Praktikum 7
Tekstur dengan Gambar

Tekstur adalah tampilan permukaan (corak) dari suatu benda yang dapat dinilai dengan cara dilihat atau diraba. Pada prakteknya, tekstur sering dikategorikan sebagai corak dari suatu permukaan benda, misalnya permukaan karpet, baju, kulit kayu, dan lain sebagainya. Tekstur merupakan karakteristik intrinsik dari suatu citra yang terkait dengan tingkat kekasaran (*roughness*), granularitas (*granulation*), dan keteraturan (*regularity*) susunan struktural piksel. Aspek tekstural dari sebuah citra dapat dimanfaatkan sebagai dasar dari segmentasi, klasifikasi, maupun interpretasi citra. Tekstur dapat didefinisikan sebagai fungsi dari variasi spasial intensitas piksel (nilai keabuan) dalam citra.

Objek 3D pada open GL merupakan objek yang lebih hidup dibandingkan objek 2D. Namun permukaan objek 3D yang polos membuat 3D cenderung kurang menarik dan kaku. Untuk membuat objek yang lebih hidup pada OpenGL diperlukan suatu texture mapping. Mapping ialah sebuah bentuk kegiatan untuk melakukan pewarnaan atau memetakan permukaan geometri pada objek 3D. Sedangkan Maps adalah bentuk gambar atau warna yang digunakan untuk melapisi objek 3D pada saat dilakukan mapping. Dengan kata lain pemetaan texture merupakan pemberian sebuah gambar pada permukaan objek sehingga objek akan tampak realistis. Texture mapping memungkinkan untuk menaruh gambar pada geometric primitive tersebut dan sekaligus mengikuti transformasi yang diterapkan kepada objek. Contohnya apabila sebuah objek kubus tanpa gambar diberi texture bebatuan pada permukaannya, maka objek tersebut akan tampak memiliki tekstur kasar seperti batu.

Untuk membuat suatu tekstur pada suatu object dapat memanfaatkan sebuah gambar sebagai alat bantu pembentuk tekstur.

