CG Lab 7

SAHIL BONDRE: U18CO021

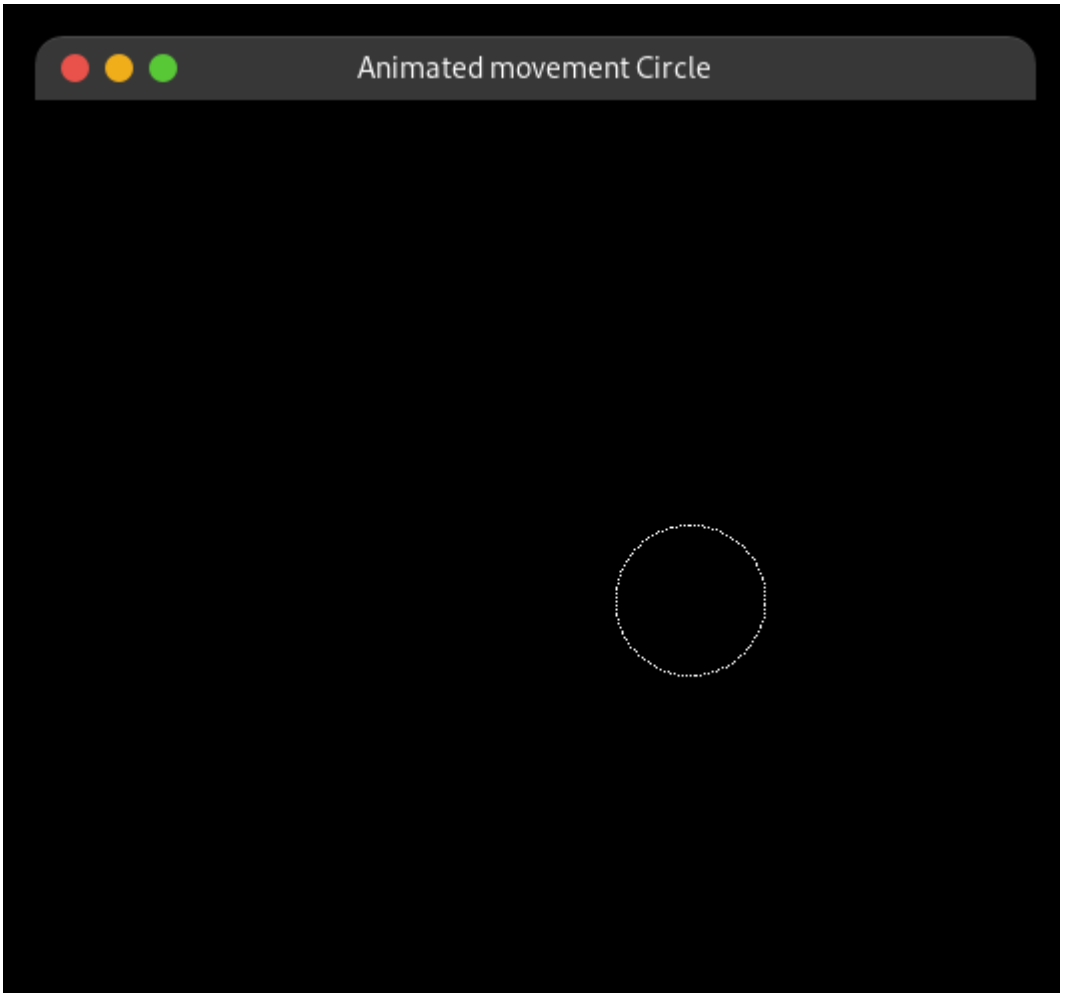
1. Write programs for designing simple animations using 2D transformation Concepts.
    - Circle moving from left to right and vice versa
    - Wind mill rotation
    - Man walking
    - Simple animation of football goal

