x33, 0x18, 0x10, 0xc4, 0x23, 0x08,
                0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,
                0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08};
        GLubyte halftone[] = {
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
```

```
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
                0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55};

       glClear (GL_COLOR_BUFFER_BIT);
       glColor3f (1.0, 1.0, 1.0);
       /* draw one solid, unstippled rectangle, */
       /* then two stippled rectangles */
       glRectf (25.0, 25.0, 125.0, 125.0);
       glEnable (GL_POLYGON_STIPPLE);
       glPolygonStipple (fly);
       glRectf (125.0, 25.0, 225.0, 125.0);
       glPolygonStipple (halftone);
       glRectf (225.0, 25.0, 325.0, 125.0);
       glDisable (GL_POLYGON_STIPPLE);
       glFlush ();
}
void init (void)
{
       glClearColor (0.0, 0.0, 0.0, 0.0);
       glShadeModel (GL_FLAT);
}
void reshape (int w, int h)
{
       glViewport (0, 0, (GLsizei) w, (GLsizei) h);
       glMatrixMode (GL_PROJECTION);
       glLoadIdentity ();
       gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
int main(int argc, char** argv)
{
       glutInit(&argc, argv);
       glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
       glutInitWindowSize (350, 150);
       glutCreateWindow (argv[0]);
       init ();
       glutDisplayFunc(display);
       glutReshapeFunc(reshape);
       glutMainLoop();
       return 0;
}
```

06: Wireframe Sphere (Example 3-5)

```
#include <windows.h>
#include <GL/glut.h>
void init(void)
{
       glClearColor (0.0, 0.0, 0.0, 0.0);
       glShadeModel (GL_FLAT);
}
void display(void)
```

```
{
	GLdouble eqn[4] = {0.0, 1.0, 0.0, 0.0};
	GLdouble eqn2[4] = {1.0, 0.0, 0.0, 0.0};
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f (1.0, 1.0, 1.0);
	glPushMatrix();
	glTranslatef (0.0, 0.0, -5.0);
	/* clip lower half -- y < 0 */
	glClipPlane (GL_CLIP_PLANE0, eqn);
	glEnable (GL_CLIP_PLANE0);
	/* clip left half -- x < 0 */
	glClipPlane (GL_CLIP_PLANE1, eqn2);
	glEnable (GL_CLIP_PLANE1);
	glRotatef (90.0, 1.0, 0.0, 0.0);
	glutWireSphere(1.0, 20, 16);
	glPopMatrix();
	glFlush ();
}
void reshape (int w, int h)
{
	glViewport (0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity ();
	gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
	glMatrixMode (GL_MODELVIEW);
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize (500, 500);
	glutInitWindowPosition (100, 100);
	glutCreateWindow (argv[0]);
	init ();
	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```

07: Planetary System (Example 3-6)

```
#include <windows.h>
#include <GL/glut.h>
static int year = 0, day = 0;
void init(void)
{
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_FLAT);
}
void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT);
	glColor3f (1.0, 1.0, 1.0);
	glPushMatrix();
	glutWireSphere(1.0, 20, 16); /* draw sun */
	glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
	glTranslatef (2.0, 0.0, 0.0);
	glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
	glutWireSphere(0.2, 10, 8); /* draw smaller planet */
	glPopMatrix();
	glutSwapBuffers();
}
```

```
void reshape (int w, int h)
{
	glViewport (0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity ();
	gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
void keyboard (unsigned char key, int x, int y)
{
	switch (key) {
		case 'd':
			day = (day + 10) % 360;
			glutPostRedisplay();
			break;
		case 'D':
			day = (day - 10) % 360;
			glutPostRedisplay();
			break;
		case 'y':
			year = (year + 5) % 360;
			glutPostRedisplay();
			break;
		case 'Y':
			year = (year - 5) % 360;
			glutPostRedisplay();
			break;
		default:
			break;
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize (500, 500);
	glutInitWindowPosition (100, 100);
	glutCreateWindow (argv[0]);
	init ();
	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutKeyboardFunc(keyboard);
	glutMainLoop();
	return 0;
}
```

08: Tangan ROBOT

```
#include <windows.h>
#include <GL/glut.h>
static int shoulder = 0, elbow = 0;
void init(void)
{
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_FLAT);
}
void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT);
	glPushMatrix();
	glTranslatef (-1.0, 0.0, 0.0);
```

```
        glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
        glTranslatef (1.0, 0.0, 0.0);
        glPushMatrix();
        glScalef (2.0, 0.4, 1.0);
        glutWireCube (1.0);
        glPopMatrix();
        glTranslatef (1.0, 0.0, 0.0);
        glRotatef ((GLfloat) elbow, 0.0, 0.0, 1.0);
        glTranslatef (1.0, 0.0, 0.0);
        glPushMatrix();
        glScalef (2.0, 0.4, 1.0);
        glutWireCube (1.0);
        glPopMatrix();
        glPopMatrix();
        glutSwapBuffers();
}
void reshape (int w, int h)
{
        glViewport (0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        gluPerspective(65.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef (0.0, 0.0, -5.0);
}
void keyboard (unsigned char key, int x, int y)
{
        switch (key) {
                case 's': /* s key rotates at shoulder */
                        shoulder = (shoulder + 5) % 360;
                        glutPostRedisplay();
                        break;
                case 'S':
                        shoulder = (shoulder - 5) % 360;
                        glutPostRedisplay();
                        break;
                case 'e': /* e key rotates at elbow */
                        elbow = (elbow + 5) % 360;
                        glutPostRedisplay();
                        break;
                case 'E':
                        elbow = (elbow - 5) % 360;
                        glutPostRedisplay();
                        break;
                default:
                        break;
        }
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (500, 500);
        glutInitWindowPosition (100, 100);
        glutCreateWindow (argv[0]);
        init ();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMainLoop();
        return 0;
}
```

09: Geometric Processing Pipeline (Example 3-8)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glFlush();
}
void reshape(int w, int h)
{
	glViewport (0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective (45.0, (GLfloat) w/(GLfloat) h, 1.0, 100.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
	GLint viewport[4];
	GLdouble mvmatrix[16], projmatrix[16];
	GLint realy; /* OpenGL y coordinate position */
	GLdouble wx, wy, wz; /* returned world x, y, z coords */
	switch (button) {
	case GLUT_LEFT_BUTTON:
		if (state == GLUT_DOWN) {
			glGetIntegerv (GL_VIEWPORT, viewport);
			glGetDoublev (GL_MODELVIEW_MATRIX, mvmatrix);
			glGetDoublev (GL_PROJECTION_MATRIX, projmatrix);
			/* note viewport[3] is height of window in pixels */
			realy = viewport[3] - (GLint) y - 1;
			printf ("Coordinates at cursor are (%4d, %4d)\n",
			x, realy);
			gluUnProject ((GLdouble) x, (GLdouble) realy, 0.0,
			mvmatrix, projmatrix, viewport, &wx, &wy, &wz);
			printf ("World coords at z=0.0 are (%f, %f, %f)\n",
			wx, wy, wz);
			gluUnProject ((GLdouble) x, (GLdouble) realy, 1.0,
			mvmatrix, projmatrix, viewport, &wx, &wy, &wz);
			printf ("World coords at z=1.0 are (%f, %f, %f)\n",
			wx, wy, wz);
		}
		break;
	case GLUT_RIGHT_BUTTON:
		if (state == GLUT_DOWN)
			exit(0);
		break;
	default:
		break;
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize (500, 500);
	glutInitWindowPosition (100, 100);
	glutCreateWindow (argv[0]);
	glutDisplayFunc(display);
```

```
	glutReshapeFunc(reshape);
	glutMouseFunc(mouse);
	glutMainLoop();
	return 0;
}
```

10: Menggambar Segitiga Warna-warni Bergradasi (Example 4-1)

```
#include <windows.h>
#include <GL/glut.h>
void init(void)
{
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_SMOOTH);
}
void triangle(void)
{
	glBegin (GL_TRIANGLES);
	glColor3f (1.0, 0.0, 0.0);
	glVertex2f (5.0, 5.0);
	glColor3f (0.0, 1.0, 0.0);
	glVertex2f (25.0, 5.0);
	glColor3f (0.0, 0.0, 1.0);
	glVertex2f (5.0, 25.0);
	glEnd();
}
void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT);
	triangle ();
	glFlush ();
}
void reshape (int w, int h)
{
	glViewport (0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity ();
	if (w <= h)
		gluOrtho2D (0.0, 30.0, 0.0, 30.0*(GLfloat) h/(GLfloat) w);
	else
		gluOrtho2D (0.0, 30.0*(GLfloat) w/(GLfloat) h, 0.0, 30.0);
	glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize (500, 500);
	glutInitWindowPosition (100, 100);
	glutCreateWindow (argv[0]);
	init ();
	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```

11: Menggambar Bola Bundar 3D Bercahaya (Example 5-1)

```
#include <windows.h>
#include <GL/glut.h>
void init(void)
```

```
{
	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat mat_shininess[] = { 50.0 };
	GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_SMOOTH);
	glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
	glLightfv(GL_LIGHT0, GL_POSITION, light_position);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_DEPTH_TEST);
}
void display(void)
{
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glutSolidSphere (1.0, 20, 16);
	glFlush ();
}
void reshape (int w, int h)
{
	glViewport (0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
			1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
	else
		glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
			1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize (500, 500);
	glutInitWindowPosition (100, 100);
	glutCreateWindow (argv[0]);
	init ();
	glutDisplayFunc(display);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```

12: Campuran Warna (Blending) (Example 6-1)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
static int leftFirst = GL_TRUE;
/* Initialize alpha blending function. */
static void init(void)
{
	glEnable (GL_BLEND);
	glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	glShadeModel (GL_FLAT);
	glClearColor (0.0, 0.0, 0.0, 0.0);
}
static void drawLeftTriangle(void)
{
```

```
	/* draw yellow triangle on LHS of screen */
	glBegin (GL_TRIANGLES);
	glColor4f(1.0, 1.0, 0.0, 0.75);
	glVertex3f(0.1, 0.9, 0.0);
	glVertex3f(0.1, 0.1, 0.0);
	glVertex3f(0.7, 0.5, 0.0);
	glEnd();
}
static void drawRightTriangle(void)
{
	/* draw cyan triangle on RHS of screen */
	glBegin (GL_TRIANGLES);
	glColor4f(0.0, 1.0, 1.0, 0.75);
	glVertex3f(0.9, 0.9, 0.0);
	glVertex3f(0.3, 0.5, 0.0);
	glVertex3f(0.9, 0.1, 0.0);
	glEnd();
}
void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	if (leftFirst) {
		drawLeftTriangle();
		drawRightTriangle();
	}
	else {
		drawRightTriangle();
		drawLeftTriangle();
	}
	glFlush();
}
void reshape(int w, int h)
{
	glViewport(0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		gluOrtho2D (0.0, 1.0, 0.0, 1.0*(GLfloat)h/(GLfloat)w);
	else
		gluOrtho2D (0.0, 1.0*(GLfloat)w/(GLfloat)h, 0.0, 1.0);
}
void keyboard(unsigned char key, int x, int y)
{
	switch (key) {
		case 't':
		case 'T':
			leftFirst = !leftFirst;
			glutPostRedisplay();
			break;
		case 27: /* Escape key */
			exit(0);
			break;
		default:
		break;
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize (200, 200);
	glutCreateWindow (argv[0]);
	init();
```

```
	glutReshapeFunc (reshape);
	glutKeyboardFunc (keyboard);
	glutDisplayFunc (display);
	glutMainLoop();
	return 0;
}
```

