1. Bangun kubus 3D

#include <GL/glut.h>

void drawCube(int n) {

glBegin(GL\_QUADS);

glColor3f(1.0, 0.0, 0.0); // Merah

glVertex3f(-n, -n, -n);

glVertex3f(n, -n, -n);

glVertex3f(n, n, -n);

glVertex3f(-n, n, -n);

glColor3f(0.0, 1.0, 0.0); // Hijau

glVertex3f(-n, -n, n);

glVertex3f(n, -n, n);

glVertex3f(n, n, n);

glVertex3f(-n, n, n);

glColor3f(0.0, 0.0, 1.0); // Biru

glVertex3f(-n, -n, -n);

glVertex3f(-n, n, -n);

glVertex3f(-n, n, n);

glVertex3f(-n, -n, n);

glVertex3f(n, -n, -n);

glVertex3f(n, n, -n);

glVertex3f(n, n, n);

glVertex3f(n, -n, n);

glColor3f(1.0, 1.0, 0.0); // Kuning

glVertex3f(-n, -n, -n);

glVertex3f(n, -n, -n);

glVertex3f(n, -n, n);

glVertex3f(-n, -n, n);

glVertex3f(-n, n, -n);

glVertex3f(n, n, -n);

glVertex3f(n, n, n);

glVertex3f(-n, n, n);

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

gluLookAt(5.0, 5.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glRotatef(45, 1, 1, 0);

drawCube(4); // Ganti dengan n yang diinginkan (4, 8, 12, 20)

glutSwapBuffers();

}

void reshape(int width, int height) {

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (float)width / (float)height, 1.0, 100.0);

glMatrixMode(GL\_MODELVIEW);

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutCreateWindow("Kubus 3D");

glEnable(GL\_DEPTH\_TEST);

glClearColor(1.0, 1.0, 1.0, 1.0); // Latar belakang putih

glutDisplayFunc(display);

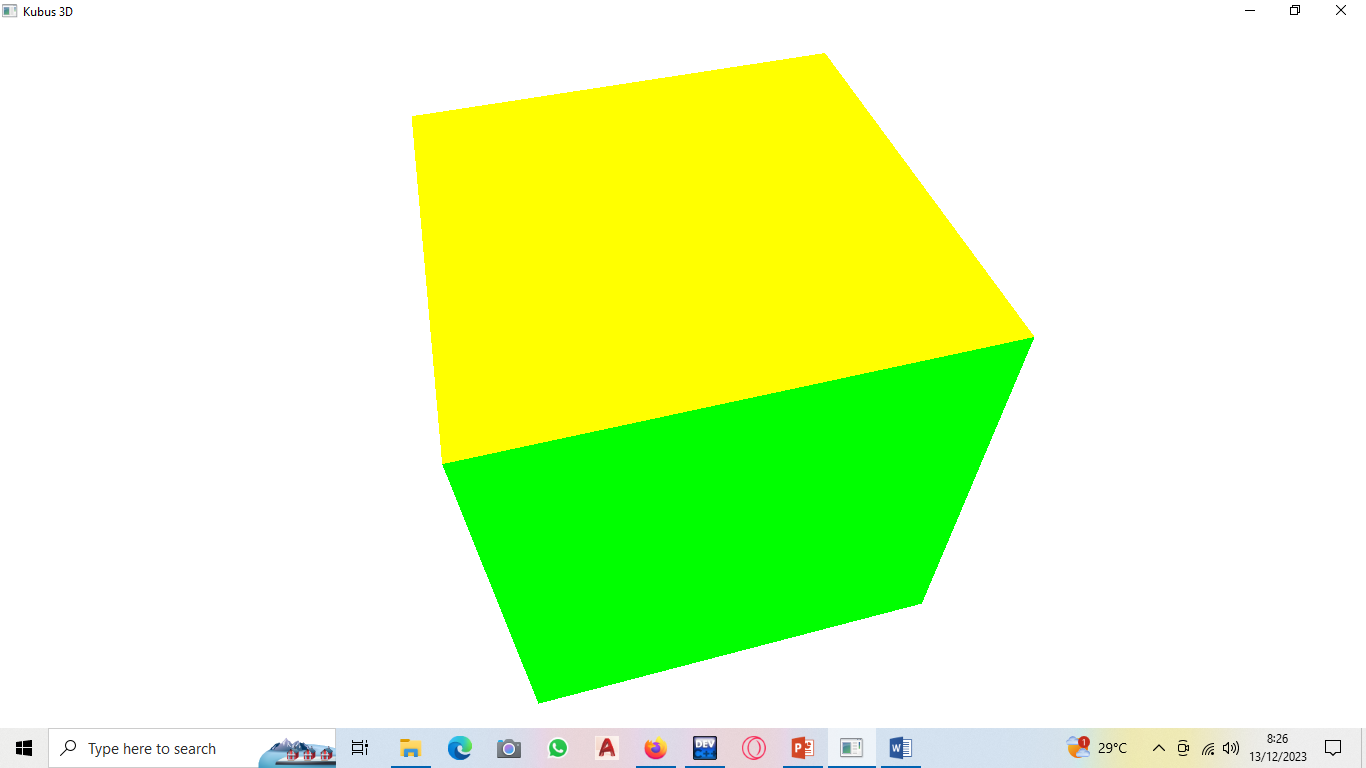
glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

Hasil running



1. Kerucut
2. Diket n: 4

#include <math.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include <stdio.h>

#define WIDTH 400

#define HEIGHT 400

typedef struct

{

float m[4][4];