q1a:

```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <unordered_map>
#include <vector>

using namespace std;

#define PI 3.14159265
double tx = -80;
bool flag = false; // right to left => false ; left to right => true
double radius = 15;

void put_pixel(double x, double y) { glVertex2d(x, y); }

void drawCircle(double xMiddle) {
  double yMiddle = 0;
  float pi = PI;
  glPointSize(1.0);
  glBegin(GL_POINTS);

  for (float i = 0.0; i <= 2 * pi; i += 0.05)
    put_pixel(xMiddle + (sin(i) * radius), yMiddle + (cos(i) * radius));

  glEnd();
}

void timer(int id) {
  if (tx >= 80)
```

```
    flag = true;
  if (tx <= -80)
    flag = false;

  if (!flag) {
    tx += 32;
    flag = false;

  } else {
    tx -= 32;
    flag = true;
  }
  glutPostRedisplay();
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);

  drawCircle(tx + 15);

  glutTimerFunc(1000, timer, 1);

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void init(void) { glClearColor(0, 0, 0, 0); }

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

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);
```

```
  glutCreateWindow("Animated movement Circle");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```

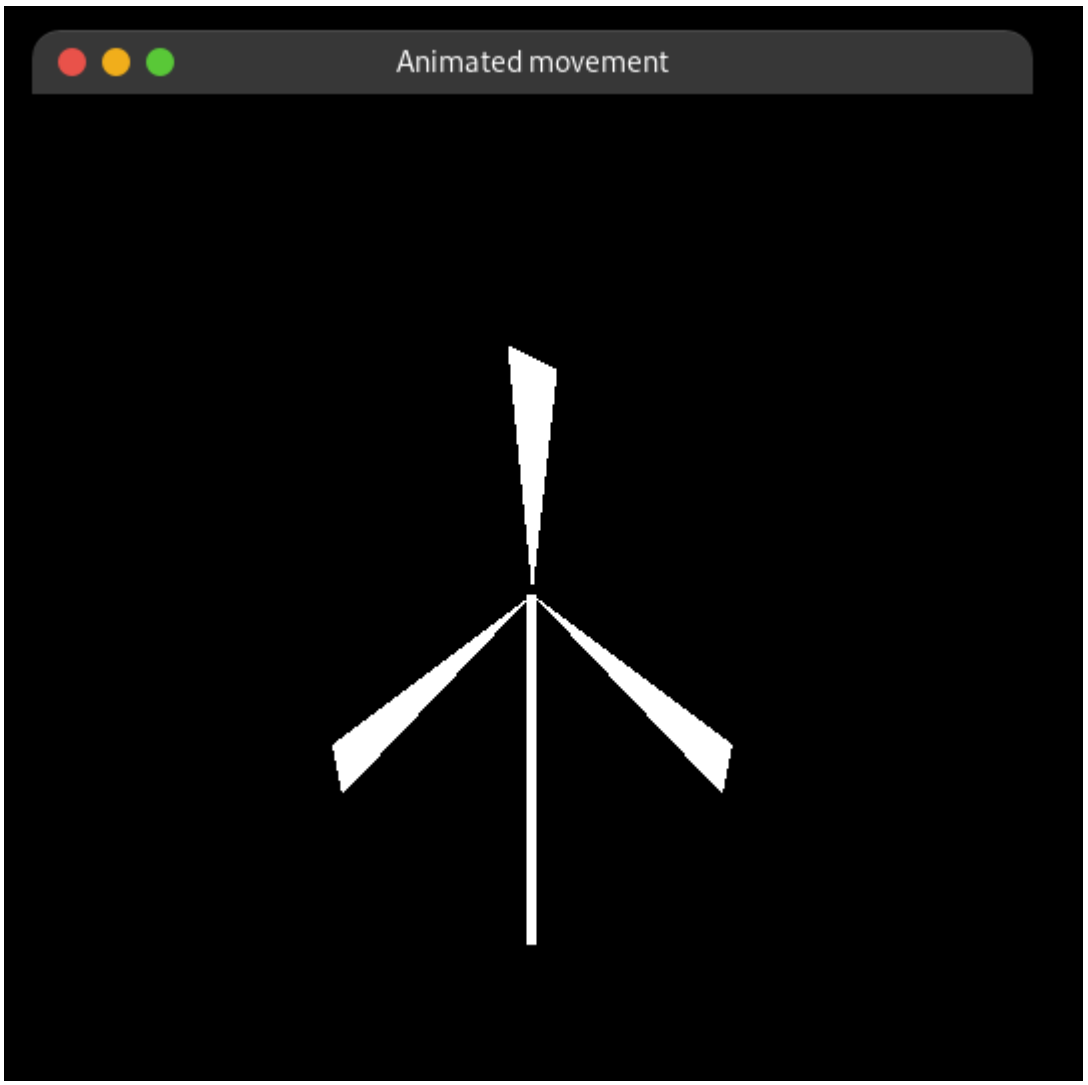


q1b:

```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <unordered_map>
#include <vector>

using namespace std;

#define PI 3.14159265
double degree = 0;

void put_pixel(double x, double y) { glVertex2d(x, y); }

vector<vector<double>> multiplyMatrix(vector<vector<double>> a,
                                      vector<vector<double>> b) {
  vector<vector<double>> resMatrix(3, vector<double>(3, 0));
  for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
      for (int k = 0; k < 3; ++k) {
        resMatrix[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return resMatrix;
}

vector<vector<double>> set_rotate_matrix(double degree) {
  vector<vector<double>> rMatrix{
      {cos(degree * PI / 180), sin(degree * PI / 180), 0},
      {sin(-1 * degree * PI / 180), cos(degree * PI / 180), 0},
      {0, 0, 1}};
  return rMatrix;
}

void drawBladeOne() {
  vector<vector<double>> points = {{-40, -30, 1}, {0, 0, 1}, {-38, -40,
1}};
  vector<vector<double>> outputMatrix = set_rotate_matrix(degree);
  points = multiplyMatrix(points, outputMatrix);

  glBegin(GL_TRIANGLES);

  for (vector<double> point : points)
```

```
    put_pixel(point[0], point[1]);
  glEnd();
}

void drawBladeTwo() {
  vector<vector<double>> points = {
      {40, -30, 1},
      {0, 0, 1},
      {38, -40, 1},
  };

  vector<vector<double>> outputMatrix = set_rotate_matrix(degree);
  points = multiplyMatrix(points, outputMatrix);

  glBegin(GL_TRIANGLES);
  for (vector<double> point : points)
    put_pixel(point[0], point[1]);
  glEnd();
}

void drawBladeThree() {
  vector<vector<double>> points = {{-5, 50, 1}, {0, 0, 1}, {5, 45, 1}};

  vector<vector<double>> outputMatrix = set_rotate_matrix(degree);
  points = multiplyMatrix(points, outputMatrix);

  glBegin(GL_TRIANGLES);
  for (vector<double> point : points)
    put_pixel(point[0], point[1]);
  glEnd();
}

void timer(int id) {
  degree += 30;
  if (degree == 360)
    degree = 0;
  glutPostRedisplay();
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);

  drawBladeOne();
  drawBladeTwo();
```

```
  drawBladeThree();

  glBegin(GL_POLYGON);
  put_pixel(-1, 0);
  put_pixel(1, 0);
  put_pixel(1, -70);
  put_pixel(-1, -70);
  glEnd();

  glutTimerFunc(200, timer, 1);