Imageloader.cpp

```
#include <assert.h>
#include <fstream>
#include "imageloader.h"
using namespace std;
Image::Image(char* ps, int w, int h) : pixels(ps), width(w),
height(h) {}
Image::~Image() {
  delete[] pixels;
}
namespace {
  //Konversi 4 buah karakter ke integer,
  //menggunakan bentuk little-endian
```

```
int toInt(const char* bytes) {
  return (int)(((unsigned char)bytes[3] << 24) |
               ((unsigned char)bytes[2] << 16) |
               ((unsigned char)bytes[1] << 8) |
               (unsigned char)bytes[0]);
    }
//Konversi 2 buah karakter ke integer,
//menggunakan bentuk little-endian
    short toShort(const char* bytes) {
      return (short)(((unsigned char)bytes[1] << 8) |
                     (unsigned char)bytes[0]);
    }
//Membaca 4 byte selanjutnya sebagai integer,
//menggunakan bentuk little-endian
int readInt(ifstream &input) {
    char buffer[4];
    input.read(buffer, 4);
    return toInt(buffer);
}
short readShort(ifstream &input) {
    char buffer[2];
    input.read(buffer, 2);
    return toShort(buffer);
}
template<class T>
class auto_array {
private:
    T* array;
    mutable bool isReleased;
public:
    explicit auto_array(T* array_ = NULL) :
    array(array_), isReleased(false) {}
    auto_array(const auto_array<T> &aarray) {
          array = aarray.array;
          isReleased = aarray.isReleased;
          aarray.isReleased = true;
    }
    ~auto_array() {
    if (!isReleased && array != NULL) {
          delete[] array;
    }
}
    T* get() const {
          return array;
    }
    T &operator*() const {
          return *array;
    }
    void operator=(const auto_array<T> &aarray) {
          if (!isReleased && array != NULL) {
                delete[] array;
          }
          array = aarray.array;
          isReleased = aarray.isReleased;
          aarray.isReleased = true;
```

```
	}
	T* operator->() const {
		return array;
	}
	T* release() {
		isReleased = true;
		return array;
	}
	void reset(T* array_ = NULL) {
		if (!isReleased && array != NULL) {
			delete[] array;
		}
		array = array_;
	}
			T* operator+(int i) {
				return array + i;
			}

			T &operator[](int i) {
				return array[i];
			}
	};
}

Image* loadBMP(const char* filename) {
	ifstream input;
	input.open(filename, ifstream::binary);
	assert(!input.fail() || !"File tidak ditemukan!!!");
	char buffer[2];
	input.read(buffer, 2);
	assert(buffer[0] == 'B' && buffer[1] == 'M' || !"Bukan file
bitmap!!!");
	input.ignore(8);
	int dataOffset = readInt(input);

	int headerSize = readInt(input);
	int width;
	int height;
	switch(headerSize) {
		case 40:
			width = readInt(input);
			height = readInt(input);
			input.ignore(2);
			assert(readShort(input) == 24 || !"Gambar tidak 24
bits per pixel!");
			assert(readShort(input)    ==    0    ||    !"Gambar
dikompres!");
			break;
		case 12:
			width = readShort(input);
			height = readShort(input);
			input.ignore(2);
			assert(readShort(input) == 24 || !"Gambar tidak 24
bits per pixel!");
			break;
```

```
            case 64:
                  assert(!"Tidak dapat mengambil OS/2 V2 bitmaps");
                  break;
            case 108:
                  assert(!"Tidak dapat mengambil Windows V4 bitmaps");
                  break;
            case 124:
                  assert(!"Tidak dapat mengambil Windows V5 bitmaps");
                  break;
            default:
                  assert(!"Format bitmap ini tidak diketahui!");
      }

      //Membaca data
      int bytesPerRow = ((width * 3 + 3) / 4) * 4 - (width * 3 % 4);
      int size = bytesPerRow * height;
      auto_array<char> pixels(new char[size]);
      input.seekg(dataOffset, ios_base::beg);
      input.read(pixels.get(), size);

      //Mengambil data yang mempunyai format benar
      auto_array<char> pixels2(new char[width * height * 3]);
      for(int y = 0; y < height; y++) {
            for(int x = 0; x < width; x++) {
                  for(int c = 0; c < 3; c++) {
                        pixels2[3 * (width * y + x) + c] =
                              pixels[bytesPerRow * y + 3 * x + (2 -
c)];
                  }
            }
      }

      input.close();
      return new Image(pixels2.release(), width, height);
}
```

Imageloader.h

```
/* Permission is hereby granted, free of charge, to any person
obtaining a copy
 * of this software and associated documentation files (the
"Software"), to deal
 * in the Software without restriction, including without limitation
the rights
 * to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell
 * copies of the Software, and to permit persons to whom the
Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above notice and this permission notice shall be included in
all copies
 * or substantial portions of the Software.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE
 * SOFTWARE.
 */
/* File for "Putting It All Together" lesson of the OpenGL tutorial
on
 * www.videotutorialsrock.com
 */

#ifndef IMAGE_LOADER_H_INCLUDED
#define IMAGE_LOADER_H_INCLUDED

//Represents an image
class Image {
      public:
           Image(char* ps, int w, int h);
           ~Image();

           /* An array of the form (R1, G1, B1, R2, G2, B2, ...)
indicating the
            * color of each pixel in image.  Color components range
from 0 to 255.
            * The array starts the bottom-left pixel, then moves
right to the end
            * of the row, then moves up to the next column, and so
on.  This is the
            * format in which OpenGL likes images.
            */
           char* pixels;
           int width;
           int height;
};
//Reads a bitmap image from file.
Image* loadBMP(const char* filename);
#endif
```

Main.cpp

```
#include <iostream>
#include <stdlib.h>

#ifdef __APPLE__
#include <OpenGL/OpenGL.h>
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
```