} Matrix3D\_t;

typedef struct

{

float v[4];

} Vector3D\_t;

typedef struct

{

float x;

float y;

float z;

} Point3D\_t;

typedef struct

{

float x;

float y;

} Point2D\_t;

typedef struct

{

float r;

float g;

float b;

} Color\_t;

Matrix3D\_t createIdentity(void)

{

Matrix3D\_t u;

int i, j;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++) u.m[i][j] = 0.;

u.m[i][i] = 1.;

}

return u;

}

Matrix3D\_t operator \*(Matrix3D\_t a, Matrix3D\_t b)

{

Matrix3D\_t c;//c=a\*b

int i, j, k;

for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

{

c.m[i][j] = 0;

for (k = 0; k < 4; k++) c.m[i][j] += a.m[i][k] \* b.m[k][j];

}

return c;

}

Vector3D\_t operator \*(Matrix3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

int i, j;

for (i = 0; i < 4; i++)

{

c.v[i] = 0;

for (j = 0; j < 4; j++) c.v[i] += a.m[i][j] \* b.v[j];

}

return c;

}

Matrix3D\_t rotationX(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[1][1] = cs;

rotate.m[1][2] = -sn;

rotate.m[2][1] = sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationY(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][2] = sn;

rotate.m[2][0] = -sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationZ(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][1] = -sn;

rotate.m[1][0] = sn;

rotate.m[1][1] = cs;

return rotate;

}

Point2D\_t Vector2Point2D(Vector3D\_t vec)

{

Point2D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

return pnt;

}

Point3D\_t Vector2Point3D(Vector3D\_t vec)

{

Point3D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

pnt.z = vec.v[2];

return pnt;

}

Vector3D\_t Point2Vector(Point3D\_t pnt)

{

Vector3D\_t vec;

vec.v[0] = pnt.x;

vec.v[1] = pnt.y;

vec.v[2] = pnt.z;

vec.v[3] = 1.;

return vec;

}

float operator \*(Vector3D\_t a, Vector3D\_t b)

{

float c;//c=a\*b

int i;

c = 0;

for (i = 0; i < 3; i++)

{

c += a.v[i] \* b.v[i];

}

return c;

}

Vector3D\_t operator ^(Vector3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

c.v[0] = a.v[1] \* b.v[2] - a.v[2] \* b.v[1];

c.v[1] = a.v[2] \* b.v[0] - a.v[0] \* b.v[2];

c.v[2] = a.v[0] \* b.v[1] - a.v[1] \* b.v[0];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v1, Vector3D\_t v0)

{

Vector3D\_t c;//c=v1-v0

c.v[0] = v1.v[0] - v0.v[0];

c.v[1] = v1.v[1] - v0.v[1];

c.v[2] = v1.v[2] - v0.v[2];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v)

{

Vector3D\_t c;//c=-v

c.v[0] = -v.v[0];

c.v[1] = -v.v[1];

c.v[2] = -v.v[2];

c.v[3] = 1.;

return c;

}

void setColor(float red, float green, float blue)

{

glColor3f(red, green, blue);

}

void setColor(Color\_t col)

{

glColor3f(col.r, col.g, col.b);

}

void drawDot(float x, float y)

{

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

}

void drawDot(Point2D\_t p)

{

glBegin(GL\_POINTS);

glVertex2f(p.x, p.y);

glEnd();

}

void drawLine(float x1, float y1, float x2, float y2)

{

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

void drawLine(Point2D\_t p1, Point2D\_t p2)

{

drawLine(p1.x, p1.y, p2.x, p2.y);

}

//n: number of points

void drawPolyline(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//n: number of vertices

void drawPolygon(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_LOOP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void fillPolygon(Point2D\_t pnt[], int n, Color\_t color)

{

int i;

setColor(color);

glBegin(GL\_POLYGON);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void gradatePolygon(Point2D\_t pnt[], int num, Color\_t col[])

{

int i;

glBegin(GL\_POLYGON);

for (i = 0; i < num; i++)

{

setColor(col[i]);

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//////////// End of OpenGL drawShape Functions ver 1 ////////////

void drawcharX(float x, float y)

{

drawLine(x, y, x + 10, y + 12);

drawLine(x, y + 12, x + 10, y);

}

void drawcharY(float x, float y)

{

drawLine(x + 5, y, x + 5, y + 7);

drawLine(x, y + 12, x + 5, y + 7);

drawLine(x + 10, y + 12, x + 5, y + 7);

}

void drawcharZ(float x, float y)

{

drawLine(x, y + 12, x + 10, y + 12);

drawLine(x + 10, y + 12, x, y);

drawLine(x, y, x + 10, y);

}

typedef struct

{

int numofVertices; //in the face

short int pnt[50];

Color\_t col;

} Face\_t;

typedef struct

{

int numofVertices; //of the object

Point3D\_t pnt[1600];

Color\_t col[1600];

int numofFaces; //of the object

Face\_t fc[1000];

} Object3D\_t;

void draw3D(Object3D\_t obyek, Matrix3D\_t mat)

{

Vector3D\_t vec[32], vecbuff[32];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(1, 0, 0);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] < 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

setColor(0, 1, 1);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void draw3D(Object3D\_t obyek, Matrix3D\_t mat, Color\_t col)

{

Vector3D\_t vec[1600], vecbuff[50];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(col);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void makeKerucut(Object3D\_t& kerucut, int n, float h, float r)