13: Campuran Warna 3D (Example 6-2)

```
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include <GL/glut.h>
#define MAXZ 8.0
#define MINZ -8.0
#define ZINC 0.4
static float solidZ = MAXZ;
static float transparentZ = MINZ;
static GLuint sphereList, cubeList;
static void init(void)
{
	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 0.15 };
	GLfloat mat_shininess[] = { 100.0 };
	GLfloat position[] = { 0.5, 0.5, 1.0, 0.0 };
	glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
	glLightfv(GL_LIGHT0, GL_POSITION, position);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_DEPTH_TEST);
	sphereList = glGenLists(1);
	glNewList(sphereList, GL_COMPILE);
		glutSolidSphere (0.4, 16, 16);
	glEndList();
	cubeList = glGenLists(1);
	glNewList(cubeList, GL_COMPILE);
		glutSolidCube (0.6);
	glEndList();
}
void display(void)
{
	GLfloat mat_solid[] = { 0.75, 0.75, 0.0, 1.0 };
	GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 };
	GLfloat mat_transparent[] = { 0.0, 0.8, 0.8, 0.6 };
	GLfloat mat_emission[] = { 0.0, 0.3, 0.3, 0.6 };
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushMatrix ();
	glTranslatef (-0.15, -0.15, solidZ);
	glMaterialfv(GL_FRONT, GL_EMISSION, mat_zero);
	glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_solid);
	glCallList (sphereList);
	glPopMatrix ();
	glPushMatrix ();
	glTranslatef (0.15, 0.15, transparentZ);
	glRotatef (15.0, 1.0, 1.0, 0.0);
	glRotatef (30.0, 0.0, 1.0, 0.0);
	glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
	glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);
	glEnable (GL_BLEND);
	glDepthMask (GL_FALSE);
	glBlendFunc (GL_SRC_ALPHA, GL_ONE);
	glCallList (cubeList);
	glDepthMask (GL_TRUE);
```

```
	glDisable (GL_BLEND);
	glPopMatrix ();
	glutSwapBuffers();
}
void reshape(int w, int h)
{
	glViewport(0, 0, (GLint) w, (GLint) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
		1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
	else
		glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
		1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void animate(void)
{
	if (solidZ <= MINZ || transparentZ >= MAXZ)
		glutIdleFunc(NULL);
	else {
		solidZ -= ZINC;
		transparentZ += ZINC;
		glutPostRedisplay();
	}
}
void keyboard(unsigned char key, int x, int y)
{
	switch (key) {
		case 'a':
		case 'A':
			solidZ = MAXZ;
			transparentZ = MINZ;
			glutIdleFunc(animate);
			break;
		case 'r':
		case 'R':
			solidZ = MAXZ;
			transparentZ = MINZ;
			glutPostRedisplay();
			break;
		case 27:
			exit(0);
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(500, 500);
	glutCreateWindow(argv[0]);
	init();
	glutReshapeFunc(reshape);
	glutKeyboardFunc(keyboard);
	glutDisplayFunc(display);
	glutMainLoop();
	return 0;
}
```

14: Antialias Garis (Example 6-3)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
static float rotAngle = 0.;
void init(void)
{
	GLfloat values[2];
	glGetFloatv (GL_LINE_WIDTH_GRANULARITY, values);
	printf ("GL_LINE_WIDTH_GRANULARITY value is %3.1f\n",values[0]);
	glGetFloatv (GL_LINE_WIDTH_RANGE, values);
	printf ("GL_LINE_WIDTH_RANGE values are %3.1f %3.1f\n",values[0],
values[1]);
	glEnable (GL_LINE_SMOOTH);
	glEnable (GL_BLEND);
	glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	glHint (GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
	glLineWidth (1.5);
	glClearColor(0.0, 0.0, 0.0, 0.0);
}
void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f (0.0, 1.0, 0.0);
	glPushMatrix();
	glRotatef(-rotAngle, 0.0, 0.0, 0.1);
	glBegin (GL_LINES);
		glVertex2f (-0.5, 0.5);
		glVertex2f (0.5, -0.5);
	glEnd ();
	glPopMatrix();
	glColor3f (0.0, 0.0, 1.0);
	glPushMatrix();
	glRotatef(rotAngle, 0.0, 0.0, 0.1);
	glBegin (GL_LINES);
		glVertex2f (0.5, 0.5);
		glVertex2f (-0.5, -0.5);
	glEnd ();
	glPopMatrix();
	glFlush();
}
void reshape(int w, int h)
{
	glViewport(0, 0, (GLint) w, (GLint) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		gluOrtho2D (-1.0, 1.0,-1.0*(GLfloat)h/(GLfloat)w,
1.0*(GLfloat)h/(GLfloat)w);
	else
		gluOrtho2D (-1.0*(GLfloat)w/(GLfloat)h,1.0*(GLfloat)w/(GLfloat)h,
-1.0, 1.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y)
{
	switch (key) {
		case 'r':
		case 'R':
			rotAngle += 20.;
			if (rotAngle >= 360.) rotAngle = 0.;
			glutPostRedisplay();
```

```
                break;
        case 27: /* Escape Key */
                exit(0);
                break;
        default:
                break;
    }
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (200, 200);
    glutCreateWindow (argv[0]);
    init();
    glutReshapeFunc (reshape);
    glutKeyboardFunc (keyboard);
    glutDisplayFunc (display);
    glutMainLoop();
    return 0;
}
```

15: Antialis Color Index (Example 6-4)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#define RAMPSIZE 16
#define RAMP1START 32
#define RAMP2START 48
static float rotAngle = 0.;
void init(void)
{
    int i;
    for (i = 0; i < RAMPSIZE; i++) {
        GLfloat shade;
        shade = (GLfloat) i/(GLfloat) RAMPSIZE;
        glutSetColor(RAMP1START+(GLint)i, 0., shade, 0.);
        glutSetColor(RAMP2START+(GLint)i, 0., 0., shade);
    }
    glEnable (GL_LINE_SMOOTH);
    glHint (GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
    glLineWidth (1.5);
    glClearIndex ((GLfloat) RAMP1START);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glIndexi(RAMP1START);
    glPushMatrix();
    glRotatef(-rotAngle, 0.0, 0.0, 0.1);
    glBegin (GL_LINES);
        glVertex2f (-0.5, 0.5);
        glVertex2f (0.5, -0.5);
    glEnd ();
    glPopMatrix();
    glIndexi(RAMP2START);
    glPushMatrix();
    glRotatef(rotAngle, 0.0, 0.0, 0.1);
    glBegin (GL_LINES);
        glVertex2f (0.5, 0.5);
        glVertex2f (-0.5, -0.5);
    glEnd ();
```

```
	glPopMatrix();
	glFlush();
}
void reshape(int w, int h)
{
	glViewport(0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		gluOrtho2D (-1.0, 1.0,-1.0*(GLfloat)h/(GLfloat)w,
1.0*(GLfloat)h/(GLfloat)w);
	else
		gluOrtho2D (-1.0*(GLfloat)w/(GLfloat)h,1.0*(GLfloat)w/(GLfloat)h,
-1.0, 1.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y)
{
	switch (key) {
		case 'r':
		case 'R':
			rotAngle += 20.;
			if (rotAngle >= 360.) rotAngle = 0.;
				glutPostRedisplay();
			break;
		case 27: /* Escape Key */
			exit(0);
			break;
		default:
			break;
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_INDEX);
	glutInitWindowSize (200, 200);
	glutCreateWindow (argv[0]);
	init();
	glutReshapeFunc (reshape);
	glutKeyboardFunc (keyboard);
	glutDisplayFunc (display);
	glutMainLoop();
	return 0;
}
```

16: Antialias Filled Polygons (Example 6-5)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
GLboolean polySmooth = GL_TRUE;
static void init(void)
{
	glCullFace (GL_BACK);
	glEnable (GL_CULL_FACE);
	glBlendFunc (GL_SRC_ALPHA_SATURATE, GL_ONE);
	glClearColor (0.0, 0.0, 0.0, 0.0);
}
#define NFACE 6
```

```
#define NVERT 8
void drawCube(GLdouble x0, GLdouble x1, GLdouble y0,
	GLdouble y1, GLdouble z0, GLdouble z1)
{
	static GLfloat v[8][3];
	static GLfloat c[8][4] = {
		{0.0, 0.0, 0.0, 1.0}, {1.0, 0.0, 0.0, 1.0},
		{0.0, 1.0, 0.0, 1.0}, {1.0, 1.0, 0.0, 1.0},
		{0.0, 0.0, 1.0, 1.0}, {1.0, 0.0, 1.0, 1.0},
		{0.0, 1.0, 1.0, 1.0}, {1.0, 1.0, 1.0, 1.0}
	};
	/* indices of front, top, left, bottom, right, back faces */
	static GLubyte indices[NFACE][4] = {
		{4, 5, 6, 7}, {2, 3, 7, 6}, {0, 4, 7, 3},
		{0, 1, 5, 4}, {1, 5, 6, 2}, {0, 3, 2, 1}
	};
	v[0][0] = v[3][0] = v[4][0] = v[7][0] = x0;
	v[1][0] = v[2][0] = v[5][0] = v[6][0] = x1;
	v[0][1] = v[1][1] = v[4][1] = v[5][1] = y0;
	v[2][1] = v[3][1] = v[6][1] = v[7][1] = y1;
	v[0][2] = v[1][2] = v[2][2] = v[3][2] = z0;
	v[4][2] = v[5][2] = v[6][2] = v[7][2] = z1;
	#ifdef GL_VERSION_1_1
		glEnableClientState (GL_VERTEX_ARRAY);
		glEnableClientState (GL_COLOR_ARRAY);
		glVertexPointer (3, GL_FLOAT, 0, v);
		glColorPointer (4, GL_FLOAT, 0, c);
		glDrawElements(GL_QUADS, NFACE*4, GL_UNSIGNED_BYTE, indices);
		glDisableClientState (GL_VERTEX_ARRAY);
		glDisableClientState (GL_COLOR_ARRAY);
	#else
		printf ("If this is GL Version 1.0, ");
		printf ("vertex arrays are not supported.\n");
		exit(1);
	#endif
}
/* Note: polygons must be drawn from front to back
* for proper blending.
*/
void display(void)
{
	if (polySmooth) {
		glClear (GL_COLOR_BUFFER_BIT);
		glEnable (GL_BLEND);
		glEnable (GL_POLYGON_SMOOTH);
		glDisable (GL_DEPTH_TEST);
	}
	else {
		glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		glDisable (GL_BLEND);
		glDisable (GL_POLYGON_SMOOTH);
		glEnable (GL_DEPTH_TEST);
	}
	glPushMatrix ();
	glTranslatef (0.0, 0.0, -8.0);
	glRotatef (30.0, 1.0, 0.0, 0.0);
	glRotatef (60.0, 0.0, 1.0, 0.0);
	drawCube(-0.5, 0.5, -0.5, 0.5, -0.5, 0.5);
	glPopMatrix ();
	glFlush ();
	}
void reshape(int w, int h)
{
```

```
	glViewport(0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(30.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y)
{
	switch (key) {
		case 't':
		case 'T':
			polySmooth = !polySmooth;
			glutPostRedisplay();
			break;
		case 27:
			exit(0); /* Escape key */
			break;
		default:
			break;
	}
}
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_ALPHA | GLUT_DEPTH);
	glutInitWindowSize(200, 200);
	glutCreateWindow(argv[0]);
	init ();
	glutReshapeFunc (reshape);
	glutKeyboardFunc (keyboard);
	glutDisplayFunc (display);
	glutMainLoop();
	return 0;
}
```

17: Fogged Spehere in RGBA Mode (Example 6-6)