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void init(void) { glClearColor(0, 0, 0, 0); }

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

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Animated movement");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```


Animated movement

q1c:

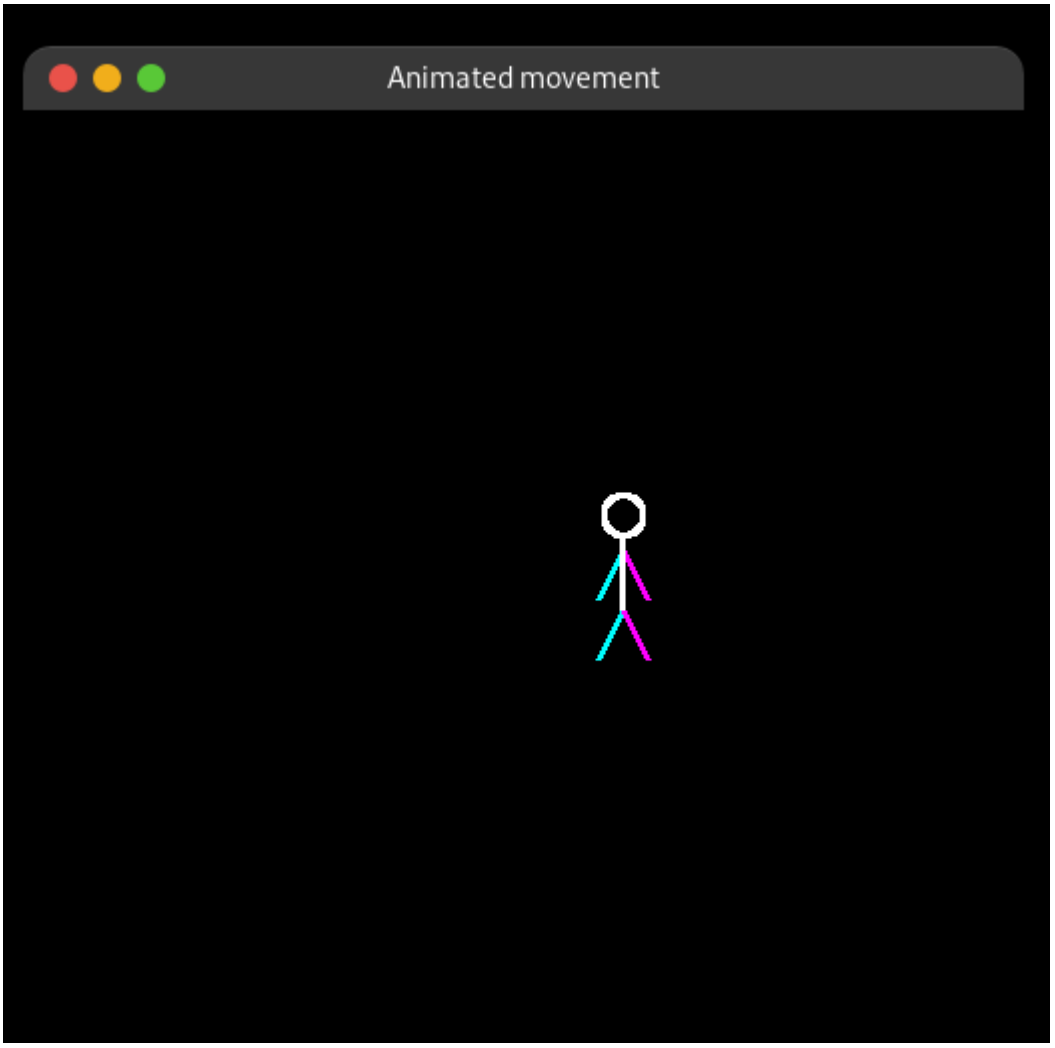
```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <unordered_map>
#include <vector>

using namespace std;

#define PI 3.14159265
double degree = 53.2;
double txCenter = -60;

vector<vector<double>> leftLegPoints = {{-60, 0, 1}, {-65, -10, 1}};

vector<vector<double>> leftHandPoints = {{-60, 12, 1}, {-65, 2, 1}};

vector<vector<double>> rightLegPoints = {{-60, 0, 1}, {-55, -10, 1}};

vector<vector<double>> rightHandPoints = {{-60, 12, 1}, {-55, 2, 1}};

bool leftLegBack = true; // assumption: left leg back, right leg front
initially

void put_pixel(float r, float g, float b, double x, double y) {
  glColor3f(r, g, b);
  glVertex2d(x, y);
}

/*
Flow:
1. Back leg (bl) rotates by 53.2 and coincides with front leg(fl) about
the
txCenter point
2. Translate the bl forward by 10 units in the x-axis
3. Rotate the fl in clockwise direction by 53.2 about its own point
4. We update the new txCenter to txCenter + 10, fl becomes bl and bl
becomes fl
5. We repeat step 1-4 till txCenter < 90
*/

vector<vector<double>> multiplyMatrix(vector<vector<double>> a,
                                      vector<vector<double>> b) {
  vector<vector<double>> resMatrix(a.size(), vector<double>(3, 0));
```

```
  for (int i = 0; i < a.size(); ++i) {
    for (int j = 0; j < 3; ++j) {
      for (int k = 0; k < 3; ++k) {
        resMatrix[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return resMatrix;
}

vector<vector<double>> set_translation_matrix(int dx, int dy) {
  vector<vector<double>> tMatrix{
      {1, 0, 0}, {0, 1, 0}, {(double)dx, (double)dy, 1}};
  return tMatrix;
}

vector<vector<double>> set_rotate_matrix(double degree) {
  vector<vector<double>> rMatrix{
      {cos(degree * PI / 180), sin(degree * PI / 180), 0},
      {sin(-1 * degree * PI / 180), cos(degree * PI / 180), 0},
      {0, 0, 1}};
  return rMatrix;
}

vector<vector<double>> set_rotate_point_matrix(double degree, int px,
int py) {
  // T(-px, -py).R(degree).T(px, py)
  vector<vector<double>> rPointMatrix(3, vector<double>(3, 0));
  rPointMatrix = multiplyMatrix(set_translation_matrix(-1 * px, -1 *