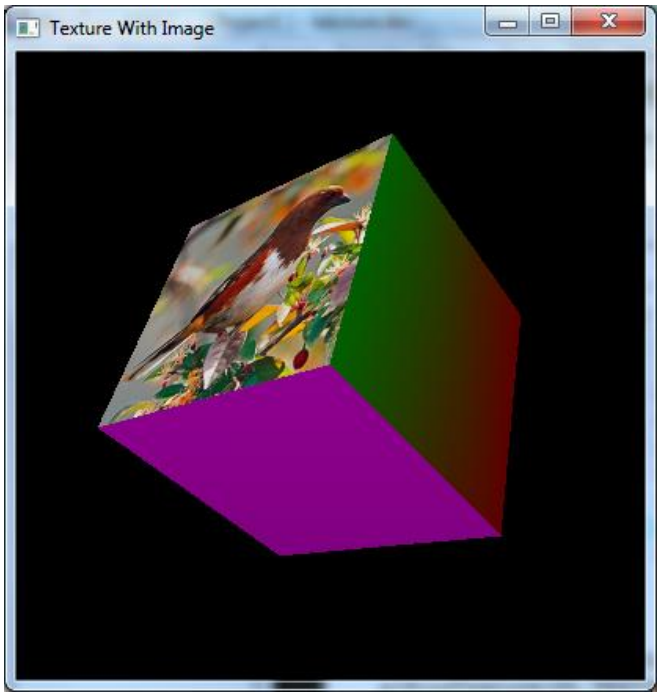
```
#endif

#include "imageloader.h"
#include "imageloader.cpp"

using namespace std;

const float BOX_SIZE = 7.0f; //Panjang tiap sisi kubus
float _angle = 0;            //Rotasi terhadap box
GLuint _textureId;           //ID OpenGL untuk tekstur

void handleKeypress(unsigned char key, int x, int y) {
    switch (key) {
     case 27:             //Tekan Escape untuk EXIT
          exit(0);
    }
}

//Membuat gambar menjadi tekstur kemudian berikan ID pada tekstur
GLuint loadTexture(Image* image) {
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image-
>height, 0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    return textureId;
}

void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);

    Image* image = loadBMP("bg.bmp");
    _textureId = loadTexture(image);
    delete image;
}

void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)w / (float)h, 1.0, 200.0);
}

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

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef(0.0f, 0.0f, -20.0f);
```

```
    GLfloat ambientLight[] = {0.3f, 0.3f, 0.3f, 1.0f};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);

    GLfloat lightColor[] = {0.7f, 0.7f, 0.7f, 1.0f};
    GLfloat lightPos[] = {-2 * BOX_SIZE, BOX_SIZE, 4 * BOX_SIZE,
1.0f};
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

    glRotatef(-_angle, 1.0f, 1.0f, 0.0f);

    glBegin(GL_QUADS);

    //Sisi atas
    glColor3f(1.0f, 1.0f, 0.0f);
    glNormal3f(0.0, 1.0f, 0.0f);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);

    //Sisi bawah
    glColor3f(1.0f, 0.0f, 1.0f);
    glNormal3f(0.0, -1.0f, 0.0f);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);

    //Sisi kiri
    glNormal3f(-1.0, 0.0f, 0.0f);
    glColor3f(0.0f, 1.0f, 1.0f);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);

    //Sisi kanan
    glNormal3f(1.0, 0.0f, 0.0f);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);

    glEnd();

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, _textureId);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
```

```
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);

    //Sisi depan
    glNormal3f(0.0, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);

    //Sisi belakang
    glNormal3f(0.0, 0.0f, -1.0f);
    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);

    glEnd();
    glDisable(GL_TEXTURE_2D);

    glutSwapBuffers();
}

//Panggil setiap 25ms
void update(int value) {
    _angle += 1.0f;
    if (_angle > 360) {
     _angle -= 360;
    }
    glutPostRedisplay();
    glutTimerFunc(25, update, 0);
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(400, 400);
    glutCreateWindow("Texture With Image");
    initRendering();
    glutDisplayFunc(drawScene);
    glutKeyboardFunc(handleKeypress);
    glutReshapeFunc(handleResize);
    glutTimerFunc(25, update, 0);
    glutMainLoop();
    return 0;
}
```