```
#include <windows.h>
#include <math.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
static GLint fogMode;
static void init(void)
{
	GLfloat position[] = { 0.5, 0.5, 3.0, 0.0 };
	glEnable(GL_DEPTH_TEST);
	glLightfv(GL_LIGHT0, GL_POSITION, position);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	{
		GLfloat mat[3] = {0.1745, 0.01175, 0.01175};
		glMaterialfv (GL_FRONT, GL_AMBIENT, mat);
		mat[0] = 0.61424; mat[1] = 0.04136; mat[2] = 0.04136;
		glMaterialfv (GL_FRONT, GL_DIFFUSE, mat);
		mat[0] = 0.727811; mat[1] = 0.626959; mat[2] = 0.626959;
		glMaterialfv (GL_FRONT, GL_SPECULAR, mat);
		glMaterialf (GL_FRONT, GL_SHININESS, 0.6*128.0);
	}
	glEnable(GL_FOG);
	{
		GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};
```

```
                fogMode = GL_EXP;
                glFogi (GL_FOG_MODE, fogMode);
                glFogfv (GL_FOG_COLOR, fogColor);
                glFogf (GL_FOG_DENSITY, 0.35);
                glHint (GL_FOG_HINT, GL_DONT_CARE);
                glFogf (GL_FOG_START, 1.0);
                glFogf (GL_FOG_END, 5.0);
        }
        glClearColor(0.5, 0.5, 0.5, 1.0); /* fog color */
}
static void renderSphere (GLfloat x, GLfloat y, GLfloat z)
{
        glPushMatrix();
        glTranslatef (x, y, z);
        glutSolidSphere(0.4, 16, 16);
        glPopMatrix();
}
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        renderSphere (-2., -0.5, -1.0);
        renderSphere (-1., -0.5, -2.0);
        renderSphere (0., -0.5, -3.0);
        renderSphere (1., -0.5, -4.0);
        renderSphere (2., -0.5, -5.0);
        glFlush();
}
void reshape(int w, int h)
{
        glViewport(0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
                glOrtho (-2.5, 2.5, -
2.5*(GLfloat)h/(GLfloat)w,2.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
        else
                glOrtho (-2.5*(GLfloat)w/(GLfloat)h,2.5*(GLfloat)w/(GLfloat)h, -
2.5, 2.5, -10.0, 10.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity ();
}
void keyboard(unsigned char key, int x, int y)
{
        switch (key) {
                case 'f':
                case 'F':
                        if (fogMode == GL_EXP) {
                                fogMode = GL_EXP2;
                                printf ("Fog mode is GL_EXP2\n");
                        }
                        else if (fogMode == GL_EXP2) {
                                fogMode = GL_LINEAR;
                                printf ("Fog mode is GL_LINEAR\n");
                        }
                        else if (fogMode == GL_LINEAR) {
                                fogMode = GL_EXP;
                                printf ("Fog mode is GL_EXP\n");
                        }
                        glFogi (GL_FOG_MODE, fogMode);
                        glutPostRedisplay();
                        break;
                case 27:
                        exit(0);
```

```
                    break;
                default:
                break;
        }
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow(argv[0]);
        init();
        glutReshapeFunc (reshape);
        glutKeyboardFunc (keyboard);
        glutDisplayFunc (display);
        glutMainLoop();
        return 0;
}
```

18: Menggambar Torus menggunakan Display List (Example 7.1 torus.c)

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

#define PI_ 3.14159265358979323846

GLuint theTorus;

/* Draw a torus */
static void torus(int numc, int numt)
{
   int i, j, k;
   double s, t, x, y, z, twopi;

   twopi = 2 * PI_;
   for (i = 0; i < numc; i++) {
      glBegin(GL_QUAD_STRIP);
      for (j = 0; j <= numt; j++) {
         for (k = 1; k >= 0; k--) {
            s = (i + k) % numc + 0.5;
            t = j % numt;

            x = (1+.1*cos(s*twopi/numc))*cos(t*twopi/numt);
            y = (1+.1*cos(s*twopi/numc))*sin(t*twopi/numt);
            z = .1 * sin(s * twopi / numc);
            glVertex3f(x, y, z);
         }
      }
      glEnd();
   }
}

/* Create display list with Torus and initialize state */
static void init(void)
{
```

```
   theTorus = glGenLists (1);
   glNewList(theTorus, GL_COMPILE);
   torus(8, 25);
   glEndList();

   glShadeModel(GL_FLAT);
   glClearColor(0.0, 0.0, 0.0, 0.0);
}

/* Clear window and draw torus */
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glCallList(theTorus);
   glFlush();
}

/* Handle window resize */
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(30, (GLfloat) w/(GLfloat) h, 1.0, 100.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
}

/* Rotate about x-axis when "x" typed; rotate about y-axis
   when "y" typed; "i" returns torus to original view */
void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
   case 'x':
   case 'X':
      glRotatef(30.,1.0,0.0,0.0);
      glutPostRedisplay();
      break;
   case 'y':
   case 'Y':
      glRotatef(30.,0.0,1.0,0.0);
      glutPostRedisplay();
      break;
   case 'i':
   case 'I':
      glLoadIdentity();
      gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
      glutPostRedisplay();
      break;
   case 27:
      exit(0);
      break;
   }
}
```

```
int main(int argc, char **argv)
{
   glutInitWindowSize(200, 200);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

19: Menggunakan display list (Example 7-2 : Using a Display List: list.c)

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>

GLuint listName;

static void init (void)
{
   listName = glGenLists (1);
   glNewList (listName, GL_COMPILE);
      glColor3f (1.0, 0.0, 0.0);  /*  current color red  */
      glBegin (GL_TRIANGLES);
      glVertex2f (0.0, 0.0);
      glVertex2f (1.0, 0.0);
      glVertex2f (0.0, 1.0);
      glEnd ();
      glTranslatef (1.5, 0.0, 0.0); /*  move position  */
   glEndList ();
   glShadeModel (GL_FLAT);
}

static void drawLine (void)
{
   glBegin (GL_LINES);
   glVertex2f (0.0, 0.5);
   glVertex2f (15.0, 0.5);
   glEnd ();
}

void display(void)
{
   GLuint i;

   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (0.0, 1.0, 0.0);  /*  current color green  */
   for (i = 0; i < 10; i++)    /*  draw 10 triangles    */
      glCallList (listName);
   drawLine ();  /*  is this line green?  NO!  */
                 /*  where is the line drawn?  */
   glFlush ();
```

```
}

void reshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      gluOrtho2D (0.0, 2.0, -0.5 * (GLfloat) h/(GLfloat) w,
         1.5 * (GLfloat) h/(GLfloat) w);
   else
      gluOrtho2D (0.0, 2.0 * (GLfloat) w/(GLfloat) h, -0.5, 1.5);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(650, 50);
   glutCreateWindow(argv[0]);
   init ();
   glutReshapeFunc (reshape);
   glutDisplayFunc (display);
   glutKeyboardFunc (keyboard);
   glutMainLoop();
   return 0;
}
```

20: Example 7-5 : Multiple Display Lists to Define a Stroked Font: stroke.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <string.h>

#define PT 1
#define STROKE 2
#define END 3

typedef struct charpoint {
   GLfloat   x, y;
```

```
   int    type;
} CP;

CP Adata[] = {
   { 0, 0, PT}, {0, 9, PT}, {1, 10, PT}, {4, 10, PT},
   {5, 9, PT}, {5, 0, STROKE}, {0, 5, PT}, {5, 5, END}
};

CP Edata[] = {
   {5, 0, PT}, {0, 0, PT}, {0, 10, PT}, {5, 10, STROKE},
   {0, 5, PT}, {4, 5, END}
};

CP Pdata[] = {
   {0, 0, PT}, {0, 10, PT},  {4, 10, PT}, {5, 9, PT}, {5, 6, PT},
   {4, 5, PT}, {0, 5, END}
};

CP Rdata[] = {
   {0, 0, PT}, {0, 10, PT},  {4, 10, PT}, {5, 9, PT}, {5, 6, PT},
   {4, 5, PT}, {0, 5, STROKE}, {3, 5, PT}, {5, 0, END}
};

CP Sdata[] = {
   {0, 1, PT}, {1, 0, PT}, {4, 0, PT}, {5, 1, PT}, {5, 4, PT},
   {4, 5, PT}, {1, 5, PT}, {0, 6, PT}, {0, 9, PT}, {1, 10, PT},
   {4, 10, PT}, {5, 9, END}
};

/*  drawLetter() interprets the instructions from the array
 *  for that letter and renders the letter with line segments.
 */
static void drawLetter(CP *l)
{
   glBegin(GL_LINE_STRIP);
   while (1) {
      switch (l->type) {
         case PT:
            glVertex2fv(&l->x);
            break;
         case STROKE:
            glVertex2fv(&l->x);
            glEnd();
            glBegin(GL_LINE_STRIP);
            break;
         case END:
            glVertex2fv(&l->x);
            glEnd();
            glTranslatef(8.0, 0.0, 0.0);
            return;
      }
      l++;
   }
}

/*  Create a display list for each of 6 characters    */
static void init (void)
```

```
{
   GLuint base;

   glShadeModel (GL_FLAT);

   base = glGenLists (128);
   glListBase(base);
   glNewList(base+'A', GL_COMPILE); drawLetter(Adata); glEndList();
   glNewList(base+'E', GL_COMPILE); drawLetter(Edata); glEndList();
   glNewList(base+'P', GL_COMPILE); drawLetter(Pdata); glEndList();
   glNewList(base+'R', GL_COMPILE); drawLetter(Rdata); glEndList();
   glNewList(base+'S', GL_COMPILE); drawLetter(Sdata); glEndList();
   glNewList(base+' ', GL_COMPILE); glTranslatef(8.0, 0.0, 0.0);
glEndList();
}

char *test1 = "A SPARE SERAPE APPEARS AS";
char *test2 = "APES PREPARE RARE PEPPERS";

static void printStrokedString(char *s)
{
   GLsizei len = strlen(s);
   glCallLists(len, GL_BYTE, (GLbyte *)s);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glPushMatrix();
   glScalef(2.0, 2.0, 2.0);
   glTranslatef(10.0, 30.0, 0.0);
   printStrokedString(test1);
   glPopMatrix();
   glPushMatrix();
   glScalef(2.0, 2.0, 2.0);
   glTranslatef(10.0, 13.0, 0.0);
   printStrokedString(test2);
   glPopMatrix();
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case ' ':
         glutPostRedisplay();
         break;
      case 27:
```

```
         exit(0);
   }
}

/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (440, 120);
   glutCreateWindow ("stroke");
   init ();
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

21: Example 8-1 : Drawing a Bitmapped Character: drawf.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>

GLubyte rasters[24] = {
   0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
   0xff, 0x00, 0xff, 0x00, 0xc0, 0x00, 0xc0, 0x00, 0xc0, 0x00,
   0xff, 0xc0, 0xff, 0xc0};

void init(void)
{
   glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
   glClearColor (0.0, 0.0, 0.0, 0.0);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glRasterPos2i (20, 20);
   glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
   glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
   glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, rasters);
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho (0, w, 0, h, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
```

```
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
   }
}

/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(100, 100);
   glutInitWindowPosition(100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

22: Example 8-2 : Drawing a Complete Font: font.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <string.h>

GLubyte space[] =
{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00};

GLubyte letters[][13] = {
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xff, 0xc3, 0xc3, 0xc3, 0x66,
0x3c, 0x18},
{0x00, 0x00, 0xfe, 0xc7, 0xc3, 0xc3, 0xc7, 0xfe, 0xc7, 0xc3, 0xc3,
0xc7, 0xfe},
{0x00, 0x00, 0x7e, 0xe7, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0,
0xe7, 0x7e},
{0x00, 0x00, 0xfc, 0xce, 0xc7, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc7,
0xce, 0xfc},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0xc0, 0xc0, 0xfc, 0xc0, 0xc0, 0xc0,
0xc0, 0xff},
{0x00, 0x00, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xfc, 0xc0, 0xc0,
0xc0, 0xff},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xcf, 0xc0, 0xc0, 0xc0, 0xc0,
0xe7, 0x7e},
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xff, 0xc3, 0xc3, 0xc3,
0xc3, 0xc3},
```

