# q6

Thursday, May 02, 2024 9:09 PM

6) Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates).

========== reflection about an arbitary axis==========

```
#include  <iostream>
#include  <conio.h>
#include  <math.h>


#include "dda.cpp"
#include <graphics.h>


using namespace std;




void matrixMultiplication(double **mat1, int rows1, int cols1,
                          double **mat2, int rows2, int cols2,
                          double **result) {

   if (cols1 != rows2) {
      std::cerr << "Error: Matrix dimensions \n";
      return;
   }



   for (int i = 0; i < rows1; ++i) {
      for (int j = 0; j < cols2; ++j) {
         result[i][j] = 0;
         for (int k = 0; k < cols1; ++k) {
            result[i][j] += mat1[i][k] * mat2[k][j];
         }
      }
   }
}




void print_sq_arr(double **arr ,int row, int col){



for(int i=0;i<row;i++){
      for(int j=0;j<col;j++){
         cout<<arr[i][j]<<" , ";
      }
      cout<<endl;
   }


}


double **make_hom_2d_array_zero(int row,int col){
    double **arr = new double*[row];


   for(int i=0;i<row;i++){
      arr[i] = new double[col];
   }
for(int i=0;i<row;i++){
```

```
        for(int j=0;j<col;j++){
            arr[i][j]=0.0;
        }
    }
    return &*arr;
}


double **make_hom_2d_array_ones(int row,int col){
     double **arr = new double*[row];


    for(int i=0;i<row;i++){
        arr[i] = new double[col];
    }
for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            arr[i][j]=1.0;
        }
    }
    return &*arr;
}


void fill_composite_matrix(double **arr,double m, double c){


    arr[0][0] =  (1 - pow(m,2) )/ (1 + pow(m,2));
    arr[1][0] =  (2*m  )/ (1 + pow(m,2));
    arr[2][0] =  (-2*c*m )/ (1 + pow(m,2));


    arr[0][1] =  (2*m )/ (1 + pow(m,2));
    arr[1][1] =  ( pow(m,2) -1 )/ (1 + pow(m,2));
    arr[2][1] =  (2*c )/ (1 + pow(m,2));


    arr[2][2] = 1;


}


void draw_square(double **arr,int r,int c,int colorr){
    dda(arr[0][0], arr[0][1], arr[1][0], arr[1][1],colorr);
    dda( arr[2][0], arr[2][1],arr[1][0], arr[1][1],colorr); //200, 100   200, 0
    dda(arr[2][0], arr[2][1], arr[3][0], arr[3][1],colorr);
    dda( arr[0][0], arr[0][1],arr[3][0], arr[3][1],colorr);
}


void draw_mirror(double m, double c){  // bcoz it fails at vertical line


    double y1=0, x1=0, y2=0, x2=0;
    if(m !=0){
        y1 = m*(0) + c;
        y2 = m*(getmaxx()) + c;
        x1 =   (y1 - c)/m;
        x2 =   (y2 - c)/m;
    }else{


            x1 = 0;
            x2 = getmaxx();
            y1 = c;
            y2 = c;


    }
    dda(x1,y1,x2,y2,GREEN);


}
```

```
int main()
{




 int gd = DETECT, gm;
 char pathtodriver[] = "";
initgraph(&gd, &gm, pathtodriver);


double **mat = make_hom_2d_array_zero(3,3);


double m =1.0;
double c =-20.0;


draw_mirror(m,c);   // 0 for horizontal , 1 for vertical , y = 200


fill_composite_matrix(mat,m,c);


int row = 4;
double **arr = make_hom_2d_array_ones(row,3);


//x
arr[0][0] = 100;
arr[1][0] = 200;
arr[2][0] = 200;
arr[3][0] = 100;


//y
arr[0][1] =0;
arr[1][1] =0;
arr[2][1] =100;
arr[3][1] =100;


draw_square(arr,row,3,GREEN);




double **res = make_hom_2d_array_zero(row,3);


matrixMultiplication(arr,row,3,mat,3,3,res);
draw_square(res,row,3,BLUE);


// will fail only for vertical line


   getch();
   closegraph();


return 0;


}
```

```
//================================= ROTATION. + translation ====================
#include <iostream>
#include <conio.h>
#include <math.h>
#include "dda.cpp"


#include <graphics.h>
using namespace std;




#define M_PI 3.14159265358979323846


class my_matrix{


   public:


   double** matrix ;
   int row;
   int col;


   my_matrix(){
   }


   my_matrix(int r,int v){


      row = r;
      col = 3; // for homogeneous


      matrix = new double*[r];


       // Allocate memory for each row
         for (int i = 0; i < r; ++i) {