{

float a = 6.28 /4;

int i;

kerucut.pnt[0].x = 0;

kerucut.pnt[0].y = h;

kerucut.pnt[0].z = 0;

for (i = 1; i <= n; i++) {

kerucut.pnt[i].x = r\*cos(i\*a);

kerucut.pnt[i].y = 0;

kerucut.pnt[i].z = r\*sin(i\*a);

}

for (i = 0; i<n; i++) {

kerucut.fc[i].numofVertices = 3;

kerucut.fc[i].pnt[0] = 0;

kerucut.fc[i].pnt[1] = i + 2;

kerucut.fc[i].pnt[2] = i + 1;

if (i == (n - 1)) kerucut.fc[i].pnt[1] = 1;

}

kerucut.fc[n].numofVertices = n;

for (i = 0; i<n; i++) kerucut.fc[n].pnt[i] = i + 1;

kerucut.numofVertices = n + 1;

kerucut.numofFaces = n + 1;

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(1000, timer, 0);

}

void Initialize()

{

glClearColor(0.5, 0.5, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1 \* (WIDTH / 2), WIDTH / 2, -1 \* (HEIGHT / 2), HEIGHT / 2);

}

float sudut = 0.0;

void userdraw(void)

{

Matrix3D\_t tilting = rotationX(170)\*rotationY(sudut);

setColor(0, 1, 0);

Object3D\_t kerucut;

makeKerucut(kerucut, 20, 100, 50);

setColor(1, 1, 1);

draw3D(kerucut, tilting);

sudut++;

if (sudut >= 360.0) sudut = 0.0;

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

userdraw();

glutSwapBuffers();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutInitWindowPosition(200, 200);

glutCreateWindow("Kerucut");

Initialize();

glutDisplayFunc(display);

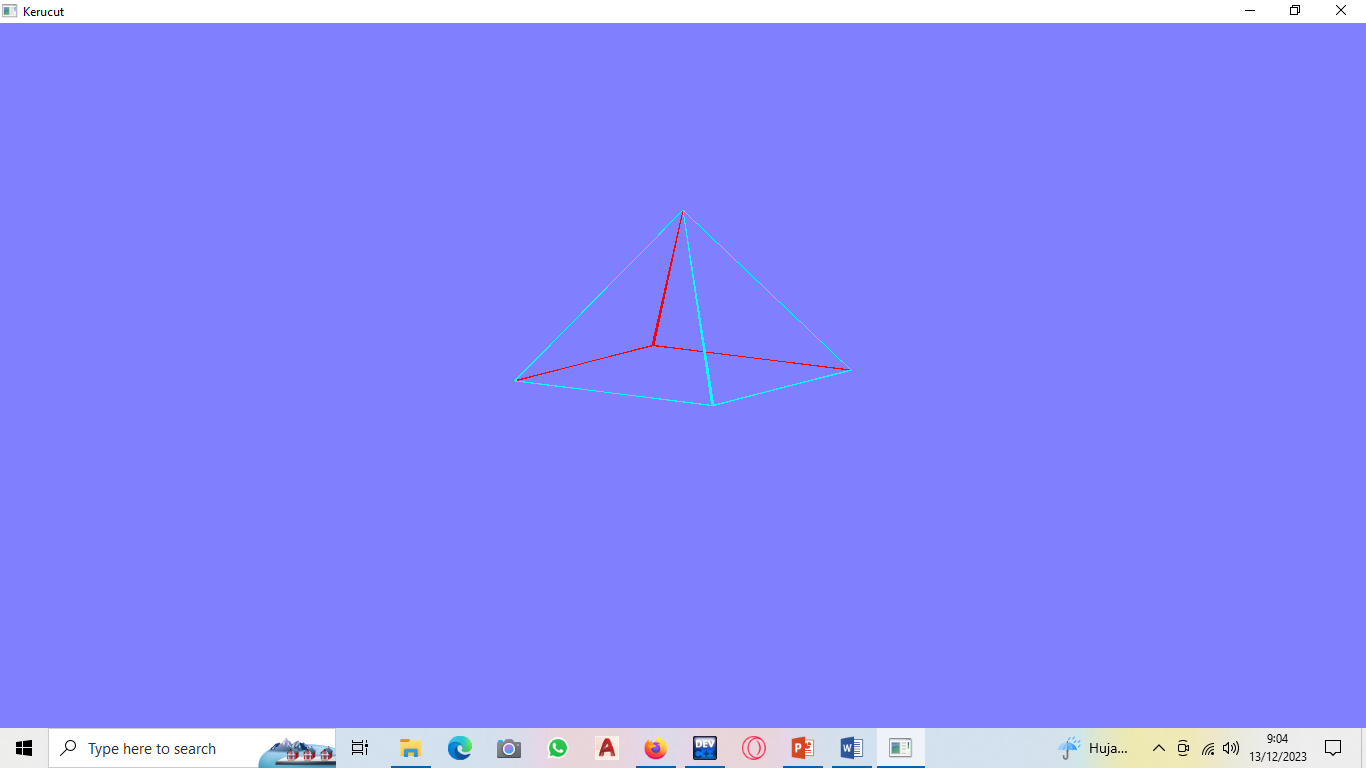
glutTimerFunc(1, timer, 0);

glutMainLoop();

return 0;

}

Hasil running



1. Diket n: 8

#include <math.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include <stdio.h>

#define WIDTH 400

#define HEIGHT 400

typedef struct

{

float m[4][4];

} Matrix3D\_t;

typedef struct

{

float v[4];

} Vector3D\_t;

typedef struct

{

float x;

float y;

float z;