```
{0x00, 0x00, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x7e},
{0x00, 0x00, 0x7c, 0xee, 0xc6, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
0x06, 0x06},
{0x00, 0x00, 0xc3, 0xc6, 0xcc, 0xd8, 0xf0, 0xe0, 0xf0, 0xd8, 0xcc,
0xc6, 0xc3},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0,
0xc0, 0xc0},
{0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xdb, 0xff, 0xff,
0xe7, 0xc3},
{0x00, 0x00, 0xc7, 0xc7, 0xcf, 0xcf, 0xdf, 0xdb, 0xfb, 0xf3, 0xf3,
0xe3, 0xe3},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3,
0xe7, 0x7e},
{0x00, 0x00, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xfe, 0xc7, 0xc3, 0xc3,
0xc7, 0xfe},
{0x00, 0x00, 0x3f, 0x6e, 0xdf, 0xdb, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3,
0x66, 0x3c},
{0x00, 0x00, 0xc3, 0xc6, 0xcc, 0xd8, 0xf0, 0xfe, 0xc7, 0xc3, 0xc3,
0xc7, 0xfe},
{0x00, 0x00, 0x7e, 0xe7, 0x03, 0x03, 0x07, 0x7e, 0xe0, 0xc0, 0xc0,
0xe7, 0x7e},
{0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0xff},
{0x00, 0x00, 0x7e, 0xe7, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3, 0xc3,
0xc3, 0xc3},
{0x00, 0x00, 0x18, 0x3c, 0x3c, 0x66, 0x66, 0xc3, 0xc3, 0xc3, 0xc3,
0xc3, 0xc3},
{0x00, 0x00, 0xc3, 0xe7, 0xff, 0xff, 0xdb, 0xdb, 0xc3, 0xc3, 0xc3,
0xc3, 0xc3},
{0x00, 0x00, 0xc3, 0x66, 0x66, 0x3c, 0x3c, 0x18, 0x3c, 0x3c, 0x66,
0x66, 0xc3},
{0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c, 0x3c, 0x66,
0x66, 0xc3},
{0x00, 0x00, 0xff, 0xc0, 0xc0, 0x60, 0x30, 0x7e, 0x0c, 0x06, 0x03,
0x03, 0xff}
};

GLuint fontOffset;

void makeRasterFont(void)
{
   GLuint i, j;
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

   fontOffset = glGenLists (128);
   for (i = 0,j = 'A'; i < 26; i++,j++) {
      glNewList(fontOffset + j, GL_COMPILE);
      glBitmap(8, 13, 0.0, 2.0, 10.0, 0.0, letters[i]);
      glEndList();
   }
   glNewList(fontOffset + ' ', GL_COMPILE);
   glBitmap(8, 13, 0.0, 2.0, 10.0, 0.0, space);
   glEndList();
}

void init(void)
```

```
{
   glShadeModel (GL_FLAT);
   makeRasterFont();
}

void printString(char *s)
{
   glPushAttrib (GL_LIST_BIT);
   glListBase(fontOffset);
   glCallLists(strlen(s), GL_UNSIGNED_BYTE, (GLubyte *) s);
   glPopAttrib ();
}

/* Everything above this line could be in a library
 * that defines a font.  To make it work, you've got
 * to call makeRasterFont() before you start making
 * calls to printString().
 */
void display(void)
{
   GLfloat white[3] = { 1.0, 1.0, 1.0 };

   glClear(GL_COLOR_BUFFER_BIT);
   glColor3fv(white);

   glRasterPos2i(20, 60);
   printString("THE QUICK BROWN FOX JUMPS");
   glRasterPos2i(20, 40);
   printString("OVER A LAZY DOG");
   glFlush ();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho (0.0, w, 0.0, h, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
   }
}

/*  Main Loop
 *  Open window with initial window size, title bar,
 *  RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
   glutInitWindowSize(300, 100);
   glutInitWindowPosition (100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

23: Example 8-4 : Drawing, Copying, and Zooming Pixel Data: image.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

/*    Create checkerboard image     */
#define     checkImageWidth 64
#define     checkImageHeight 64
GLubyte checkImage[checkImageHeight][checkImageWidth][3];

static GLdouble zoomFactor = 1.0;
static GLint height;

void makeCheckImage(void)
{
   int i, j, c;

   for (i = 0; i < checkImageHeight; i++) {
      for (j = 0; j < checkImageWidth; j++) {
         c = ((((i&0x8)==0)^((j&0x8))==0))*255;
         checkImage[i][j][0] = (GLubyte) c;
         checkImage[i][j][1] = (GLubyte) c;
         checkImage[i][j][2] = (GLubyte) c;
      }
   }
}

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
   makeCheckImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glRasterPos2i(0, 0);
   glDrawPixels(checkImageWidth, checkImageHeight, GL_RGB,
                GL_UNSIGNED_BYTE, checkImage);
   glFlush();
}
```

```
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   height = (GLint) h;
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, (GLdouble) w, 0.0, (GLdouble) h);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void motion(int x, int y)
{
   static GLint screeny;

   screeny = height - (GLint) y;
   glRasterPos2i (x, screeny);
   glPixelZoom (zoomFactor, zoomFactor);
   glCopyPixels (0, 0, checkImageWidth, checkImageHeight, GL_COLOR);
   glPixelZoom (1.0, 1.0);
   glFlush ();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 'r':
      case 'R':
         zoomFactor = 1.0;
         glutPostRedisplay();
         printf ("zoomFactor reset to 1.0\n");
         break;
      case 'z':
         zoomFactor += 0.5;
         if (zoomFactor >= 3.0)
            zoomFactor = 3.0;
         printf ("zoomFactor is now %4.1f\n", zoomFactor);
         break;
      case 'Z':
         zoomFactor -= 0.5;
         if (zoomFactor <= 0.5)
            zoomFactor = 0.5;
         printf ("zoomFactor is now %4.1f\n", zoomFactor);
         break;
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(250, 250);
```

```
   glutInitWindowPosition(100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMotionFunc(motion);
   glutMainLoop();
   return 0;
}
```

24: Example 9-1 : Texture-Mapped Checkerboard: checker.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

/*    Create checkerboard texture   */
#define     checkImageWidth 64
#define     checkImageHeight 64
static GLubyte checkImage[checkImageHeight][checkImageWidth][4];

#ifdef GL_VERSION_1_1
static GLuint texName;
#endif

void makeCheckImage(void)
{
   int i, j, c;

   for (i = 0; i < checkImageHeight; i++) {
      for (j = 0; j < checkImageWidth; j++) {
         c = ((((i&0x8)==0)^((j&0x8))==0))*255;
         checkImage[i][j][0] = (GLubyte) c;
         checkImage[i][j][1] = (GLubyte) c;
         checkImage[i][j][2] = (GLubyte) c;
         checkImage[i][j][3] = (GLubyte) 255;
      }
   }
}

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
   glEnable(GL_DEPTH_TEST);

   makeCheckImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

#ifdef GL_VERSION_1_1
   glGenTextures(1, &texName);
   glBindTexture(GL_TEXTURE_2D, texName);
#endif

   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

```
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
#ifdef GL_VERSION_1_1
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth,
checkImageHeight,
                0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
#else
   glTexImage2D(GL_TEXTURE_2D, 0, 4, checkImageWidth, checkImageHeight,
                0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
#endif
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glEnable(GL_TEXTURE_2D);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
#ifdef GL_VERSION_1_1
   glBindTexture(GL_TEXTURE_2D, texName);
#endif

   glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
   glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
   glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);

   glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
   glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
   glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
   glEnd();
   glFlush();
   glDisable(GL_TEXTURE_2D);
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 30.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(0.0, 0.0, -3.6);
}

void keyboard (unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
      default:
         break;
   }
```

```
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(250, 250);
   glutInitWindowPosition(100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

25: Example 9-4 : Mipmap Textures: mipmap.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>

GLubyte mipmapImage32[32][32][4];
GLubyte mipmapImage16[16][16][4];
GLubyte mipmapImage8[8][8][4];
GLubyte mipmapImage4[4][4][4];
GLubyte mipmapImage2[2][2][4];
GLubyte mipmapImage1[1][1][4];

#ifdef GL_VERSION_1_1
static GLuint texName;
#endif

void makeImages(void)
{
   int i, j;

   for (i = 0; i < 32; i++) {
      for (j = 0; j < 32; j++) {
         mipmapImage32[i][j][0] = 255;
         mipmapImage32[i][j][1] = 255;
         mipmapImage32[i][j][2] = 0;
         mipmapImage32[i][j][3] = 255;
      }
   }
   for (i = 0; i < 16; i++) {
      for (j = 0; j < 16; j++) {
         mipmapImage16[i][j][0] = 255;
         mipmapImage16[i][j][1] = 0;
         mipmapImage16[i][j][2] = 255;
         mipmapImage16[i][j][3] = 255;
      }
   }
   for (i = 0; i < 8; i++) {
      for (j = 0; j < 8; j++) {
         mipmapImage8[i][j][0] = 255;
```

```
            mipmapImage8[i][j][1] = 0;
            mipmapImage8[i][j][2] = 0;
            mipmapImage8[i][j][3] = 255;
         }
      }
      for (i = 0; i < 4; i++) {
         for (j = 0; j < 4; j++) {
            mipmapImage4[i][j][0] = 0;
            mipmapImage4[i][j][1] = 255;
            mipmapImage4[i][j][2] = 0;
            mipmapImage4[i][j][3] = 255;
         }
      }
      for (i = 0; i < 2; i++) {
         for (j = 0; j < 2; j++) {
            mipmapImage2[i][j][0] = 0;
            mipmapImage2[i][j][1] = 0;
            mipmapImage2[i][j][2] = 255;
            mipmapImage2[i][j][3] = 255;
         }
      }
      mipmapImage1[0][0][0] = 255;
      mipmapImage1[0][0][1] = 255;
      mipmapImage1[0][0][2] = 255;
      mipmapImage1[0][0][3] = 255;
   }

   void init(void)
   {
      glEnable(GL_DEPTH_TEST);
      glShadeModel(GL_FLAT);

      glTranslatef(0.0, 0.0, -3.6);
      makeImages();
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

   #ifdef GL_VERSION_1_1
      glGenTextures(1, &texName);
      glBindTexture(GL_TEXTURE_2D, texName);
   #endif
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                      GL_NEAREST_MIPMAP_NEAREST);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 32, 32, 0,
                   GL_RGBA, GL_UNSIGNED_BYTE, mipmapImage32);
      glTexImage2D(GL_TEXTURE_2D, 1, GL_RGBA, 16, 16, 0,
                   GL_RGBA, GL_UNSIGNED_BYTE, mipmapImage16);
      glTexImage2D(GL_TEXTURE_2D, 2, GL_RGBA, 8, 8, 0,
                   GL_RGBA, GL_UNSIGNED_BYTE, mipmapImage8);
      glTexImage2D(GL_TEXTURE_2D, 3, GL_RGBA, 4, 4, 0,
                   GL_RGBA, GL_UNSIGNED_BYTE, mipmapImage4);
      glTexImage2D(GL_TEXTURE_2D, 4, GL_RGBA, 2, 2, 0,
                   GL_RGBA, GL_UNSIGNED_BYTE, mipmapImage2);
      glTexImage2D(GL_TEXTURE_2D, 5, GL_RGBA, 1, 1, 0,
                   GL_RGBA, GL_UNSIGNED_BYTE, mipmapImage1);
```

```
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
   glEnable(GL_TEXTURE_2D);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
#ifdef GL_VERSION_1_1
   glBindTexture(GL_TEXTURE_2D, texName);
#endif
   glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
   glTexCoord2f(0.0, 8.0); glVertex3f(-2.0, 1.0, 0.0);
   glTexCoord2f(8.0, 8.0); glVertex3f(2000.0, 1.0, -6000.0);
   glTexCoord2f(8.0, 0.0); glVertex3f(2000.0, -1.0, -6000.0);
   glEnd();
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 30000.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard (unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (50, 50);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