            matrix[i] = new double[3]; // Create an array of integers for each row
         }


           // Initialize each element to 1
         for (int i = 0; i < r; ++i) {
            for (int j = 0; j < 3; ++j) {
               matrix[i][j] = v;
            }
         }




   }


   void print_obj(){


  cout<<"-------------------------------------OBJECT-------------------------------------\n";
   for (int i = 0; i < row; ++i) {
      for (int j = 0; j < col; ++j) {
         std::cout << matrix[i][j] << " ";
      }
      std::cout << std::endl;
   }
```

```
    cout<<"--------------------------------------------------------------------------
------------------\n";
    }


    void deallocate_matrix(){
        // Free dynamically allocated memory
    for (int i = 0; i < row; ++i) {
       delete[] matrix[i];
    }
    delete[] matrix;
    }
};

double angle_to_radian(double degrees){
  return degrees * (M_PI / 180.0);
}


void draw_object(my_matrix *obj,int colorr){
    int x1=0;
    int y1=0;


    int x2=0;
    int y2=0;
      x1 = obj->matrix[0][0];
      y1 = obj->matrix[0][1];


     for (int i = 1; i < obj->row; ++i) {


            x2 = obj->matrix[i][0];
            y2 = obj->matrix[i][1];


            dda(x1,y1,x2,y2,colorr);


            x1 = x2;
            y1 = y2;


            // draw and switch
    }


    dda(x1,y1, obj->matrix[0][0], obj->matrix[0][1],colorr);
}







void create_rotation_matrix(double degrees, my_matrix *obj){   // angle is
in degree , can be used -90 degree


    obj->matrix[0][0] = round(cos(angle_to_radian(degrees)) * 10000) /
10000;
    obj->matrix[1][1] = round(cos(angle_to_radian(degrees)) * 10000) /
10000 ;
    obj->matrix[0][1] = round(sin(angle_to_radian(degrees))* 10000) /
10000  ;
    obj->matrix[1][0] = round(-sin(angle_to_radian(degrees)) * 10000) /
10000;
    obj->matrix[2][2] = 1;



}
```

```
void matrix_multiplication(my_matrix *obj , my_matrix *scaling ,
my_matrix *result){
     // Function to perform matrix multiplication
      if (obj->col != scaling->row) {
         std::cerr << "Error: Matrix dimensions mismatch for
multiplication." << std::endl;
         exit(1);
      }


      for (int i = 0; i < obj->row; ++i) {
         for (int j = 0; j < scaling ->col; ++j) {
            result -> matrix[i][j] = 0;


            for (int k = 0; k < obj -> col ; ++k) {
               result -> matrix[i][j] +=  obj->matrix[i][k] * scaling -> matrix[k]
[j];
            }


         }
      }
}


void create_translation_matrix(int tx,int ty, my_matrix *obj){
   obj->matrix[0][0] = 1;
   obj->matrix[1][1] = 1;
   obj->matrix[2][2] = 1;
   obj->matrix[2][0] = tx;
   obj->matrix[2][1] = ty;
}




int main(){


   int r = 4 ;
 //  cout<<"\ngive the number of points : ";
  // cin>>r;
    my_matrix *obj;


if(r>=1){
   cout<<"matrix made"<<endl;
   obj  = new my_matrix(r,1);
   obj->print_obj();
}else{
   cout<<"invalid input";
   return 0;
}


// int temp  =0;
// double x =0;
// double y = 0;




// while(temp < r){

//     cout<<"x , y : ";
//     cin>>x>>y;


//     obj->matrix[temp][0] = x;
//     obj->matrix[temp][1] = y;
```

```
// temp +=1;



// }



//x
obj->matrix[0][0] = 100;
obj->matrix[1][0] = 200;
obj->matrix[2][0] = 200;
obj->matrix[3][0] = 100;


//y
obj->matrix[0][1] =0;
obj->matrix[1][1] =0;
obj->matrix[2][1] =100;
obj->matrix[3][1] =100;







double degrees = 45.0 ;
// cout<<"\nrotation factor in x : ";
// cin>>degrees;


my_matrix *rotation_matrix = new my_matrix(3,0);
create_rotation_matrix(degrees, rotation_matrix);


 // 4  x 3     3 x 3 -->  4 x 3


//  before rotation translate centroid to center
double x_centroid = 0;
double y_centroid= 0;


for(int i =0;i<obj->row;i++){
    x_centroid  +=  obj->matrix[i][0];
    y_centroid  +=  obj->matrix[i][1];
}


x_centroid = -1*(x_centroid / obj->row);
y_centroid = -1*(y_centroid / obj->row);