} Point3D\_t;

typedef struct

{

float x;

float y;

} Point2D\_t;

typedef struct

{

float r;

float g;

float b;

} Color\_t;

Matrix3D\_t createIdentity(void)

{

Matrix3D\_t u;

int i, j;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++) u.m[i][j] = 0.;

u.m[i][i] = 1.;

}

return u;

}

Matrix3D\_t operator \*(Matrix3D\_t a, Matrix3D\_t b)

{

Matrix3D\_t c;//c=a\*b

int i, j, k;

for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

{

c.m[i][j] = 0;

for (k = 0; k < 4; k++) c.m[i][j] += a.m[i][k] \* b.m[k][j];

}

return c;

}

Vector3D\_t operator \*(Matrix3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

int i, j;

for (i = 0; i < 4; i++)

{

c.v[i] = 0;

for (j = 0; j < 4; j++) c.v[i] += a.m[i][j] \* b.v[j];

}

return c;

}

Matrix3D\_t rotationX(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[1][1] = cs;

rotate.m[1][2] = -sn;

rotate.m[2][1] = sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationY(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][2] = sn;

rotate.m[2][0] = -sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationZ(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][1] = -sn;

rotate.m[1][0] = sn;

rotate.m[1][1] = cs;

return rotate;

}

Point2D\_t Vector2Point2D(Vector3D\_t vec)

{

Point2D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

return pnt;

}

Point3D\_t Vector2Point3D(Vector3D\_t vec)

{

Point3D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

pnt.z = vec.v[2];

return pnt;

}

Vector3D\_t Point2Vector(Point3D\_t pnt)

{

Vector3D\_t vec;

vec.v[0] = pnt.x;

vec.v[1] = pnt.y;

vec.v[2] = pnt.z;

vec.v[3] = 1.;

return vec;

}

float operator \*(Vector3D\_t a, Vector3D\_t b)

{

float c;//c=a\*b

int i;

c = 0;

for (i = 0; i < 3; i++)

{

c += a.v[i] \* b.v[i];

}

return c;

}

Vector3D\_t operator ^(Vector3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

c.v[0] = a.v[1] \* b.v[2] - a.v[2] \* b.v[1];

c.v[1] = a.v[2] \* b.v[0] - a.v[0] \* b.v[2];

c.v[2] = a.v[0] \* b.v[1] - a.v[1] \* b.v[0];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v1, Vector3D\_t v0)

{

Vector3D\_t c;//c=v1-v0

c.v[0] = v1.v[0] - v0.v[0];

c.v[1] = v1.v[1] - v0.v[1];

c.v[2] = v1.v[2] - v0.v[2];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v)

{

Vector3D\_t c;//c=-v

c.v[0] = -v.v[0];

c.v[1] = -v.v[1];

c.v[2] = -v.v[2];

c.v[3] = 1.;

return c;

}

void setColor(float red, float green, float blue)

{

glColor3f(red, green, blue);

}

void setColor(Color\_t col)

{

glColor3f(col.r, col.g, col.b);

}

void drawDot(float x, float y)

{

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

}

void drawDot(Point2D\_t p)

{

glBegin(GL\_POINTS);

glVertex2f(p.x, p.y);

glEnd();

}

void drawLine(float x1, float y1, float x2, float y2)

{

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

void drawLine(Point2D\_t p1, Point2D\_t p2)

{

drawLine(p1.x, p1.y, p2.x, p2.y);

}

//n: number of points

void drawPolyline(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//n: number of vertices

void drawPolygon(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_LOOP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void fillPolygon(Point2D\_t pnt[], int n, Color\_t color)

{

int i;

setColor(color);

glBegin(GL\_POLYGON);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void gradatePolygon(Point2D\_t pnt[], int num, Color\_t col[])

{

int i;

glBegin(GL\_POLYGON);

for (i = 0; i < num; i++)

{

setColor(col[i]);

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//////////// End of OpenGL drawShape Functions ver 1 ////////////

void drawcharX(float x, float y)

{

drawLine(x, y, x + 10, y + 12);

drawLine(x, y + 12, x + 10, y);

}

void drawcharY(float x, float y)

{

drawLine(x + 5, y, x + 5, y + 7);

drawLine(x, y + 12, x + 5, y + 7);

drawLine(x + 10, y + 12, x + 5, y + 7);

}

void drawcharZ(float x, float y)

{

drawLine(x, y + 12, x + 10, y + 12);

drawLine(x + 10, y + 12, x, y);

drawLine(x, y, x + 10, y);

}

typedef struct

{

int numofVertices; //in the face

short int pnt[50];

Color\_t col;

} Face\_t;

typedef struct

{

int numofVertices; //of the object

Point3D\_t pnt[1600];

Color\_t col[1600];

int numofFaces; //of the object

Face\_t fc[1000];

} Object3D\_t;

void draw3D(Object3D\_t obyek, Matrix3D\_t mat)

{

Vector3D\_t vec[32], vecbuff[32];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(1, 0, 0);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] < 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

setColor(0, 1, 1);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void draw3D(Object3D\_t obyek, Matrix3D\_t mat, Color\_t col)

{

Vector3D\_t vec[1600], vecbuff[50];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(col);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void makeKerucut(Object3D\_t& kerucut, int n, float h, float r)

{

float a = 6.28 /8;

int i;

kerucut.pnt[0].x = 0;

kerucut.pnt[0].y = h;

kerucut.pnt[0].z = 0;

for (i = 1; i <= n; i++) {

kerucut.pnt[i].x = r\*cos(i\*a);

kerucut.pnt[i].y = 0;

kerucut.pnt[i].z = r\*sin(i\*a);