26: Example 9-6 : Automatic Texture-Coordinate Generation: texgen.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

#define     stripeImageWidth 32
GLubyte stripeImage[4*stripeImageWidth];

#ifdef GL_VERSION_1_1
static GLuint texName;
#endif

void makeStripeImage(void)
{
   int j;

   for (j = 0; j < stripeImageWidth; j++) {
      stripeImage[4*j] = (GLubyte) ((j<=4) ? 255 : 0);
      stripeImage[4*j+1] = (GLubyte) ((j>4) ? 255 : 0);
      stripeImage[4*j+2] = (GLubyte) 0;
      stripeImage[4*j+3] = (GLubyte) 255;
   }
}

/*  planes for texture coordinate generation  */
static GLfloat xequalzero[] = {1.0, 0.0, 0.0, 0.0};
static GLfloat slanted[] = {1.0, 1.0, 1.0, 0.0};
static GLfloat *currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   makeStripeImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

#ifdef GL_VERSION_1_1
   glGenTextures(1, &texName);
   glBindTexture(GL_TEXTURE_1D, texName);
#endif
   glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
#ifdef GL_VERSION_1_1
   glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA, stripeImageWidth, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);
#else
   glTexImage1D(GL_TEXTURE_1D, 0, 4, stripeImageWidth, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);
#endif

   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

```
   currentCoeff = xequalzero;
   currentGenMode = GL_OBJECT_LINEAR;
   currentPlane = GL_OBJECT_PLANE;
   glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
   glTexGenfv(GL_S, currentPlane, currentCoeff);

   glEnable(GL_TEXTURE_GEN_S);
   glEnable(GL_TEXTURE_1D);
   glEnable(GL_CULL_FACE);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_AUTO_NORMAL);
   glEnable(GL_NORMALIZE);
   glFrontFace(GL_CW);
   glCullFace(GL_BACK);
   glMaterialf (GL_FRONT, GL_SHININESS, 64.0);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glPushMatrix ();
   glRotatef(45.0, 0.0, 0.0, 1.0);
#ifdef GL_VERSION_1_1
   glBindTexture(GL_TEXTURE_1D, texName);
#endif
   glutSolidTeapot(2.0);
   glPopMatrix ();
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho (-3.5, 3.5, -3.5*(GLfloat)h/(GLfloat)w,
               3.5*(GLfloat)h/(GLfloat)w, -3.5, 3.5);
   else
      glOrtho (-3.5*(GLfloat)w/(GLfloat)h,
               3.5*(GLfloat)w/(GLfloat)h, -3.5, 3.5, -3.5, 3.5);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard (unsigned char key, int x, int y)
{
   switch (key) {
      case 'e':
      case 'E':
         currentGenMode = GL_EYE_LINEAR;
         currentPlane = GL_EYE_PLANE;
         glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
         glTexGenfv(GL_S, currentPlane, currentCoeff);
         glutPostRedisplay();
```

```
            break;
         case 'o':
         case 'O':
            currentGenMode = GL_OBJECT_LINEAR;
            currentPlane = GL_OBJECT_PLANE;
            glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
            glTexGenfv(GL_S, currentPlane, currentCoeff);
            glutPostRedisplay();
            break;
         case 's':
         case 'S':
            currentCoeff = slanted;
            glTexGenfv(GL_S, currentPlane, currentCoeff);
            glutPostRedisplay();
            break;
         case 'x':
         case 'X':
            currentCoeff = xequalzero;
            glTexGenfv(GL_S, currentPlane, currentCoeff);
            glutPostRedisplay();
            break;
         case 27:
            exit(0);
            break;
         default:
            break;
      }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(256, 256);
   glutInitWindowPosition(100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

27: Example 10-1 : Using the Stencil Test: stencil.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>

#define YELLOWMAT   1
#define BLUEMAT 2

void init (void)
{
   GLfloat yellow_diffuse[] = { 0.7, 0.7, 0.0, 1.0 };
   GLfloat yellow_specular[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
   GLfloat blue_diffuse[] = { 0.1, 0.1, 0.7, 1.0 };
   GLfloat blue_specular[] = { 0.1, 1.0, 1.0, 1.0 };

   GLfloat position_one[] = { 1.0, 1.0, 1.0, 0.0 };

   glNewList(YELLOWMAT, GL_COMPILE);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, yellow_diffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, yellow_specular);
   glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
   glEndList();

   glNewList(BLUEMAT, GL_COMPILE);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, blue_diffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, blue_specular);
   glMaterialf(GL_FRONT, GL_SHININESS, 45.0);
   glEndList();

   glLightfv(GL_LIGHT0, GL_POSITION, position_one);

   glEnable(GL_LIGHT0);
   glEnable(GL_LIGHTING);
   glEnable(GL_DEPTH_TEST);

   glClearStencil(0x0);
   glEnable(GL_STENCIL_TEST);
}

/* Draw a sphere in a diamond-shaped section in the
 * middle of a window with 2 torii.
 */
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

/* draw blue sphere where the stencil is 1 */
   glStencilFunc (GL_EQUAL, 0x1, 0x1);
   glStencilOp (GL_KEEP, GL_KEEP, GL_KEEP);
   glCallList (BLUEMAT);
   glutSolidSphere (0.5, 15, 15);

/* draw the tori where the stencil is not 1 */
   glStencilFunc (GL_NOTEQUAL, 0x1, 0x1);
   glPushMatrix();
      glRotatef (45.0, 0.0, 0.0, 1.0);
      glRotatef (45.0, 0.0, 1.0, 0.0);
      glCallList (YELLOWMAT);
      glutSolidTorus (0.275, 0.85, 15, 15);
      glPushMatrix();
         glRotatef (90.0, 1.0, 0.0, 0.0);
         glutSolidTorus (0.275, 0.85, 15, 15);
      glPopMatrix();
   glPopMatrix();
}

/*  Whenever the window is reshaped, redefine the
 *  coordinate system and redraw the stencil area.
```

```
 */
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);

/* create a diamond shaped stencil area */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      gluOrtho2D(-3.0, 3.0, -3.0*(GLfloat)h/(GLfloat)w,
                 3.0*(GLfloat)h/(GLfloat)w);
   else
      gluOrtho2D(-3.0*(GLfloat)w/(GLfloat)h,
                 3.0*(GLfloat)w/(GLfloat)h, -3.0, 3.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   glClear(GL_STENCIL_BUFFER_BIT);
   glStencilFunc (GL_ALWAYS, 0x1, 0x1);
   glStencilOp (GL_REPLACE, GL_REPLACE, GL_REPLACE);
   glBegin(GL_QUADS);
      glVertex2f (-1.0, 0.0);
      glVertex2f (0.0, 1.0);
      glVertex2f (1.0, 0.0);
      glVertex2f (0.0, -1.0);
   glEnd();

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(45.0, (GLfloat) w/(GLfloat) h, 3.0, 7.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(0.0, 0.0, -5.0);
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

/* Main Loop
 * Be certain to request stencil bits.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB
                        | GLUT_DEPTH | GLUT_STENCIL);
   glutInitWindowSize (400, 400);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutReshapeFunc(reshape);
```

```
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

28: Example 10-3 : Scene Antialiasing: accpersp.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <jitter.h>

#ifdef WIN32
#define near zNear
#define far zFar
#endif

#define PI_ 3.14159265358979323846

/* accFrustum()
 * The first 6 arguments are identical to the glFrustum() call.
 *
 * pixdx and pixdy are anti-alias jitter in pixels.
 * Set both equal to 0.0 for no anti-alias jitter.
 * eyedx and eyedy are depth-of field jitter in pixels.
 * Set both equal to 0.0 for no depth of field effects.
 *
 * focus is distance from eye to plane in focus.
 * focus must be greater than, but not equal to 0.0.
 *
 * Note that accFrustum() calls glTranslatef().  You will
 * probably want to insure that your ModelView matrix has been
 * initialized to identity before calling accFrustum().
 */
void accFrustum(GLdouble left, GLdouble right, GLdouble bottom,
   GLdouble top, GLdouble near, GLdouble far, GLdouble pixdx,
   GLdouble pixdy, GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
   GLdouble xwsize, ywsize;
   GLdouble dx, dy;
   GLint viewport[4];

   glGetIntegerv (GL_VIEWPORT, viewport);

   xwsize = right - left;
   ywsize = top - bottom;

   dx = -(pixdx*xwsize/(GLdouble) viewport[2] + eyedx*near/focus);
   dy = -(pixdy*ywsize/(GLdouble) viewport[3] + eyedy*near/focus);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum (left + dx, right + dx, bottom + dy, top + dy, near, far);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
```

```
   glTranslatef (-eyedx, -eyedy, 0.0);
}

/* accPerspective()
 *
 * The first 4 arguments are identical to the gluPerspective() call.
 * pixdx and pixdy are anti-alias jitter in pixels.
 * Set both equal to 0.0 for no anti-alias jitter.
 * eyedx and eyedy are depth-of field jitter in pixels.
 * Set both equal to 0.0 for no depth of field effects.
 *
 * focus is distance from eye to plane in focus.
 * focus must be greater than, but not equal to 0.0.
 *
 * Note that accPerspective() calls accFrustum().
 */
void accPerspective(GLdouble fovy, GLdouble aspect,
   GLdouble near, GLdouble far, GLdouble pixdx, GLdouble pixdy,
   GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
   GLdouble fov2,left,right,bottom,top;

   fov2 = ((fovy*PI_) / 180.0) / 2.0;

   top = near / (cos(fov2) / sin(fov2));
   bottom = -top;

   right = top * aspect;
   left = -right;

   accFrustum (left, right, bottom, top, near, far,
               pixdx, pixdy, eyedx, eyedy, focus);
}

/*  Initialize lighting and other values.
 */
void init(void)
{
   GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };
   GLfloat lm_ambient[] = { 0.2, 0.2, 0.2, 1.0 };

   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialf(GL_FRONT, GL_SHININESS, 50.0);
   glLightfv(GL_LIGHT0, GL_POSITION, light_position);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lm_ambient);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel (GL_FLAT);

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClearAccum(0.0, 0.0, 0.0, 0.0);
}
```

```
void displayObjects(void)
{
   GLfloat torus_diffuse[] = { 0.7, 0.7, 0.0, 1.0 };
   GLfloat cube_diffuse[] = { 0.0, 0.7, 0.7, 1.0 };
   GLfloat sphere_diffuse[] = { 0.7, 0.0, 0.7, 1.0 };
   GLfloat octa_diffuse[] = { 0.7, 0.4, 0.4, 1.0 };

   glPushMatrix ();
   glTranslatef (0.0, 0.0, -5.0);
   glRotatef (30.0, 1.0, 0.0, 0.0);

   glPushMatrix ();
   glTranslatef (-0.80, 0.35, 0.0);
   glRotatef (100.0, 1.0, 0.0, 0.0);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, torus_diffuse);
   glutSolidTorus (0.275, 0.85, 16, 16);
   glPopMatrix ();

   glPushMatrix ();
   glTranslatef (-0.75, -0.50, 0.0);
   glRotatef (45.0, 0.0, 0.0, 1.0);
   glRotatef (45.0, 1.0, 0.0, 0.0);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, cube_diffuse);
   glutSolidCube (1.5);
   glPopMatrix ();

   glPushMatrix ();
   glTranslatef (0.75, 0.60, 0.0);
   glRotatef (30.0, 1.0, 0.0, 0.0);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, sphere_diffuse);
   glutSolidSphere (1.0, 16, 16);
   glPopMatrix ();

   glPushMatrix ();
   glTranslatef (0.70, -0.90, 0.25);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, octa_diffuse);
   glutSolidOctahedron ();
   glPopMatrix ();

   glPopMatrix ();
}

#define ACSIZE    8

void display(void)
{
   GLint viewport[4];
   int jitter;

   glGetIntegerv (GL_VIEWPORT, viewport);

   glClear(GL_ACCUM_BUFFER_BIT);
   for (jitter = 0; jitter < ACSIZE; jitter++) {
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      accPerspective (50.0,
         (GLdouble) viewport[2]/(GLdouble) viewport[3],
```

```
            1.0, 15.0, j8[jitter].x, j8[jitter].y, 0.0, 0.0, 1.0);
         displayObjects ();
         glAccum(GL_ACCUM, 1.0/ACSIZE);
      }
      glAccum (GL_RETURN, 1.0);
      glFlush();
}

void reshape(int w, int h)
{
      glViewport(0, 0, (GLsizei) w, (GLsizei) h);
}

void keyboard(unsigned char key, int x, int y)
{
      switch (key) {
         case 27:
            exit(0);
            break;
      }
}

/*  Main Loop
 *  Be certain you request an accumulation buffer.
 */
int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB
                           | GLUT_ACCUM | GLUT_DEPTH);
      glutInitWindowSize (250, 250);
      glutInitWindowPosition (100, 100);
      glutCreateWindow (argv[0]);
      init();
      glutReshapeFunc(reshape);
      glutDisplayFunc(display);
      glutKeyboardFunc(keyboard);
      glutMainLoop();
      return 0;
}
```