Cara Menjalankan :

1. Buat Project dengan openGL
2. Buat file imageloader.h dan imageloader.cpp pada folder project tersebut
3. Kopi file main ke main project
4. Kopi gambar ke folder project (gambar dapat di download di : http://aliipp.com/wp-content/uploads/2014/06/bg.bmp)



Pertanyaan :

1. Tekstur apa saja yang dibuat pada kubus tersebut!

Tugas :

1. Analisa dan Jelaskan alur / cara kerja teksture yang telah dikerjakan.
2. Buatlah sebuah kubus yang setiap sisinya berisi sebuah gambar.

PRAKTIKUM 8
Bayangan

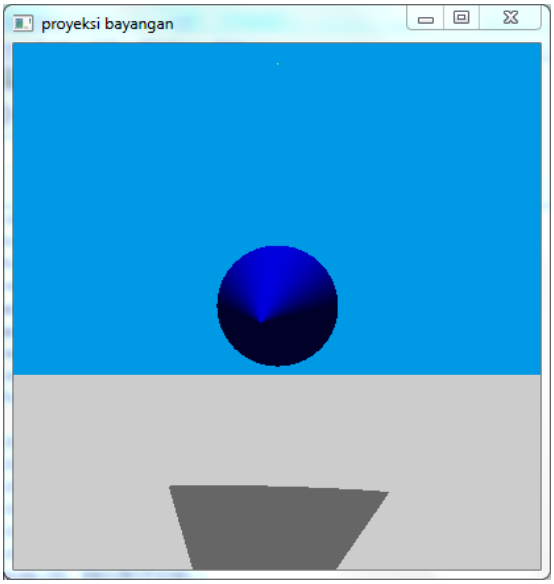
Untuk menghasilkan gambar yang realistik perlu memodelkan pencerminan dan pembiasan maupun memunculkan bayangan karena pengaruh dari adanya cahaya. Dengan memodelkan pencerminan untuk benda yang reflektif seperti cermin akan dihasilkan pantulan ataupun bayangan benda. Dan efek pembiasan cahaya dapat dimodelkan pada benda yang transparan untuk menghasilkan penampakan obyek lain yang berada di belakang obyek transparan tersebut serta efek pengumpulan cahaya bias. Effek bayangan ini sangat penting karena dengan adanya effek tersebut seolah-olah benda tersebut nampak nyata. Bayangan sebuah obyek benda harus disesuaikan dengan bentuk benda aslinya dan asal sumber cahaya tersebut berada dan banyaknya sumber cahaya.

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

double rx = 0.0;
double ry = 0.0;

float l[] = { 0.0,  80.0, 0.0 }; // koordinat sumber cahaya
float n[] = { 0.0,  -40.0, 0.0 };
float e[] = { 0.0, -60.0, 0.0 };

void help();

// obyek yang akan digambar
void draw()
{
//glutSolidTeapot(30.0);       //Isi dengan salah satu saja, dan
berilah obyek yang selain dalam tabel ini
//glutSolidTorus(20, 40, 20,10);
//glutWireSphere(60,10,10);
//glutSolidCube (40);
glutSolidCone(20,50,40,50);
}

//membuat proyeksi bayangan
void glShadowProjection(float * l, float * e, float * n)
{
  float d, c;
  float mat[16];

  d = n[0]*l[0] + n[1]*l[1] + n[2]*l[2];
  c = e[0]*n[0] + e[1]*n[1] + e[2]*n[2] - d;
```