}

for (i = 0; i<n; i++) {

kerucut.fc[i].numofVertices = 3;

kerucut.fc[i].pnt[0] = 0;

kerucut.fc[i].pnt[1] = i + 2;

kerucut.fc[i].pnt[2] = i + 1;

if (i == (n - 1)) kerucut.fc[i].pnt[1] = 1;

}

kerucut.fc[n].numofVertices = n;

for (i = 0; i<n; i++) kerucut.fc[n].pnt[i] = i + 1;

kerucut.numofVertices = n + 1;

kerucut.numofFaces = n + 1;

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(1000, timer, 0);

}

void Initialize()

{

glClearColor(0.5, 0.5, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1 \* (WIDTH / 2), WIDTH / 2, -1 \* (HEIGHT / 2), HEIGHT / 2);

}

float sudut = 0.0;

void userdraw(void)

{

Matrix3D\_t tilting = rotationX(170)\*rotationY(sudut);

setColor(0, 1, 0);

Object3D\_t kerucut;

makeKerucut(kerucut, 20, 100, 50);

setColor(1, 1, 1);

draw3D(kerucut, tilting);

sudut++;

if (sudut >= 360.0) sudut = 0.0;

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

userdraw();

glutSwapBuffers();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutInitWindowPosition(200, 200);

glutCreateWindow("Kerucut");

Initialize();

glutDisplayFunc(display);

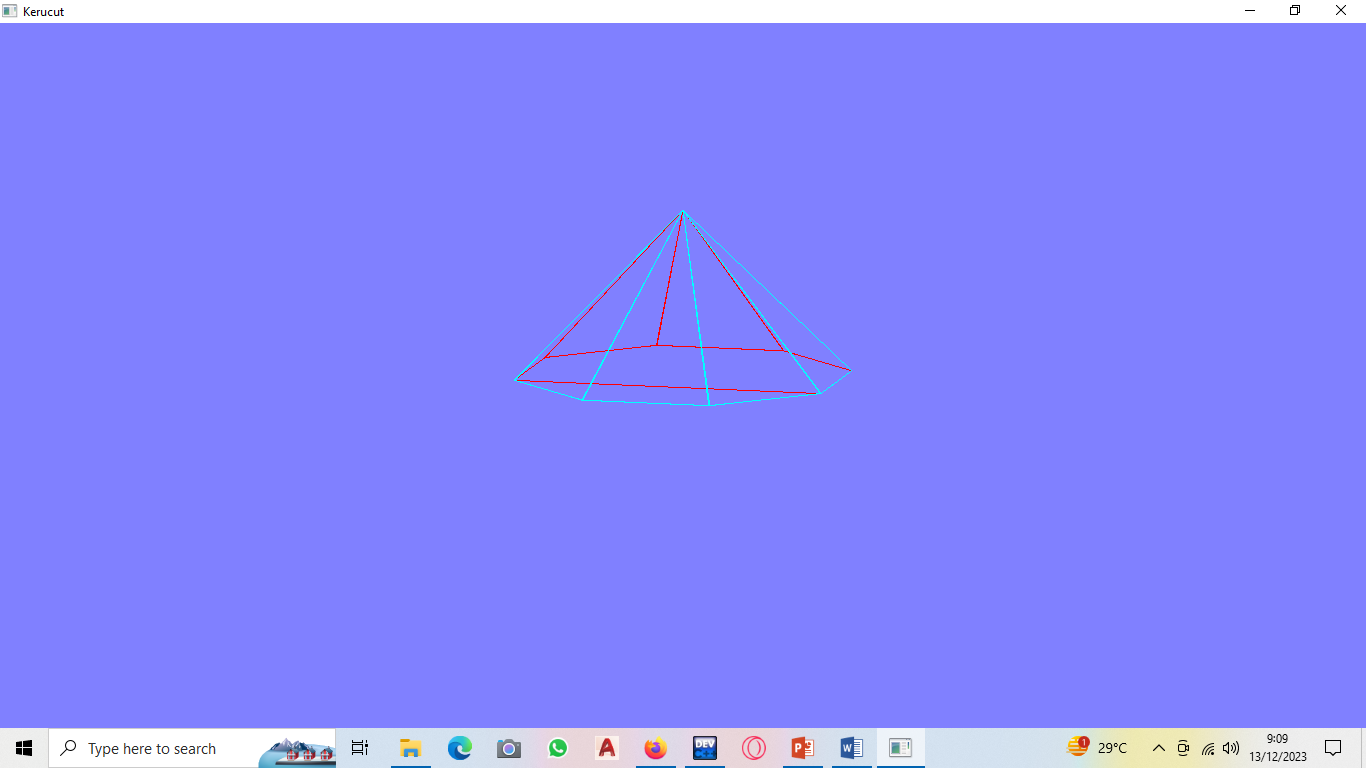
glutTimerFunc(1, timer, 0);

glutMainLoop();

return 0;

}

Hasil running



1. Diket : 12

#include <math.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include <stdio.h>

#define WIDTH 400

#define HEIGHT 400

typedef struct

{

float m[4][4];

} Matrix3D\_t;

typedef struct

{

float v[4];

} Vector3D\_t;

typedef struct

{

float x;

float y;

float z;

} Point3D\_t;

typedef struct

{

float x;

float y;

} Point2D\_t;

typedef struct

{

float r;

float g;

float b;

} Color\_t;