29: Example 10-5 : Depth-of-Field Effect: dof.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <jitter.h>

#ifdef WIN32
#define near zNear
#define far zFar
#endif

#define PI_ 3.14159265358979323846

GLuint teapotList;
```

```
/* accFrustum()
 * The first 6 arguments are identical to the glFrustum() call.
 *
 * pixdx and pixdy are anti-alias jitter in pixels.
 * Set both equal to 0.0 for no anti-alias jitter.
 * eyedx and eyedy are depth-of field jitter in pixels.
 * Set both equal to 0.0 for no depth of field effects.
 *
 * focus is distance from eye to plane in focus.
 * focus must be greater than, but not equal to 0.0.
 *
 * Note that accFrustum() calls glTranslatef().  You will
 * probably want to insure that your ModelView matrix has been
 * initialized to identity before calling accFrustum().
 */
void accFrustum(GLdouble left, GLdouble right, GLdouble bottom,
   GLdouble top, GLdouble near, GLdouble far, GLdouble pixdx,
   GLdouble pixdy, GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
   GLdouble xwsize, ywsize;
   GLdouble dx, dy;
   GLint viewport[4];

   glGetIntegerv (GL_VIEWPORT, viewport);

   xwsize = right - left;
   ywsize = top - bottom;

   dx = -(pixdx*xwsize/(GLdouble) viewport[2] + eyedx*near/focus);
   dy = -(pixdy*ywsize/(GLdouble) viewport[3] + eyedy*near/focus);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum (left + dx, right + dx, bottom + dy, top + dy, near, far);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef (-eyedx, -eyedy, 0.0);
}

/*  accPerspective()
 *
 *  The first 4 arguments are identical to the gluPerspective() call.
 *  pixdx and pixdy are anti-alias jitter in pixels.
 *  Set both equal to 0.0 for no anti-alias jitter.
 *  eyedx and eyedy are depth-of field jitter in pixels.
 *  Set both equal to 0.0 for no depth of field effects.
 *
 *  focus is distance from eye to plane in focus.
 *  focus must be greater than, but not equal to 0.0.
 *
 *  Note that accPerspective() calls accFrustum().
 */
void accPerspective(GLdouble fovy, GLdouble aspect,
   GLdouble near, GLdouble far, GLdouble pixdx, GLdouble pixdy,
   GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
```

```
   GLdouble fov2,left,right,bottom,top;

   fov2 = ((fovy*PI_) / 180.0) / 2.0;

   top = near / (cos(fov2) / sin(fov2));
   bottom = -top;

   right = top * aspect;
   left = -right;

   accFrustum (left, right, bottom, top, near, far,
               pixdx, pixdy, eyedx, eyedy, focus);
}

void init(void)
{
   GLfloat ambient[] = { 0.0, 0.0, 0.0, 1.0 };
   GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat position[] = { 0.0, 3.0, 3.0, 0.0 };

   GLfloat lmodel_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat local_view[] = { 0.0 };

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
   glLightfv(GL_LIGHT0, GL_POSITION, position);

   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
   glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

   glFrontFace (GL_CW);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_AUTO_NORMAL);
   glEnable(GL_NORMALIZE);
   glEnable(GL_DEPTH_TEST);

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClearAccum(0.0, 0.0, 0.0, 0.0);
/*  make teapot display list */
   teapotList = glGenLists(1);
   glNewList (teapotList, GL_COMPILE);
   glutSolidTeapot (0.5);
   glEndList ();
}

void renderTeapot (GLfloat x, GLfloat y, GLfloat z,
   GLfloat ambr, GLfloat ambg, GLfloat ambb,
   GLfloat difr, GLfloat difg, GLfloat difb,
   GLfloat specr, GLfloat specg, GLfloat specb, GLfloat shine)
{
   GLfloat mat[4];

   glPushMatrix();
   glTranslatef (x, y, z);
   mat[0] = ambr; mat[1] = ambg; mat[2] = ambb; mat[3] = 1.0;
```

```
   glMaterialfv (GL_FRONT, GL_AMBIENT, mat);
   mat[0] = difr; mat[1] = difg; mat[2] = difb;
   glMaterialfv (GL_FRONT, GL_DIFFUSE, mat);
   mat[0] = specr; mat[1] = specg; mat[2] = specb;
   glMaterialfv (GL_FRONT, GL_SPECULAR, mat);
   glMaterialf (GL_FRONT, GL_SHININESS, shine*128.0);
   glCallList(teapotList);
   glPopMatrix();
}

/*  display() draws 5 teapots into the accumulation buffer
 *  several times; each time with a jittered perspective.
 *  The focal point is at z = 5.0, so the gold teapot will
 *  stay in focus.  The amount of jitter is adjusted by the
 *  magnitude of the accPerspective() jitter; in this example, 0.33.
 *  In this example, the teapots are drawn 8 times.  See jitter.h
 */
void display(void)
{
   int jitter;
   GLint viewport[4];

   glGetIntegerv (GL_VIEWPORT, viewport);
   glClear(GL_ACCUM_BUFFER_BIT);

   for (jitter = 0; jitter < 8; jitter++) {
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      accPerspective (45.0,
         (GLdouble) viewport[2]/(GLdouble) viewport[3],
         1.0, 15.0, 0.0, 0.0,
         0.33*j8[jitter].x, 0.33*j8[jitter].y, 5.0);

/*    ruby, gold, silver, emerald, and cyan teapots   */
      renderTeapot (-1.1, -0.5, -4.5, 0.1745, 0.01175,
                    0.01175, 0.61424, 0.04136, 0.04136,
                    0.727811, 0.626959, 0.626959, 0.6);
      renderTeapot (-0.5, -0.5, -5.0, 0.24725, 0.1995,
                    0.0745, 0.75164, 0.60648, 0.22648,
                    0.628281, 0.555802, 0.366065, 0.4);
      renderTeapot (0.2, -0.5, -5.5, 0.19225, 0.19225,
                    0.19225, 0.50754, 0.50754, 0.50754,
                    0.508273, 0.508273, 0.508273, 0.4);
      renderTeapot (1.0, -0.5, -6.0, 0.0215, 0.1745, 0.0215,
                    0.07568, 0.61424, 0.07568, 0.633,
                    0.727811, 0.633, 0.6);
      renderTeapot (1.8, -0.5, -6.5, 0.0, 0.1, 0.06, 0.0,
                    0.50980392, 0.50980392, 0.50196078,
                    0.50196078, 0.50196078, .25);
      glAccum (GL_ACCUM, 0.125);
   }
   glAccum (GL_RETURN, 1.0);
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
```

```
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

/*  Main Loop
 *  Be certain you request an accumulation buffer.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB
                  | GLUT_ACCUM | GLUT_DEPTH);
   glutInitWindowSize (400, 400);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

30: Example 11-4 : Quadrics Objects: quadric.c

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>

#ifndef CALLBACK
#define CALLBACK
#endif

GLuint startList;

void CALLBACK errorCallback(GLenum errorCode)
{
   const GLubyte *estring;

   estring = gluErrorString(errorCode);
   fprintf(stderr, "Quadric Error: %s\n", estring);
   exit(0);
}

void init(void)
{
   GLUquadricObj *qobj;
   GLfloat mat_ambient[] = { 0.5, 0.5, 0.5, 1.0 };
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
   GLfloat mat_shininess[] = { 50.0 };
   GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
   GLfloat model_ambient[] = { 0.5, 0.5, 0.5, 1.0 };

   glClearColor(0.0, 0.0, 0.0, 0.0);

   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
   glLightfv(GL_LIGHT0, GL_POSITION, light_position);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, model_ambient);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);

/*  Create 4 display lists, each with a different quadric object.
 *  Different drawing styles and surface normal specifications
 *  are demonstrated.
 */
   startList = glGenLists(4);
   qobj = gluNewQuadric();
   gluQuadricCallback(qobj, GLU_ERROR,
                      errorCallback);

   gluQuadricDrawStyle(qobj, GLU_FILL); /* smooth shaded */
   gluQuadricNormals(qobj, GLU_SMOOTH);
   glNewList(startList, GL_COMPILE);
      gluSphere(qobj, 0.75, 15, 10);
   glEndList();

   gluQuadricDrawStyle(qobj, GLU_FILL); /* flat shaded */
   gluQuadricNormals(qobj, GLU_FLAT);
   glNewList(startList+1, GL_COMPILE);
      gluCylinder(qobj, 0.5, 0.3, 1.0, 15, 5);
   glEndList();

   gluQuadricDrawStyle(qobj, GLU_LINE); /* all polygons wireframe */
   gluQuadricNormals(qobj, GLU_NONE);
   glNewList(startList+2, GL_COMPILE);
      gluDisk(qobj, 0.25, 1.0, 20, 4);
   glEndList();

   gluQuadricDrawStyle(qobj, GLU_SILHOUETTE); /* boundary only  */
   gluQuadricNormals(qobj, GLU_NONE);
   glNewList(startList+3, GL_COMPILE);
      gluPartialDisk(qobj, 0.0, 1.0, 20, 4, 0.0, 225.0);
   glEndList();
}

void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();

   glEnable(GL_LIGHTING);
   glShadeModel (GL_SMOOTH);
```

```
   glTranslatef(-1.0, -1.0, 0.0);
   glCallList(startList);

   glShadeModel (GL_FLAT);
   glTranslatef(0.0, 2.0, 0.0);
   glPushMatrix();
   glRotatef(300.0, 1.0, 0.0, 0.0);
   glCallList(startList+1);
   glPopMatrix();

   glDisable(GL_LIGHTING);
   glColor3f(0.0, 1.0, 1.0);
   glTranslatef(2.0, -2.0, 0.0);
   glCallList(startList+2);

   glColor3f(1.0, 1.0, 0.0);
   glTranslatef(0.0, 2.0, 0.0);
   glCallList(startList+3);

   glPopMatrix();
   glFlush();
}

void reshape (int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho(-2.5, 2.5, -2.5*(GLfloat)h/(GLfloat)w,
         2.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
   else
      glOrtho(-2.5*(GLfloat)w/(GLfloat)h,
         2.5*(GLfloat)w/(GLfloat)h, -2.5, 2.5, -10.0, 10.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
```

```
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

31: Example 12-1 : Bézier Curve with Four Control Points: bezcurve.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>