my_matrix *translation_mat =  new my_matrix(3,0);


// translation
my_matrix *result1 =  new my_matrix(r,0);
create_translation_matrix(x_centroid , y_centroid,translation_mat);
matrix_multiplication(obj,translation_mat,result1);


//rotation
my_matrix *result2 =  new my_matrix(r,0);
matrix_multiplication(result1,rotation_matrix, result2);


my_matrix *translation_mat_inv =  new my_matrix(3,0);
```

```
create_translation_matrix(-1*x_centroid , -1*y_centroid,
translation_mat_inv);


// inverse translation
my_matrix *result3 =  new my_matrix(r,0);
matrix_multiplication(result2 , translation_mat_inv, result3);




   int gd = DETECT, gm;
   char pathtodriver[] = "";
   initgraph(&gd, &gm, pathtodriver);




   draw_object(obj,GREEN);
   draw_object(result3,RED);
      getch();
      closegraph();


return 0;
}

//==================================.
SCALING=========================


#include <iostream>
#include <conio.h>
#include <math.h>
#include "dda.cpp"


#include <graphics.h>
using namespace std;
class my_matrix{  //    6 x 3 {object , triangle , square , polygon}  3 x 3 {
scaling matrix, scaling}  --> 6 x 3


   public:


   int** matrix ;
   int row;
   int col;


   my_matrix(){


   }


   my_matrix(int r,int v){  // 0, 1 object ,,1 transfo 0


      row = r;
      col = 3; // for homogeneous


      matrix = new int*[r];


       // Allocate memory for each row
         for (int i = 0; i < r; ++i) {
            matrix[i] = new int[3]; // Create an array of integers for each
row
          }


            // Initialize each element to 1
```

```
            for (int i = 0; i < r; ++i) {
                for (int j = 0; j < 3; ++j) {
                    matrix[i][j] = v;
                }
            }
    }


    void print_obj(){


  cout<<"-------------------------------------OBJECT-------------------------
------------------\n";
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < col; ++j) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }

    cout<<"-----------------------------------------------------------------------
------------------\n";
    }


    void deallocate_matrix(){
         // Free dynamically allocated memory
    for (int i = 0; i < row; ++i) {
        delete[] matrix[i];
    }
    delete[] matrix;
    }
};




void create_scaling_matrix(int sx,int sy, my_matrix *obj){
    obj->matrix[0][0] = sx;
    obj->matrix[1][1] = sy;
    obj->matrix[2][2] = 1;
}


// object  x  scaling matrix - -> result
void matrix_multiplication(my_matrix *obj , my_matrix *scaling ,
my_matrix *result){
     // Function to perform matrix multiplication


        if (obj->col != scaling->row) {
            std::cerr << "Error: Matrix dimensions mismatch for
multiplication." << std::endl;
            exit(1);
        }




        for (int i = 0; i < obj->row; ++i) {
            for (int j = 0; j < scaling ->col; ++j) {
                result -> matrix[i][j] = 0;


                for (int k = 0; k < obj -> col ; ++k) {
                    result -> matrix[i][j] +=  obj->matrix[i][k] * scaling -> matrix[k]
[j];
                }


            }
        }

}
```

```
// [x1, y1, 1]
//[ x2, y2, 1]
//[ x3 , y3, 1]
//[x4, y4, 1]


void draw_object(my_matrix *obj,int colorr){
    int x1=0;
    int y1=0;


    int x2=0;
    int y2=0;




      x1 = obj->matrix[0][0];
      y1 = obj->matrix[0][1];


     for (int i = 1; i < obj->row; ++i) {


            x2 = obj->matrix[i][0];
            y2 = obj->matrix[i][1];


            dda(x1,y1,x2,y2,colorr);


            x1 = x2;
            y1 = y2;


            // draw and switch
    }


    dda(x1,y1, obj->matrix[0][0], obj->matrix[0][1],colorr);
}



int main(){



    int r = 4;
    // cout<<"\ngive the number of points : ";
    // cin>>r;
     my_matrix *obj;


if(r>=1){
    cout<<"matrix made"<<endl;
    obj  = new my_matrix(r,1);
    obj->print_obj();
}else{
    cout<<"invalid input";