Matrix3D\_t createIdentity(void)

{

Matrix3D\_t u;

int i, j;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++) u.m[i][j] = 0.;

u.m[i][i] = 1.;

}

return u;

}

Matrix3D\_t operator \*(Matrix3D\_t a, Matrix3D\_t b)

{

Matrix3D\_t c;//c=a\*b

int i, j, k;

for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

{

c.m[i][j] = 0;

for (k = 0; k < 4; k++) c.m[i][j] += a.m[i][k] \* b.m[k][j];

}

return c;

}

Vector3D\_t operator \*(Matrix3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

int i, j;

for (i = 0; i < 4; i++)

{

c.v[i] = 0;

for (j = 0; j < 4; j++) c.v[i] += a.m[i][j] \* b.v[j];

}

return c;

}

Matrix3D\_t rotationX(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[1][1] = cs;

rotate.m[1][2] = -sn;

rotate.m[2][1] = sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationY(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][2] = sn;

rotate.m[2][0] = -sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationZ(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][1] = -sn;

rotate.m[1][0] = sn;

rotate.m[1][1] = cs;

return rotate;

}

Point2D\_t Vector2Point2D(Vector3D\_t vec)

{

Point2D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

return pnt;

}

Point3D\_t Vector2Point3D(Vector3D\_t vec)

{

Point3D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

pnt.z = vec.v[2];

return pnt;

}

Vector3D\_t Point2Vector(Point3D\_t pnt)

{

Vector3D\_t vec;

vec.v[0] = pnt.x;

vec.v[1] = pnt.y;

vec.v[2] = pnt.z;

vec.v[3] = 1.;

return vec;

}

float operator \*(Vector3D\_t a, Vector3D\_t b)

{

float c;//c=a\*b

int i;

c = 0;

for (i = 0; i < 3; i++)

{

c += a.v[i] \* b.v[i];

}

return c;

}

Vector3D\_t operator ^(Vector3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

c.v[0] = a.v[1] \* b.v[2] - a.v[2] \* b.v[1];

c.v[1] = a.v[2] \* b.v[0] - a.v[0] \* b.v[2];

c.v[2] = a.v[0] \* b.v[1] - a.v[1] \* b.v[0];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v1, Vector3D\_t v0)

{

Vector3D\_t c;//c=v1-v0

c.v[0] = v1.v[0] - v0.v[0];

c.v[1] = v1.v[1] - v0.v[1];

c.v[2] = v1.v[2] - v0.v[2];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v)

{

Vector3D\_t c;//c=-v

c.v[0] = -v.v[0];

c.v[1] = -v.v[1];

c.v[2] = -v.v[2];

c.v[3] = 1.;

return c;

}

void setColor(float red, float green, float blue)

{

glColor3f(red, green, blue);

}

void setColor(Color\_t col)

{

glColor3f(col.r, col.g, col.b);

}

void drawDot(float x, float y)

{

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

}

void drawDot(Point2D\_t p)

{

glBegin(GL\_POINTS);

glVertex2f(p.x, p.y);

glEnd();

}

void drawLine(float x1, float y1, float x2, float y2)

{

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

void drawLine(Point2D\_t p1, Point2D\_t p2)

{

drawLine(p1.x, p1.y, p2.x, p2.y);

}

//n: number of points

void drawPolyline(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//n: number of vertices

void drawPolygon(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_LOOP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void fillPolygon(Point2D\_t pnt[], int n, Color\_t color)

{

int i;

setColor(color);

glBegin(GL\_POLYGON);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void gradatePolygon(Point2D\_t pnt[], int num, Color\_t col[])

{

int i;

glBegin(GL\_POLYGON);

for (i = 0; i < num; i++)

{

setColor(col[i]);

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//////////// End of OpenGL drawShape Functions ver 1 ////////////

void drawcharX(float x, float y)

{

drawLine(x, y, x + 10, y + 12);

drawLine(x, y + 12, x + 10, y);

}

void drawcharY(float x, float y)

{

drawLine(x + 5, y, x + 5, y + 7);

drawLine(x, y + 12, x + 5, y + 7);

drawLine(x + 10, y + 12, x + 5, y + 7);

}

void drawcharZ(float x, float y)

{

drawLine(x, y + 12, x + 10, y + 12);

drawLine(x + 10, y + 12, x, y);

drawLine(x, y, x + 10, y);

}

typedef struct

{

int numofVertices; //in the face

short int pnt[50];

Color\_t col;

} Face\_t;

typedef struct

{

int numofVertices; //of the object

Point3D\_t pnt[1600];

Color\_t col[1600];

int numofFaces; //of the object

Face\_t fc[1000];

} Object3D\_t;

void draw3D(Object3D\_t obyek, Matrix3D\_t mat)

{

Vector3D\_t vec[32], vecbuff[32];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(1, 0, 0);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] < 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

setColor(0, 1, 1);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void draw3D(Object3D\_t obyek, Matrix3D\_t mat, Color\_t col)

{

Vector3D\_t vec[1600], vecbuff[50];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(col);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void makeKerucut(Object3D\_t& kerucut, int n, float h, float r)

{

float a = 6.28 /12;

int i;

kerucut.pnt[0].x = 0;

kerucut.pnt[0].y = h;

kerucut.pnt[0].z = 0;

for (i = 1; i <= n; i++) {

kerucut.pnt[i].x = r\*cos(i\*a);

kerucut.pnt[i].y = 0;

kerucut.pnt[i].z = r\*sin(i\*a);