py),
                                set_rotate_matrix(degree));
  rPointMatrix = multiplyMatrix(rPointMatrix, set_translation_matrix(px,
py));
  return rPointMatrix;
}

void walk() {
  vector<vector<double>> step1Matrix =
      set_rotate_point_matrix(degree, txCenter, 0);
  vector<vector<double>> step2Matrix = set_translation_matrix(10, 0);
  vector<vector<double>> step3MatrixLeg =
      set_rotate_point_matrix(-1 * degree, txCenter + 5, -10);

  if (leftLegBack) {

    leftLegPoints = multiplyMatrix(leftLegPoints, step1Matrix);
```

```
    leftLegPoints = multiplyMatrix(leftLegPoints, step2Matrix);
    rightLegPoints = multiplyMatrix(rightLegPoints, step3MatrixLeg);

    leftLegBack = false;
  } else {
    rightLegPoints = multiplyMatrix(rightLegPoints, step1Matrix);
    rightLegPoints = multiplyMatrix(rightLegPoints, step2Matrix);
    leftLegPoints = multiplyMatrix(leftLegPoints, step3MatrixLeg);

    leftLegBack = true;
  }

  txCenter += 10;
  vector<vector<double>> step4Matrix = set_translation_matrix(0, 12);
  leftHandPoints = multiplyMatrix(leftLegPoints, step4Matrix);
  rightHandPoints = multiplyMatrix(rightLegPoints, step4Matrix);
}

void drawLegs() {
  // left leg
  glBegin(GL_LINES);

  for (auto point : leftLegPoints)
    put_pixel(0, 1, 1, point[0], point[1]);
  glEnd();

  // right leg
  glBegin(GL_LINES);
  for (auto point : rightLegPoints)
    put_pixel(1, 0, 1, point[0], point[1]);
  glEnd();
}

void drawHands() {
  // left hand
  glBegin(GL_LINES);

  for (auto point : leftHandPoints)
    put_pixel(0, 1, 1, point[0], point[1]);
  glEnd();

  // right right
  glBegin(GL_LINES);
  for (auto point : rightHandPoints)
    put_pixel(1, 0, 1, point[0], point[1]);
  glEnd();
```

```
}

void drawCircle(double xMiddle) {
  double yMiddle = 19;
  float pi = PI;
  glPointSize(3.0);
  glBegin(GL_POINTS);

  for (float i = 0.0; i <= 2 * pi; i += 0.05)
    put_pixel(1, 1, 1, xMiddle + (sin(i) * 4), yMiddle + (cos(i) * 4));

  glEnd();
}

void timer(int id) {
  if (txCenter <= 75)
    walk();
  glutPostRedisplay();
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);
  glLineWidth(3);

  drawLegs();
  drawHands();

  glBegin(GL_LINES);
  put_pixel(1, 1, 1, txCenter, 0);
  put_pixel(1, 1, 1, txCenter, 15);
  glEnd();

  drawCircle(txCenter);

  glutTimerFunc(1000, timer, 1);

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