```
  mat[0]   = l[0]*n[0]+c;                  // membuat matrik. OpenGL
menggunakan kolom matrik
  mat[4]  = n[1]*l[0];
  mat[8]  = n[2]*l[0];
  mat[12] = -l[0]*c-l[0]*d;

  mat[1]  = n[0]*l[1];
  mat[5]  = l[1]*n[1]+c;
  mat[9]  = n[2]*l[1];
  mat[13] = -l[1]*c-l[1]*d;

  mat[2]  = n[0]*l[2];
  mat[6]  = n[1]*l[2];
  mat[10] = l[2]*n[2]+c;
  mat[14] = -l[2]*c-l[2]*d;

  mat[3]  = n[0];
  mat[7]  = n[1];
  mat[11] = n[2];
  mat[15] = -d;

  glMultMatrixf(mat);              // kalikan matrik
}

void render()
{
  glClearColor(0.0,0.6,0.9,0.0);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

  glLightfv(GL_LIGHT0, GL_POSITION, l);
  glDisable(GL_CULL_FACE);
  glDisable(GL_LIGHTING);
  glColor3f(1.0,1.0,0.0);

  glBegin(GL_POINTS);
     glVertex3f(l[0],l[1],l[2]);
  glEnd();

  glColor3f(0.8,0.8,0.8);
  glBegin(GL_QUADS);
     glNormal3f(0.0,1.0,0.0);
     glVertex3f(-1300.0,e[1]-0.1, 1300.0);
     glVertex3f( 1300.0,e[1]-0.1, 1300.0);
     glVertex3f( 1300.0,e[1]-0.1,-1300.0);
     glVertex3f(-1300.0,e[1]-0.1,-1300.0);
  glEnd();

  // gambar bayangan
```

```
  glPushMatrix();
  glRotatef(ry,0,1,0);
  glRotatef(rx,1,0,0);
  glEnable(GL_LIGHTING);
  glColor3f(0.0,0.0,0.8);
  draw();
  glPopMatrix();

  //sekarang  gambar bayangan yang muncul
  glPushMatrix();
  glShadowProjection(l,e,n);
  glRotatef(ry,0,1,0);
  glRotatef(rx,1,0,0);
  glDisable(GL_LIGHTING);
  glColor3f(0.4,0.4,0.4);
  draw();
  glPopMatrix();
  glutSwapBuffers();
}


void keypress(unsigned char c, int a, int b)
{
  if ( c==27 ) exit(0);
  else if ( c=='s' ) l[1]-=5.0;
  else if ( c=='w' ) l[1]+=5.0;
  else if ( c=='a' ) l[0]-=5.0;
  else if ( c=='d' ) l[0]+=5.0;
  else if ( c=='q' ) l[2]-=5.0;
  else if ( c=='e' ) l[2]+=5.0;
  else if ( c=='h' ) help();
}

void help()
{
  printf("proyeksi contoh bayangan sebuah obyek teapot\n");
}

void idle()
{
  rx+=0.1;
  ry+=0.1;
  render();
}

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

```
}

int main(int argc, char * argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowPosition(300,30);
  glutCreateWindow("proyeksi bayangan");
  glutReshapeFunc(resize);
  glutReshapeWindow(400,400);
  glutKeyboardFunc(keypress);
  glutDisplayFunc(render);
  glutIdleFunc(idle);

  glEnable(GL_NORMALIZE);
  glEnable(GL_LIGHTING);
  glEnable(GL_COLOR_MATERIAL);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHT0);
  glEnable(GL_TEXTURE_2D);
  glMatrixMode(GL_PROJECTION);

  glLoadIdentity();
  gluPerspective(60.0f, 1.0, 1.0, 400.0);

  // Reset koordinat  sebelum dimodifikasi/diubah
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0, 0.0, -150.0);

  glutMainLoop();
  return 0;
}
```



Tugas dan Pertanyaan :

1. Ubahlah untuk object yang berbeda
2. Analisis dan Jelaskan secara rinci proses membuat bayangannya.
3. Jelaskan Fungsi dan alur **`glShadowProjection.`**