}

for (i = 0; i<n; i++) {

kerucut.fc[i].numofVertices = 3;

kerucut.fc[i].pnt[0] = 0;

kerucut.fc[i].pnt[1] = i + 2;

kerucut.fc[i].pnt[2] = i + 1;

if (i == (n - 1)) kerucut.fc[i].pnt[1] = 1;

}

kerucut.fc[n].numofVertices = n;

for (i = 0; i<n; i++) kerucut.fc[n].pnt[i] = i + 1;

kerucut.numofVertices = n + 1;

kerucut.numofFaces = n + 1;

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(1000, timer, 0);

}

void Initialize()

{

glClearColor(0.5, 0.5, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1 \* (WIDTH / 2), WIDTH / 2, -1 \* (HEIGHT / 2), HEIGHT / 2);

}

float sudut = 0.0;

void userdraw(void)

{

Matrix3D\_t tilting = rotationX(170)\*rotationY(sudut);

setColor(0, 1, 0);

Object3D\_t kerucut;

makeKerucut(kerucut, 20, 100, 50);

setColor(1, 1, 1);

draw3D(kerucut, tilting);

sudut++;

if (sudut >= 360.0) sudut = 0.0;

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

userdraw();

glutSwapBuffers();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutInitWindowPosition(200, 200);

glutCreateWindow("Kerucut");

Initialize();

glutDisplayFunc(display);

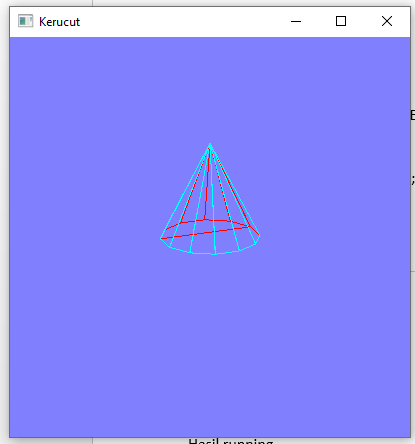
glutTimerFunc(1, timer, 0);

glutMainLoop();

return 0;

}

Hasil running



1. Diket : n=20

#include <math.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include <stdio.h>

#define WIDTH 400

#define HEIGHT 400

typedef struct

{

float m[4][4];

} Matrix3D\_t;

typedef struct

{

float v[4];

} Vector3D\_t;

typedef struct

{

float x;

float y;

float z;

} Point3D\_t;

typedef struct

{

float x;

float y;

} Point2D\_t;

typedef struct

{

float r;

float g;

float b;

} Color\_t;

Matrix3D\_t createIdentity(void)

{

Matrix3D\_t u;

int i, j;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4; j++) u.m[i][j] = 0.;

u.m[i][i] = 1.;

}

return u;

}

Matrix3D\_t operator \*(Matrix3D\_t a, Matrix3D\_t b)

{

Matrix3D\_t c;//c=a\*b

int i, j, k;

for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

{

c.m[i][j] = 0;

for (k = 0; k < 4; k++) c.m[i][j] += a.m[i][k] \* b.m[k][j];

}

return c;

}

Vector3D\_t operator \*(Matrix3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

int i, j;

for (i = 0; i < 4; i++)

{

c.v[i] = 0;

for (j = 0; j < 4; j++) c.v[i] += a.m[i][j] \* b.v[j];

}

return c;

}

Matrix3D\_t rotationX(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[1][1] = cs;

rotate.m[1][2] = -sn;

rotate.m[2][1] = sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationY(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][2] = sn;

rotate.m[2][0] = -sn;

rotate.m[2][2] = cs;

return rotate;

}

Matrix3D\_t rotationZ(float theta)

{

Matrix3D\_t rotate = createIdentity();

float cs = cos(theta);

float sn = sin(theta);

rotate.m[0][0] = cs;

rotate.m[0][1] = -sn;

rotate.m[1][0] = sn;

rotate.m[1][1] = cs;

return rotate;

}

Point2D\_t Vector2Point2D(Vector3D\_t vec)

{

Point2D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

return pnt;

}

Point3D\_t Vector2Point3D(Vector3D\_t vec)

{

Point3D\_t pnt;

pnt.x = vec.v[0];

pnt.y = vec.v[1];

pnt.z = vec.v[2];

return pnt;

}

Vector3D\_t Point2Vector(Point3D\_t pnt)

{

Vector3D\_t vec;

vec.v[0] = pnt.x;

vec.v[1] = pnt.y;

vec.v[2] = pnt.z;

vec.v[3] = 1.;

return vec;

}

float operator \*(Vector3D\_t a, Vector3D\_t b)

{

float c;//c=a\*b

int i;

c = 0;

for (i = 0; i < 3; i++)

{

c += a.v[i] \* b.v[i];

}

return c;

}

Vector3D\_t operator ^(Vector3D\_t a, Vector3D\_t b)

{

Vector3D\_t c;//c=a\*b

c.v[0] = a.v[1] \* b.v[2] - a.v[2] \* b.v[1];

c.v[1] = a.v[2] \* b.v[0] - a.v[0] \* b.v[2];

c.v[2] = a.v[0] \* b.v[1] - a.v[1] \* b.v[0];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v1, Vector3D\_t v0)

{

Vector3D\_t c;//c=v1-v0

c.v[0] = v1.v[0] - v0.v[0];

c.v[1] = v1.v[1] - v0.v[1];

c.v[2] = v1.v[2] - v0.v[2];

c.v[3] = 1.;

return c;

}

Vector3D\_t operator -(Vector3D\_t v)