```
  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void init(void) { glClearColor(0, 0, 0, 0); }

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

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Animated movement");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```


Animated movement

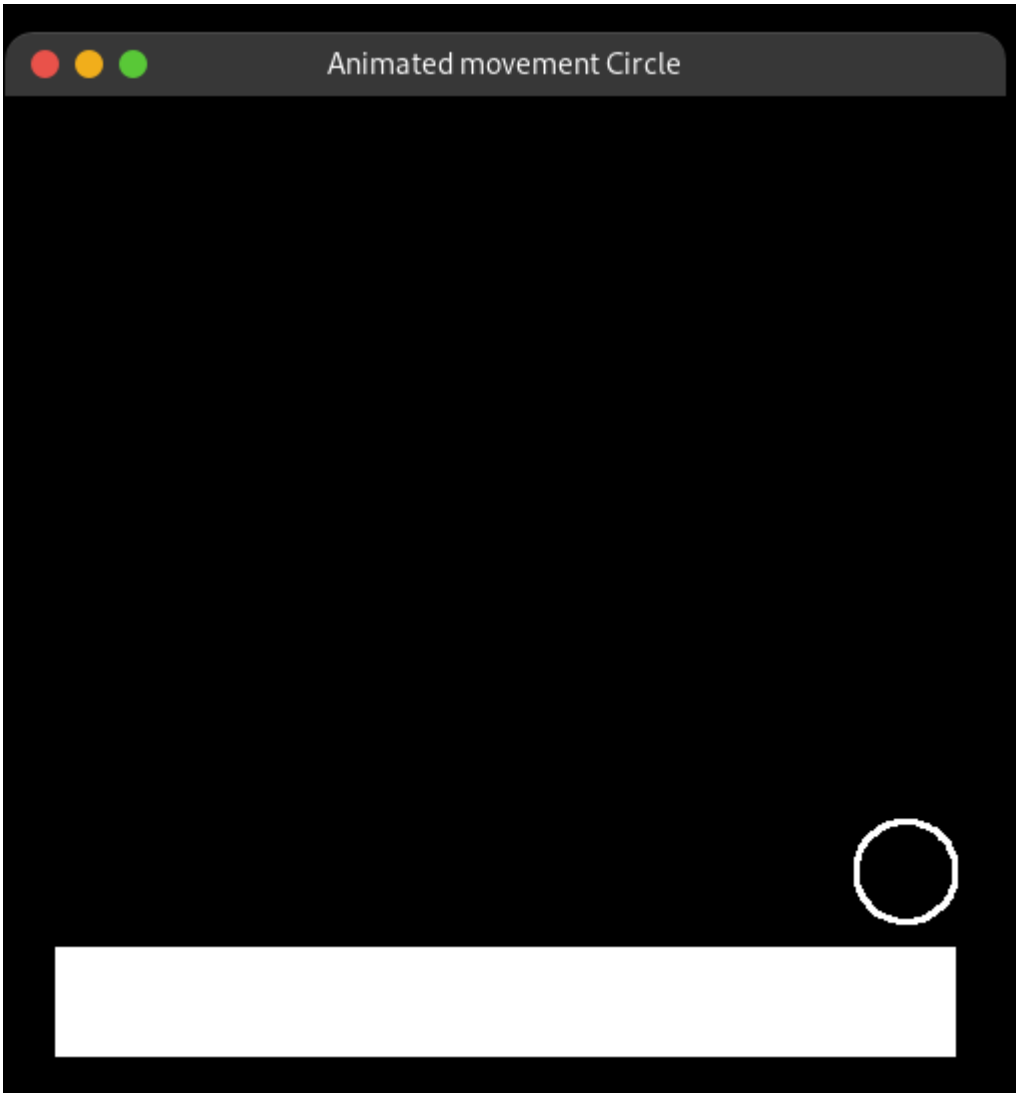
q1d:

```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <unordered_map>
#include <vector>

using namespace std;

#define PI 3.14159265
double degree = 45;
double xCoord = -80;
double yCoord = 25;

/*
Flow:
1. Rotate the  center of circle about the origin by 45 degree
2. Continue rotating till the y > 0
3. Once, y <= 0 just follow translation in the y axis in the down
direction
4. Continue step 3 till y > -70
5. Once y <= -70, remove the circle
*/

void put_pixel(double x, double y) { glVertex2d(x, y); }

vector<vector<double>> multiplyMatrix(vector<vector<double>> a,
                                      vector<vector<double>> b) {
  vector<vector<double>> resMatrix(a.size(), vector<double>(3, 0));
  for (int i = 0; i < a.size(); ++i) {
    for (int j = 0; j < 3; ++j) {
      for (int k = 0; k < 3; ++k) {
        resMatrix[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return resMatrix;
}

vector<vector<double>> set_translation_matrix(int dx, int dy) {
  vector<vector<double>> tMatrix{
      {1, 0, 0}, {0, 1, 0}, {(double)dx, (double)dy, 1}};
  return tMatrix;
}
```

```
vector<vector<double>> set_rotate_matrix(double degree) {
  vector<vector<double>> rMatrix{
      {cos(degree * PI / 180), sin(degree * PI / 180), 0},
      {sin(-1 * degree * PI / 180), cos(degree * PI / 180), 0},
      {0, 0, 1}};
  return rMatrix;
}

void drawGoal() {
  glBegin(GL_POLYGON);
  put_pixel(-90, -92);
  put_pixel(90, -92);
  put_pixel(90, -70);
  put_pixel(-90, -70);
  glEnd();
}

void drawBall(double x, double y) {
  double pi = PI;
  float radius = 10;
  glPointSize(3.0);
  glBegin(GL_POINTS);

  for (float i = 0.0; i <= 2 * pi; i += 0.05)
    put_pixel(x + (sin(i) * radius), y + (cos(i) * radius));

  glEnd();
}

void goalAnimation() {
  vector<vector<double>> point = {{xCoord, yCoord, 1}};

  if (yCoord > 0) {
    vector<vector<double>> mat = set_rotate_matrix(-1 * degree);
    point = multiplyMatrix(point, mat);

    xCoord = point[0][0];
    yCoord = point[0][1];
  } else if (yCoord <= 0 && yCoord > -70) {
    vector<vector<double>> mat = set_translation_matrix(0, -10);
    point = multiplyMatrix(point, mat);

    xCoord = point[0][0];
    yCoord = point[0][1];
  } else {
```

```
    xCoord = -80;
    yCoord = 25;
  }
}

void timer(int id) {
  goalAnimation();
  glutPostRedisplay();
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);

  drawBall(xCoord, yCoord);
  drawGoal();

  glutTimerFunc(300, timer, 1);

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void init(void) { glClearColor(0, 0, 0, 0); }

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

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Animated movement Circle");
  init();
  glutDisplayFunc(display);
```

```
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```



2. Write a menu driven program to implement set of composite transformations on 3D polygon. Program should include:

a. Rotation (about arbitrary point, arbitrary axis,arbitrary plane)
b. Scaling (fixed point)
c. Shearing
d. Reflection

Also show all the principle axis and other relevant point and lines necessary in each choice.