GLfloat ctrlpoints[4][3] = {
      { -4.0, -4.0, 0.0}, { -2.0, 4.0, 0.0},
      {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};

void init(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
   glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpoints[0][0]);
   glEnable(GL_MAP1_VERTEX_3);
}

void display(void)
{
   int i;

   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glBegin(GL_LINE_STRIP);
      for (i = 0; i <= 30; i++)
         glEvalCoord1f((GLfloat) i/30.0);
   glEnd();
   /* The following code displays the control points as dots. */
   glPointSize(5.0);
   glColor3f(1.0, 1.0, 0.0);
   glBegin(GL_POINTS);
      for (i = 0; i < 4; i++)
         glVertex3fv(&ctrlpoints[i][0]);
   glEnd();
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
               5.0*(GLfloat)h/(GLfloat)w, -5.0, 5.0);
   else
      glOrtho(-5.0*(GLfloat)w/(GLfloat)h,
               5.0*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

```
void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc (keyboard);
   glutMainLoop();
   return 0;
}
```

32: Example 12-2 : Bézier Surface: bezsurf.c

```
#include <windows.h>
#include <stdlib.h>
#include <GL/glut.h>

GLfloat ctrlpoints[4][4][3] = {
   {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
    {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
   {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
    {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
   {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},
    {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
   {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},
    {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};

void display(void)
{
   int i, j;

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glPushMatrix ();
   glRotatef(85.0, 1.0, 1.0, 1.0);
   for (j = 0; j <= 8; j++) {
      glBegin(GL_LINE_STRIP);
      for (i = 0; i <= 30; i++)
         glEvalCoord2f((GLfloat)i/30.0, (GLfloat)j/8.0);
      glEnd();
      glBegin(GL_LINE_STRIP);
      for (i = 0; i <= 30; i++)
```

```
            glEvalCoord2f((GLfloat)j/8.0, (GLfloat)i/30.0);
         glEnd();
      }
      glPopMatrix ();
      glFlush();
}

void init(void)
{
      glClearColor (0.0, 0.0, 0.0, 0.0);
      glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
              0, 1, 12, 4, &ctrlpoints[0][0][0]);
      glEnable(GL_MAP2_VERTEX_3);
      glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
      glEnable(GL_DEPTH_TEST);
      glShadeModel(GL_FLAT);
}

void reshape(int w, int h)
{
      glViewport(0, 0, (GLsizei) w, (GLsizei) h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      if (w <= h)
         glOrtho(-4.0, 4.0, -4.0*(GLfloat)h/(GLfloat)w,
                 4.0*(GLfloat)h/(GLfloat)w, -4.0, 4.0);
      else
         glOrtho(-4.0*(GLfloat)w/(GLfloat)h,
                 4.0*(GLfloat)w/(GLfloat)h, -4.0, 4.0, -4.0, 4.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
      switch (key) {
         case 27:
            exit(0);
            break;
      }
}

int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
      glutInitWindowSize (500, 500);
      glutInitWindowPosition (100, 100);
      glutCreateWindow (argv[0]);
      init ();
      glutDisplayFunc(display);
      glutReshapeFunc(reshape);
      glutKeyboardFunc(keyboard);
      glutMainLoop();
      return 0;
}
```

33: Example 12-4 : Using Evaluators for Textures: texturesurf.c

```
#include <windows.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>

GLfloat ctrlpoints[4][4][3] = {
   {{ -1.5, -1.5, 4.0}, { -0.5, -1.5, 2.0},
    {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
   {{ -1.5, -0.5, 1.0}, { -0.5, -0.5, 3.0},
    {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
   {{ -1.5, 0.5, 4.0}, { -0.5, 0.5, 0.0},
    {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
   {{ -1.5, 1.5, -2.0}, { -0.5, 1.5, -2.0},
    {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};

GLfloat texpts[2][2][2] = {{{0.0, 0.0}, {0.0, 1.0}},
                  {{1.0, 0.0}, {1.0, 1.0}}};

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glEvalMesh2(GL_FILL, 0, 20, 0, 20);
   glFlush();
}

#define     imageWidth 64
#define     imageHeight 64
GLubyte image[3*imageWidth*imageHeight];

void makeImage(void)
{
   int i, j;
   float ti, tj;

   for (i = 0; i < imageWidth; i++) {
      ti = 2.0*3.14159265*i/imageWidth;
      for (j = 0; j < imageHeight; j++) {
         tj = 2.0*3.14159265*j/imageHeight;

         image[3*(imageHeight*i+j)] = (GLubyte) 127*(1.0+sin(ti));
         image[3*(imageHeight*i+j)+1] = (GLubyte) 127*(1.0+cos(2*tj));
         image[3*(imageHeight*i+j)+2] = (GLubyte) 127*(1.0+cos(ti+tj));
      }
   }
}

void init(void)
{
   glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
           0, 1, 12, 4, &ctrlpoints[0][0][0]);
   glMap2f(GL_MAP2_TEXTURE_COORD_2, 0, 1, 2, 2,
           0, 1, 4, 2, &texpts[0][0][0]);
   glEnable(GL_MAP2_TEXTURE_COORD_2);
```

```
   glEnable(GL_MAP2_VERTEX_3);
   glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
   makeImage();
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight, 0,
                GL_RGB, GL_UNSIGNED_BYTE, image);
   glEnable(GL_TEXTURE_2D);
   glEnable(GL_DEPTH_TEST);
   glShadeModel (GL_FLAT);
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho(-4.0, 4.0, -4.0*(GLfloat)h/(GLfloat)w,
              4.0*(GLfloat)h/(GLfloat)w, -4.0, 4.0);
   else
      glOrtho(-4.0*(GLfloat)w/(GLfloat)h,
              4.0*(GLfloat)w/(GLfloat)h, -4.0, 4.0, -4.0, 4.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glRotatef(85.0, 1.0, 1.0, 1.0);
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

34: Example 12-5 : NURBS ((Non-Uniform Rational B-Spline) Surface: surface.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

#ifndef CALLBACK
#define CALLBACK
#endif

GLfloat ctlpoints[4][4][3];
int showPoints = 0;

GLUnurbsObj *theNurb;

/*
 *  Initializes the control points of the surface to a small hill.
 *  The control points range from -3 to +3 in x, y, and z
 */
void init_surface(void)
{
   int u, v;
   for (u = 0; u < 4; u++) {
      for (v = 0; v < 4; v++) {
         ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
         ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);

         if ( (u == 1 || u == 2) && (v == 1 || v == 2))
            ctlpoints[u][v][2] = 3.0;
         else
            ctlpoints[u][v][2] = -3.0;
      }
   }
}

void CALLBACK nurbsError(GLenum errorCode)
{
   const GLubyte *estring;

   estring = gluErrorString(errorCode);
   fprintf (stderr, "Nurbs Error: %s\n", estring);
   exit (0);
}

/*  Initialize material property and depth buffer.
 */
void init(void)
{
   GLfloat mat_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat mat_shininess[] = { 100.0 };

   glClearColor (0.0, 0.0, 0.0, 0.0);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

   glEnable(GL_LIGHTING);
```

```
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_AUTO_NORMAL);
   glEnable(GL_NORMALIZE);

   init_surface();

   theNurb = gluNewNurbsRenderer();
   gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 25.0);
   gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
   //gluNurbsCallback(theNurb, GLU_ERROR, nurbsError); MEMBUAT ERROR
C2664 DONTKNOWWHY?
}

void display(void)
{
   GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};
   int i, j;

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glPushMatrix();
   glRotatef(330.0, 1.,0.,0.);
   glScalef (0.5, 0.5, 0.5);

   gluBeginSurface(theNurb);
   gluNurbsSurface(theNurb,
                   8, knots, 8, knots,
                   4 * 3, 3, &ctlpoints[0][0][0],
                   4, 4, GL_MAP2_VERTEX_3);
   gluEndSurface(theNurb);

   if (showPoints) {
      glPointSize(5.0);
      glDisable(GL_LIGHTING);
      glColor3f(1.0, 1.0, 0.0);
      glBegin(GL_POINTS);
      for (i = 0; i < 4; i++) {
         for (j = 0; j < 4; j++) {
          glVertex3f(ctlpoints[i][j][0],
               ctlpoints[i][j][1], ctlpoints[i][j][2]);
         }
      }
      glEnd();
      glEnable(GL_LIGHTING);
   }
   glPopMatrix();
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective (45.0, (GLdouble)w/(GLdouble)h, 3.0, 8.0);
   glMatrixMode(GL_MODELVIEW);
```

```
   glLoadIdentity();
   glTranslatef (0.0, 0.0, -5.0);
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 'c':
      case 'C':
         showPoints = !showPoints;
         glutPostRedisplay();
         break;
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
   glutKeyboardFunc (keyboard);
   glutMainLoop();
   return 0;
}
```

35: Example 13-2 : Selection Example: select.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

/* draw a triangle with vertices at (x1, y1), (x2, y2)
 * and (x3, y3) at z units away from the origin.
 */
void drawTriangle (GLfloat x1, GLfloat y1, GLfloat x2,
    GLfloat y2, GLfloat x3, GLfloat y3, GLfloat z)
{
   glBegin (GL_TRIANGLES);
   glVertex3f (x1, y1, z);
   glVertex3f (x2, y2, z);
   glVertex3f (x3, y3, z);
   glEnd ();
}

/* draw a rectangular box with these outer x, y, and z values */
void drawViewVolume (GLfloat x1, GLfloat x2, GLfloat y1,
```

```
                    GLfloat y2, GLfloat z1, GLfloat z2)
{
   glColor3f (1.0, 1.0, 1.0);
   glBegin (GL_LINE_LOOP);
   glVertex3f (x1, y1, -z1);
   glVertex3f (x2, y1, -z1);
   glVertex3f (x2, y2, -z1);
   glVertex3f (x1, y2, -z1);
   glEnd ();

   glBegin (GL_LINE_LOOP);
   glVertex3f (x1, y1, -z2);
   glVertex3f (x2, y1, -z2);
   glVertex3f (x2, y2, -z2);
   glVertex3f (x1, y2, -z2);
   glEnd ();

   glBegin (GL_LINES);  /*  4 lines */
   glVertex3f (x1, y1, -z1);
   glVertex3f (x1, y1, -z2);
   glVertex3f (x1, y2, -z1);
   glVertex3f (x1, y2, -z2);
   glVertex3f (x2, y1, -z1);
   glVertex3f (x2, y1, -z2);
   glVertex3f (x2, y2, -z1);
   glVertex3f (x2, y2, -z2);
   glEnd ();
}

/* drawScene draws 4 triangles and a wire frame
 * which represents the viewing volume.
 */
void drawScene (void)
{
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluPerspective (40.0, 4.0/3.0, 1.0, 100.0);

   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
   gluLookAt (7.5, 7.5, 12.5, 2.5, 2.5, -5.0, 0.0, 1.0, 0.0);
   glColor3f (0.0, 1.0, 0.0); /*  green triangle      */
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -5.0);
   glColor3f (1.0, 0.0, 0.0); /*  red triangle  */
   drawTriangle (2.0, 7.0, 3.0, 7.0, 2.5, 8.0, -5.0);
   glColor3f (1.0, 1.0, 0.0); /*  yellow triangles    */
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, 0.0);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -10.0);
   drawViewVolume (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);
}

/* processHits prints out the contents of the selection array
 */
void processHits (GLint hits, GLuint buffer[])
{
   unsigned int i, j;
   GLuint names, *ptr;
```

```
   printf ("hits = %d\n", hits);
   ptr = (GLuint *) buffer;
   for (i = 0; i < hits; i++) {     /*  for each hit  */
      names = *ptr;
      printf (" number of names for hit = %d\n", names); ptr++;
      printf("  z1 is %g;", (float) *ptr/0x7fffffff); ptr++;
      printf(" z2 is %g\n", (float) *ptr/0x7fffffff); ptr++;
      printf ("   the name is ");
      for (j = 0; j < names; j++) { /*  for each name */
         printf ("%d ", *ptr); ptr++;
      }
      printf ("\n");
   }
}