{

Vector3D\_t c;//c=-v

c.v[0] = -v.v[0];

c.v[1] = -v.v[1];

c.v[2] = -v.v[2];

c.v[3] = 1.;

return c;

}

void setColor(float red, float green, float blue)

{

glColor3f(red, green, blue);

}

void setColor(Color\_t col)

{

glColor3f(col.r, col.g, col.b);

}

void drawDot(float x, float y)

{

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

}

void drawDot(Point2D\_t p)

{

glBegin(GL\_POINTS);

glVertex2f(p.x, p.y);

glEnd();

}

void drawLine(float x1, float y1, float x2, float y2)

{

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

void drawLine(Point2D\_t p1, Point2D\_t p2)

{

drawLine(p1.x, p1.y, p2.x, p2.y);

}

//n: number of points

void drawPolyline(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_STRIP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//n: number of vertices

void drawPolygon(Point2D\_t pnt[], int n)

{

int i;

glBegin(GL\_LINE\_LOOP);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void fillPolygon(Point2D\_t pnt[], int n, Color\_t color)

{

int i;

setColor(color);

glBegin(GL\_POLYGON);

for (i = 0; i < n; i++)

{

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

void gradatePolygon(Point2D\_t pnt[], int num, Color\_t col[])

{

int i;

glBegin(GL\_POLYGON);

for (i = 0; i < num; i++)

{

setColor(col[i]);

glVertex2f(pnt[i].x, pnt[i].y);

}

glEnd();

}

//////////// End of OpenGL drawShape Functions ver 1 ////////////

void drawcharX(float x, float y)

{

drawLine(x, y, x + 10, y + 12);

drawLine(x, y + 12, x + 10, y);

}

void drawcharY(float x, float y)

{

drawLine(x + 5, y, x + 5, y + 7);

drawLine(x, y + 12, x + 5, y + 7);

drawLine(x + 10, y + 12, x + 5, y + 7);

}

void drawcharZ(float x, float y)

{

drawLine(x, y + 12, x + 10, y + 12);

drawLine(x + 10, y + 12, x, y);

drawLine(x, y, x + 10, y);

}

typedef struct

{

int numofVertices; //in the face

short int pnt[50];

Color\_t col;

} Face\_t;

typedef struct

{

int numofVertices; //of the object

Point3D\_t pnt[1600];

Color\_t col[1600];

int numofFaces; //of the object

Face\_t fc[1000];

} Object3D\_t;

void draw3D(Object3D\_t obyek, Matrix3D\_t mat)

{

Vector3D\_t vec[32], vecbuff[32];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(1, 0, 0);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] < 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

setColor(0, 1, 1);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void draw3D(Object3D\_t obyek, Matrix3D\_t mat, Color\_t col)

{

Vector3D\_t vec[1600], vecbuff[50];

Vector3D\_t vecNormal;

Point2D\_t p[50];

int i, j;

for (i = 0; i < obyek.numofVertices; i++)

{

vec[i] = Point2Vector(obyek.pnt[i]);

vec[i] = mat \* vec[i];

}

setColor(col);

for (i = 0; i < obyek.numofFaces; i++)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

vecbuff[j] = vec[obyek.fc[i].pnt[j]];

vecNormal = (vecbuff[1] - vecbuff[0]) ^ (vecbuff[2] - vecbuff[0]);

if (vecNormal.v[2] >= 0)

{

for (j = 0; j < obyek.fc[i].numofVertices; j++)

{

p[j] = Vector2Point2D(vecbuff[j]);

}

drawPolygon(p, obyek.fc[i].numofVertices);

}

}

}

void makeKerucut(Object3D\_t& kerucut, int n, float h, float r)

{

float a = 6.28 /20;

int i;

kerucut.pnt[0].x = 0;

kerucut.pnt[0].y = h;

kerucut.pnt[0].z = 0;

for (i = 1; i <= n; i++) {

kerucut.pnt[i].x = r\*cos(i\*a);

kerucut.pnt[i].y = 0;

kerucut.pnt[i].z = r\*sin(i\*a);

}

for (i = 0; i<n; i++) {

kerucut.fc[i].numofVertices = 3;

kerucut.fc[i].pnt[0] = 0;

kerucut.fc[i].pnt[1] = i + 2;

kerucut.fc[i].pnt[2] = i + 1;

if (i == (n - 1)) kerucut.fc[i].pnt[1] = 1;

}

kerucut.fc[n].numofVertices = n;

for (i = 0; i<n; i++) kerucut.fc[n].pnt[i] = i + 1;

kerucut.numofVertices = n + 1;

kerucut.numofFaces = n + 1;

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(1000, timer, 0);

}

void Initialize()

{

glClearColor(0.5, 0.5, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1 \* (WIDTH / 2), WIDTH / 2, -1 \* (HEIGHT / 2), HEIGHT / 2);

}

float sudut = 0.0;

void userdraw(void)

{

Matrix3D\_t tilting = rotationX(170)\*rotationY(sudut);

setColor(0, 1, 0);

Object3D\_t kerucut;

makeKerucut(kerucut, 20, 100, 50);

setColor(1, 1, 1);

draw3D(kerucut, tilting);

sudut++;

if (sudut >= 360.0) sudut = 0.0;

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

userdraw();

glutSwapBuffers();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(400, 400);

glutInitWindowPosition(200, 200);

glutCreateWindow("Kerucut");

Initialize();

glutDisplayFunc(display);

glutTimerFunc(1, timer, 0);

glutMainLoop();

return 0;

}

Hasil running