```
#include <GL/glut.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <unordered_set>
using namespace std;
typedef float Matrix4x4[4][4];
Matrix4x4 theMatrix;
float ptsIni[8][3] = {{80, 80, -100},  {180, 80, -100}, {180, 180,
-100},
                      {80, 180, -100}, {60, 60, 0},     {160, 60, 0},
                      {160, 160, 0},   {60, 160, 0}};
// Realign above line while execution
// Initial Co-ordinates ofthe Cube to be Transformed
int flag = 0;
float ptsFin[8][3];
float refptX, refptY, refptZ;              // Reference points
float TransDistX, TransDistY, TransDistZ; // Translations along Axes
float ScaleX, ScaleY, ScaleZ;              // Scaling Factors along Axes
float Alpha, Beta, Gamma, Theta;           // Rotation angles about Axes
float A, B, C;                             // Arbitrary Line Attributes
float aa, bb, cc;
float shx, shy, shz; // Arbitrary Line Attributes
float x1, y11, z1, x2, y2, z2;
int choiceRot, choiceRef, choiceSh;
unordered_set<int> choice;
void matrixSetIdentity(Matrix4x4 m) // Initialises the matrix as Unit
Matrix
{
  int i, j;
  for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
      m[i][j] = (i == j);
}

void matrix_pre_multiply(
    Matrix4x4 a,
    Matrix4x4 b) { // Multiplies matrix a times b, putting result in b
  int i, j;
  Matrix4x4 tmp;
  for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
      tmp[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] *
b[2][j] +
                  a[i][3] * b[3][j];
```

```
    }
  }
  for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
      theMatrix[i][j] = tmp[i][j];
    }
  }
}
void translate(int tx, int ty, int tz) {
  Matrix4x4 m;
  matrixSetIdentity(m);
  m[0][3] = tx;
  m[1][3] = ty;
  m[2][3] = tz;
  matrix_pre_multiply(m, theMatrix);
}
void scale(float sx, float sy, float sz) {
  Matrix4x4 m;
  matrixSetIdentity(m);
  m[0][0] = sx;
  m[1][1] = sy;
  m[2][2] = sz;
  matrix_pre_multiply(m, theMatrix);
}
void shearx() {
  Matrix4x4 m;
  matrixSetIdentity(m);
  m[0][1] = shy;
  m[0][2] = shz;
  matrix_pre_multiply(m, theMatrix);
}
void sheary() {
  Matrix4x4 m;
  matrixSetIdentity(m);
  m[1][0] = shx;
  m[1][2] = shz;
  matrix_pre_multiply(m, theMatrix);
}
void shearz() {
  Matrix4x4 m;
  matrixSetIdentity(m);
  m[2][0] = shx;
  m[2][1] = shy;
  matrix_pre_multiply(m, theMatrix);
}
void RotateX(float angle) {
```

```
  Matrix4x4 m;
  matrixSetIdentity(m);
  angle = angle * 22 / 1260;
  m[1][1] = cos(angle);
  m[1][2] = -sin(angle);
  m[2][1] = sin(angle);
  m[2][2] = cos(angle);
  matrix_pre_multiply(m, theMatrix);
}
void RotateY(float angle) {
  Matrix4x4 m;
  matrixSetIdentity(m);
  angle = angle * 22 / 1260;
  m[0][0] = cos(angle);
  m[0][2] = sin(angle);
  m[2][0] = -sin(angle);
  m[2][2] = cos(angle);
  matrix_pre_multiply(m, theMatrix);
}
void RotateZ(float angle) {
  Matrix4x4 m;
  matrixSetIdentity(m);
  angle = angle * 22 / 1260;
  m[0][0] = cos(angle);
  m[0][1] = -sin(angle);
  m[1][0] = sin(angle);
  m[1][1] = cos(angle);
  matrix_pre_multiply(m, theMatrix);
}
void Reflect(void) {
  Matrix4x4 m;
  matrixSetIdentity(m);
  switch (choiceRef) {
  case 1:
    m[2][2] = -1;
    break;
  case 2:
    m[0][0] = -1;
    break;
  case 3:
    m[1][1] = -1;
    break;
  }
  matrix_pre_multiply(m, theMatrix);
}
void DrawRotLine(void) {
```

```
  switch (choiceRot) {
  case 1:
    glBegin(GL_LINES);
    glVertex3s(-1000, B, C);
    glVertex3s(1000, B, C);
    glEnd();
    break;
  case 2:
    glBegin(GL_LINES);
    glVertex3s(A, -1000, C);
    glVertex3s(A, 1000, C);
    glEnd();
    break;
  case 3:
    glBegin(GL_LINES);
    glVertex3s(A, B, -1000);
    glVertex3s(A, B, 1000);
    glEnd();
    break;
  case 4:
    glBegin(GL_LINES);
    glVertex3s(x1 - aa * 500, y11 - bb * 500, z1 - cc * 500);
    glVertex3s(x2 + aa * 500, y2 + bb * 500, z2 + cc * 500);
    glEnd();
    break;
  }
}
void TransformPoints(void) {
  int i, k;
  float tmp;
  for (k = 0; k < 8; k++)
    for (i = 0; i < 3; i++)
      ptsFin[k][i] = theMatrix[i][0] * ptsIni[k][0] +
                     theMatrix[i][1] * ptsIni[k][1] +
                     theMatrix[i][2] * ptsIni[k][2] + theMatrix[i][3];
  // Realign above line while execution
}
void Axes(void) {
  glColor3f(0.0, 0.0, 0.0); // Set the color to BLACK
  glBegin(GL_LINES);        // Plotting X-Axis
  glVertex2s(-1000, 0);
  glVertex2s(1000, 0);
  glEnd();
  glBegin(GL_LINES); // Plotting Y-Axis
  glVertex2s(0, -1000);
  glVertex2s(0, 1000);
```