/* selectObjects "draws" the triangles in selection mode,
 * assigning names for the triangles.  Note that the third
 * and fourth triangles share one name, so that if either
 * or both triangles intersects the viewing/clipping volume,
 * only one hit will be registered.
 */
#define BUFSIZE 512

void selectObjects(void)
{
   GLuint selectBuf[BUFSIZE];
   GLint hits;

   glSelectBuffer (BUFSIZE, selectBuf);
   (void) glRenderMode (GL_SELECT);

   glInitNames();
   glPushName(0);

   glPushMatrix ();
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
   glLoadName(1);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -5.0);
   glLoadName(2);
   drawTriangle (2.0, 7.0, 3.0, 7.0, 2.5, 8.0, -5.0);
   glLoadName(3);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, 0.0);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -10.0);
   glPopMatrix ();
   glFlush ();

   hits = glRenderMode (GL_RENDER);
   processHits (hits, selectBuf);
}

void init (void)
{
```

```
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_FLAT);
}

void display(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   drawScene ();
   selectObjects ();
   glFlush();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

/*  Main Loop  */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (200, 200);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

36: Example 13-3 : Picking Example: picksquare.c

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>

int board[3][3];   /*  amount of color for each square      */

/*  Clear color value for every square on the board   */
void init(void)
{
   int i, j;
   for (i = 0; i < 3; i++)
      for (j = 0; j < 3; j ++)
         board[i][j] = 0;
   glClearColor (0.0, 0.0, 0.0, 0.0);
}

/*  The nine squares are drawn.  In selection mode, each
```

```
 *  square is given two names:  one for the row and the
 *  other for the column on the grid.  The color of each
 *  square is determined by its position on the grid, and
 *  the value in the board[][] array.
 */
void drawSquares(GLenum mode)
{
   GLuint i, j;
   for (i = 0; i < 3; i++) {
      if (mode == GL_SELECT)
         glLoadName (i);
      for (j = 0; j < 3; j ++) {
         if (mode == GL_SELECT)
            glPushName (j);
         glColor3f ((GLfloat) i/3.0, (GLfloat) j/3.0,
                    (GLfloat) board[i][j]/3.0);
         glRecti (i, j, i+1, j+1);
         if (mode == GL_SELECT)
            glPopName ();
      }
   }
}

/*  processHits prints out the contents of the
 *  selection array.
 */
void processHits (GLint hits, GLuint buffer[])
{
   unsigned int i, j;
   GLuint ii, jj, names, *ptr;

   printf ("hits = %d\n", hits);
   ptr = (GLuint *) buffer;
   for (i = 0; i < hits; i++) {     /*  for each hit  */
      names = *ptr;
      printf (" number of names for this hit = %d\n", names); ptr++;
      printf("  z1 is %g;", (float) *ptr/0x7fffffff); ptr++;
      printf(" z2 is %g\n", (float) *ptr/0x7fffffff); ptr++;
      printf ("   names are ");
      for (j = 0; j < names; j++) { /*  for each name */
         printf ("%d ", *ptr);
         if (j == 0)  /*  set row and column  */
            ii = *ptr;
         else if (j == 1)
            jj = *ptr;
         ptr++;
      }
      printf ("\n");
      board[ii][jj] = (board[ii][jj] + 1) % 3;
   }
}

/*  pickSquares() sets up selection mode, name stack,
 *  and projection matrix for picking.  Then the
 *  objects are drawn.
 */
#define BUFSIZE 512
```

```
void pickSquares(int button, int state, int x, int y)
{
   GLuint selectBuf[BUFSIZE];
   GLint hits;
   GLint viewport[4];

   if (button != GLUT_LEFT_BUTTON || state != GLUT_DOWN)
      return;

   glGetIntegerv (GL_VIEWPORT, viewport);

   glSelectBuffer (BUFSIZE, selectBuf);
   (void) glRenderMode (GL_SELECT);

   glInitNames();
   glPushName(0);

   glMatrixMode (GL_PROJECTION);
   glPushMatrix ();
   glLoadIdentity ();
/*  create 5x5 pixel picking region near cursor location    */
   gluPickMatrix ((GLdouble) x, (GLdouble) (viewport[3] - y),
                  5.0, 5.0, viewport);
   gluOrtho2D (0.0, 3.0, 0.0, 3.0);
   drawSquares (GL_SELECT);

   glMatrixMode (GL_PROJECTION);
   glPopMatrix ();
   glFlush ();

   hits = glRenderMode (GL_RENDER);
   processHits (hits, selectBuf);
   glutPostRedisplay();
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   drawSquares (GL_RENDER);
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D (0.0, 3.0, 0.0, 3.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
```

```
         exit(0);
         break;
   }
}

/* Main Loop */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (100, 100);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutReshapeFunc (reshape);
   glutDisplayFunc(display);
   glutMouseFunc (pickSquares);
   glutKeyboardFunc (keyboard);
   glutMainLoop();
   return 0;
}
```

37: Example 13-6 : Picking with Depth Values: pickdepth.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

void init(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_FLAT);
   glDepthRange(0.0, 1.0);  /* The default z mapping */
}

/* The three rectangles are drawn.  In selection mode,
 * each rectangle is given the same name.  Note that
 * each rectangle is drawn with a different z value.
 */
void drawRects(GLenum mode)
{
   if (mode == GL_SELECT)
      glLoadName(1);
   glBegin(GL_QUADS);
   glColor3f(1.0, 1.0, 0.0);
   glVertex3i(2, 0, 0);
   glVertex3i(2, 6, 0);
   glVertex3i(6, 6, 0);
   glVertex3i(6, 0, 0);
   glEnd();
   if (mode == GL_SELECT)
      glLoadName(2);
   glBegin(GL_QUADS);
   glColor3f(0.0, 1.0, 1.0);
   glVertex3i(3, 2, -1);
```

```
   glVertex3i(3, 8, -1);
   glVertex3i(8, 8, -1);
   glVertex3i(8, 2, -1);
   glEnd();
   if (mode == GL_SELECT)
      glLoadName(3);
   glBegin(GL_QUADS);
   glColor3f(1.0, 0.0, 1.0);
   glVertex3i(0, 2, -2);
   glVertex3i(0, 7, -2);
   glVertex3i(5, 7, -2);
   glVertex3i(5, 2, -2);
   glEnd();
}

/*  processHits() prints out the contents of the
 *  selection array.
 */
void processHits(GLint hits, GLuint buffer[])
{
   unsigned int i, j;
   GLuint names, *ptr;

   printf("hits = %d\n", hits);
   ptr = (GLuint *) buffer;
   for (i = 0; i < hits; i++) {  /* for each hit  */
      names = *ptr;
      printf(" number of names for hit = %d\n", names); ptr++;
      printf("  z1 is %g;", (float) *ptr/0x7fffffff); ptr++;
      printf(" z2 is %g\n", (float) *ptr/0x7fffffff); ptr++;
      printf("   the name is ");
      for (j = 0; j < names; j++) {  /* for each name */
         printf("%d ", *ptr); ptr++;
      }
      printf("\n");
   }
}

/*  pickRects() sets up selection mode, name stack,
 *  and projection matrix for picking.  Then the objects
 *  are drawn.
 */
#define BUFSIZE 512

void pickRects(int button, int state, int x, int y)
{
   GLuint selectBuf[BUFSIZE];
   GLint hits;
   GLint viewport[4];

   if (button != GLUT_LEFT_BUTTON || state != GLUT_DOWN)
      return;

   glGetIntegerv(GL_VIEWPORT, viewport);

   glSelectBuffer(BUFSIZE, selectBuf);
   (void) glRenderMode(GL_SELECT);
```

```
   glInitNames();
   glPushName(0);

   glMatrixMode(GL_PROJECTION);
   glPushMatrix();
   glLoadIdentity();
/*  create 5x5 pixel picking region near cursor location */
   gluPickMatrix((GLdouble) x, (GLdouble) (viewport[3] - y),
                 5.0, 5.0, viewport);
   glOrtho(0.0, 8.0, 0.0, 8.0, -0.5, 2.5);
   drawRects(GL_SELECT);
   glPopMatrix();
   glFlush();

   hits = glRenderMode(GL_RENDER);
   processHits(hits, selectBuf);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   drawRects(GL_RENDER);
   glFlush();
}

void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 8.0, 0.0, 8.0, -0.5, 2.5);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

/*  Main Loop
 */
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (200, 200);
   glutInitWindowPosition (100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
```

```
   glutMouseFunc(pickRects);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

38: Example 13-7 : Feedback Mode: feedback.c

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

/*  Initialize lighting.
 */
void init(void)
{
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
}

/* Draw a few lines and two points, one of which will
 * be clipped.  If in feedback mode, a passthrough token
 * is issued between the each primitive.
 */
void drawGeometry (GLenum mode)
{
   glBegin (GL_LINE_STRIP);
   glNormal3f (0.0, 0.0, 1.0);
   glVertex3f (30.0, 30.0, 0.0);
   glVertex3f (50.0, 60.0, 0.0);
   glVertex3f (70.0, 40.0, 0.0);
   glEnd ();
   if (mode == GL_FEEDBACK)
      glPassThrough (1.0);
   glBegin (GL_POINTS);
   glVertex3f (-100.0, -100.0, -100.0);  /*  will be clipped  */
   glEnd ();
   if (mode == GL_FEEDBACK)
      glPassThrough (2.0);
   glBegin (GL_POINTS);
   glNormal3f (0.0, 0.0, 1.0);
   glVertex3f (50.0, 50.0, 0.0);
   glEnd ();
}

/* Write contents of one vertex to stdout.      */
void print3DcolorVertex (GLint size, GLint *count,
                         GLfloat *buffer)
{
   int i;

   printf ("  ");
   for (i = 0; i < 7; i++) {
      printf ("%4.2f ", buffer[size-(*count)]);
      *count = *count - 1;
   }
```

```
   printf ("\n");
}

/*  Write contents of entire buffer.  (Parse tokens!) */
void printBuffer(GLint size, GLfloat *buffer)
{
   GLint count;
   GLfloat token;

   count = size;
   while (count) {
      token = buffer[size-count]; count--;
      if (token == GL_PASS_THROUGH_TOKEN) {
         printf ("GL_PASS_THROUGH_TOKEN\n");
         printf ("  %4.2f\n", buffer[size-count]);
         count--;
      }
      else if (token == GL_POINT_TOKEN) {
         printf ("GL_POINT_TOKEN\n");
         print3DcolorVertex (size, &count, buffer);
      }
      else if (token == GL_LINE_TOKEN) {
         printf ("GL_LINE_TOKEN\n");
         print3DcolorVertex (size, &count, buffer);
         print3DcolorVertex (size, &count, buffer);
      }
      else if (token == GL_LINE_RESET_TOKEN) {
         printf ("GL_LINE_RESET_TOKEN\n");
         print3DcolorVertex (size, &count, buffer);
         print3DcolorVertex (size, &count, buffer);
      }
   }
}

void display(void)
{
   GLfloat feedBuffer[1024];
   GLint size;

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho (0.0, 100.0, 0.0, 100.0, 0.0, 1.0);

   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   drawGeometry (GL_RENDER);

   glFeedbackBuffer (1024, GL_3D_COLOR, feedBuffer);
   (void) glRenderMode (GL_FEEDBACK);
   drawGeometry (GL_FEEDBACK);

   size = glRenderMode (GL_RENDER);
   printBuffer (size, feedBuffer);
}

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

```
   switch (key) {
      case 27:
         exit(0);
         break;
   }
}

/*  Main Loop  */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (100, 100);
   glutInitWindowPosition (100, 100);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc (keyboard);
   glutMainLoop();
   return 0;
}
```