```
  glEnd();
}
void Draw(float a[8][3]) // Display the Figure
{
  int i;
  glColor3f(1.0, 0.5, 1.0);
  glBegin(GL_POLYGON);
  glVertex3f(a[0][0], a[0][1], a[0][2]);
  glVertex3f(a[1][0], a[1][1], a[1][2]);
  glVertex3f(a[2][0], a[2][1], a[2][2]);
  glVertex3f(a[3][0], a[3][1], a[3][2]);
  glEnd();
  i = 0;
  glColor3f(1.0, 0.6, 0.5);
  glBegin(GL_POLYGON);
  glVertex3s(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
  glVertex3s(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
  glVertex3s(a[5 + i][0], a[5 + i][1], a[5 + i][2]);
  glVertex3s(a[4 + i][0], a[4 + i][1], a[4 + i][2]);
  glEnd();
  glColor3f(0.2, 0.4, 1.0);
  glBegin(GL_POLYGON);
  glVertex3f(a[0][0], a[0][1], a[0][2]);
  glVertex3f(a[3][0], a[3][1], a[3][2]);
  glVertex3f(a[7][0], a[7][1], a[7][2]);
  glVertex3f(a[4][0], a[4][1], a[4][2]);
  glEnd();
  i = 1;
  glColor3f(0.5, 0.4, 0.3);
  glBegin(GL_POLYGON);
  glVertex3s(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
  glVertex3s(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
  glVertex3s(a[5 + i][0], a[5 + i][1], a[5 + i][2]);
  glVertex3s(a[4 + i][0], a[4 + i][1], a[4 + i][2]);
  glEnd();
  i = 2;
  glColor3f(0.5, 0.6, 0.2);
  glBegin(GL_POLYGON);
  glVertex3s(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
  glVertex3s(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
  glVertex3s(a[5 + i][0], a[5 + i][1], a[5 + i][2]);
  glVertex3s(a[4 + i][0], a[4 + i][1], a[4 + i][2]);
  glEnd();
  i = 4;
  glColor3f(1.0, 0.3, 0.4);
  glBegin(GL_POLYGON);
```

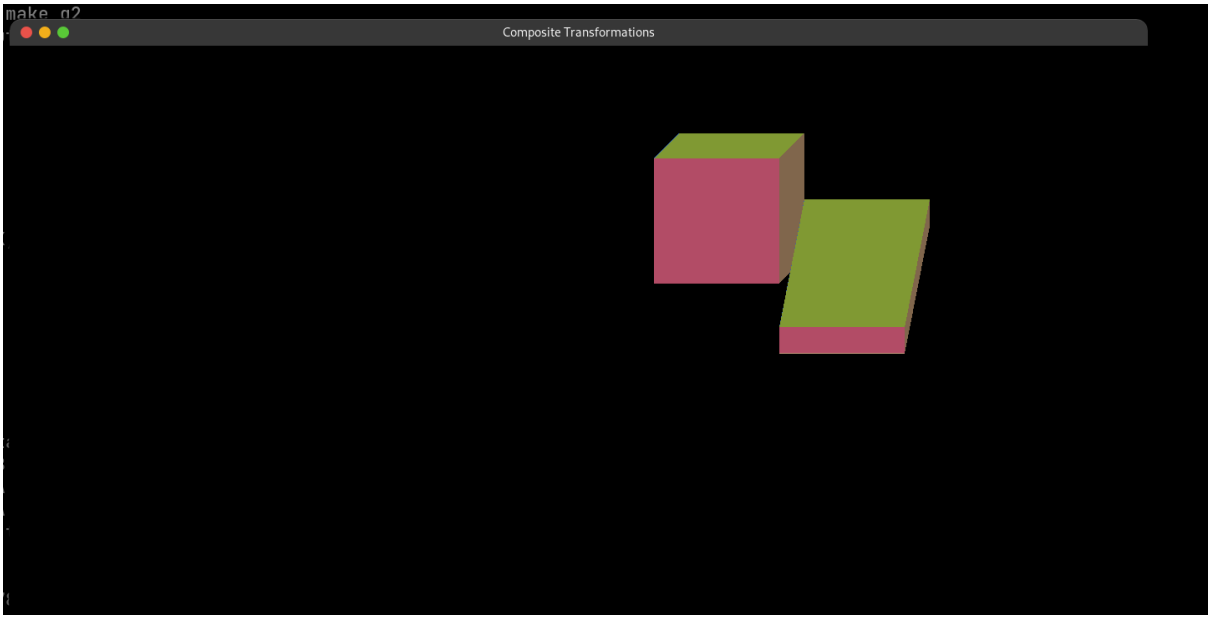
```
  glVertex3f(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
  glVertex3f(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
  glVertex3f(a[2 + i][0], a[2 + i][1], a[2 + i][2]);
  glVertex3f(a[3 + i][0], a[3 + i][1], a[3 + i][2]);
  glEnd();
}

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  Axes();
  glColor3f(1.0, 0.0, 0.0); // Set the color to RED
  Draw(ptsIni);
  matrixSetIdentity(theMatrix);
  if (choice.find(1) != choice.end()) {
    translate(TransDistX, TransDistY, TransDistZ);
  }
  if (choice.find(2) != choice.end()) {
    scale(ScaleX, ScaleY, ScaleZ);
  }
  if (choice.find(3) != choice.end()) {
    switch (choiceRot) {
    case 1:
      DrawRotLine();
      translate(0, -B, -C);
      RotateX(Alpha);
      translate(0, B, C);
      break;
    case 2:
      DrawRotLine();
      translate(-A, 0, -C);
      RotateY(Beta);
      translate(A, 0, C);
      break;
    case 3:
      DrawRotLine();
      translate(-A, -B, 0);
      RotateZ(Gamma);
      translate(A, B, 0);
      break;
    case 4:
      DrawRotLine();
      float MOD = sqrt((x2 - x1) * (x2 - x1) + (y2 - y11) * (y2 - y11) +
                       (z2 - z1) * (z2 - z1));
      aa = (x2 - x1) / MOD;
      bb = (y2 - y11) / MOD;
      cc = (z2 - z1) / MOD;
```

```
      translate(-x1, -y11, -z1);
      float ThetaDash;
      ThetaDash = 1260 * atan(bb / cc) / 22;
      RotateX(ThetaDash);
      RotateY(1260 * asin(-aa) / 22);
      RotateZ(Theta);
      RotateY(1260 * asin(aa) / 22);
      RotateX(-ThetaDash);
      translate(x1, y11, z1);
      break;
    }
  }
  if (choice.find(4) != choice.end()) {
    Reflect();
  }
  if (choice.find(5) != choice.end()) {

    if (choiceSh == 1) {
      shearx();
    } else if (choiceSh == 2) {
      sheary();
    } else {
      shearz();
    }
  }
  TransformPoints();
  Draw(ptsFin);
  glFlush();
}
void init(void) {
  glClearColor(0.0, 0.0, 0.0, 0.0);
  // Set the Background color to WHITE
  glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);
  // Set the no. of Co-ordinates along X & Y axes and their gappings
  glEnable(GL_DEPTH_TEST);
  // To Render the surfaces Properly according to their depths
}

int main(int argc, char **argv) {
  while (true) {
    printf("Enter your choice "

"number:\n1.Translation\n2.Scaling\n3.Rotation\n4.Reflection\n5."
           "Shearing\n: ");
    int code;
    cin >> code;
```

```
    choice.insert(code);
    if (code == 1) {
      printf("Enter Translation along X, Y & Z\n: ");
      scanf("%f%f%f", &TransDistX, &TransDistY, &TransDistZ);
    } else if (code == 2) {
      printf("Enter Scaling ratios along X, Y & Z\n: ");
      scanf("%f%f%f", &ScaleX, &ScaleY, &ScaleZ);
    } else if (code == 3) {
      printf("Enter your choice for Rotation about axis:\n1.parallel to
"
             "X-axis.(y=B & z=C)\n2.parallel to Y-axis.(x=A &
z=C)\n3.parallel "
             "to Z-axis.(x=A & y=B)\n4.Arbitrary line passing through "
             "(x1,y1,z1) & (x2,y2,z2)\n: ");
      // Realign above line while execution
      scanf("%d", &choiceRot);

      if (choiceRot == 1) {
        printf("Enter B & C: ");
        scanf("%f %f", &B, &C);
        printf("Enter Rot. Angle Alpha: ");
        scanf("%f", &Alpha);
      } else if (choiceRot == 2) {
        printf("Enter A & C: ");
        scanf("%f %f", &A, &C);
        printf("Enter Rot. Angle Beta: ");
        scanf("%f", &Beta);
      } else if (choiceRot == 3) {
        printf("Enter A & B: ");
        scanf("%f %f", &A, &B);
        printf("Enter Rot. Angle Gamma: ");
        scanf("%f", &Gamma);
      } else if (choiceRot == 4) {
        printf("Enter values of x1 ,y1 & z1:\n");
        scanf("%f %f %f", &x1, &y11, &z1);
        printf("Enter values of x2 ,y2 & z2:\n");
        scanf("%f %f %f", &x2, &y2, &z2);
        printf("Enter Rot. Angle Theta: ");
        scanf("%f", &Theta);
      } else {
        cout << "Invalid option opted.";
      }

    } else if (code == 4) {
      printf("Enter your choice for reflection about "
             "plane:\n1.X-Y\n2.Y-Z\n3.X-Z\n: ");
```

```
      scanf("%d", &choiceRef);
    } else if (code == 5) {
      cout << "Enter your choice\n1.Shear X \n 2.Shear Y\n3.ShearZ\n";
      cin >> choiceSh;
      if (choiceSh == 1) {
        cout << "Enter shy and shz:";
        cin >> shy >> shz;
      } else if (choiceSh == 2) {
        cout << "Enter shx and shz:";
        cin >> shx >> shz;
      } else if (choiceSh == 3) {
        cout << "Enter shx and shy:";
        cin >> shx >> shy;
      } else {
        cout << "Invalid option opted";
      }
    } else {
      break;
    }
  }
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(1362, 750);
  glutInitWindowPosition(0, 0);
  glutCreateWindow(" Composite Transformations ");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

```
➜  lab-07 git:(master) ✗ ./a.out
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
: 1
Enter Translation along X, Y & Z
: 100 10 10
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
: 3
Enter your choice for Rotation about axis:
1.parallel to X-axis.(y=B & z=C)
2.parallel to Y-axis.(x=A & z=C)
3.parallel to Z-axis.(x=A & y=B)
4.Arbitrary line passing through (x1,y1,z1) & (x2,y2,z2)
: 1
Enter B & C: 0 0
Enter Rot. Angle Alpha: 78
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
: q
```


make g2
Composite Transformations