    return 0;
}


int temp  =0;
 int x =0;
```

```
 int y = 0;



// while(temp < r){

//     cout<<"x , y : ";
//     cin>>x>>y;


//     obj->matrix[temp][0] = x;
//     obj->matrix[temp][1] = y;


// temp +=1;


// }
//x
//x
obj->matrix[0][0] = 100;
obj->matrix[1][0] = 200;
obj->matrix[2][0] = 200;
obj->matrix[3][0] = 100;


//y
obj->matrix[0][1] =0;
obj->matrix[1][1] =0;
obj->matrix[2][1] =100;
obj->matrix[3][1] =100;


cout<<endl;


int sx = 2 ;
int sy = 2;
// cout<<"scaling factor in x : ";
// cin>>sx;


// cout<<"scaling factor in y : ";
// cin>>sy;




my_matrix *scaling_matrix = new my_matrix(3,0);  // 3 x 3 , default 0
create_scaling_matrix(sx,sy,scaling_matrix);


cout<<"scaling matrix::"<<endl;
scaling_matrix->print_obj();


cout<<"\nyour input :"<<endl;
obj->print_obj();




my_matrix *result =  new my_matrix(r,0);  // 4  x 3     3 x 3 -->  4 x 3


matrix_multiplication(obj,scaling_matrix,result); // pre multiplication


cout<<"============================RESULT============
======================\n";
result->print_obj();
```

```
    int gd = DETECT, gm;
    char pathtodriver[] = "";
    initgraph(&gd, &gm, pathtodriver);


    draw_object(obj,BLUE);
    draw_object(result,GREEN);









getch();
closegraph();


return 0;
}



//============================== SHEAR
=========================================



#include <iostream>
#include <conio.h>
#include <math.h>
#include "dda.cpp"


#include <graphics.h>
using namespace std;


class my_matrix{


    public:


    int** matrix ;
    int row;
    int col;


    my_matrix(){


    }


    my_matrix(int r,int v){


        row = r;
        col = 3; // for homogeneous


        matrix = new int*[r];


         // Allocate memory for each row
           for (int i = 0; i < r; ++i) {
               matrix[i] = new int[3]; // Create an array of integers for each
row
           }
```

```
            // Initialize each element to 1
            for (int i = 0; i < r; ++i) {
                for (int j = 0; j < 3; ++j) {
                    matrix[i][j] = v;
                }
            }



    }


    void print_obj(){


  cout<<"-------------------------------------OBJECT-------------------------
------------------\n";
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < col; ++j) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }

    cout<<"-----------------------------------------------------------------------------
------------------\n";
    }


    void deallocate_matrix(){
         // Free dynamically allocated memory
    for (int i = 0; i < row; ++i) {
        delete[] matrix[i];
    }
    delete[] matrix;
    }



};



void create_shear_matrix(int sx,int sy, my_matrix *obj){
    obj->matrix[0][0] = 1;
    obj->matrix[1][1] = 1;
    obj->matrix[2][2] = 1;


    obj->matrix[0][1] = sy;
    obj->matrix[1][0] = sx;


    // [1 sy 0]
    // [sx 1 0]
    // [0  0 1]


}



void matrix_multiplication(my_matrix *obj , my_matrix *shear_mat ,
my_matrix *result){
      // Function to perform matrix multiplication


        if (obj->col != shear_mat->row) {
```

```
            std::cerr << "Error: Matrix dimensions mismatch for
multiplication." << std::endl;
            exit(1);
        }




        for (int i = 0; i < obj->row; ++i) {
            for (int j = 0; j < shear_mat ->col; ++j) {
                result -> matrix[i][j] = 0;


                for (int k = 0; k < obj -> col ; ++k) {
                    result -> matrix[i][j] +=  obj->matrix[i][k] * shear_mat ->
matrix[k][j];
                }


            }
        }

}




void draw_object(my_matrix *obj,int colorr){
    int x1=0;
    int y1=0;


    int x2=0;
    int y2=0;







      x1 = obj->matrix[0][0];
      y1 = obj->matrix[0][1];


     for (int i = 1; i < obj->row; ++i) {


            x2 = obj->matrix[i][0];
            y2 = obj->matrix[i][1];


            dda(x1,y1,x2,y2,colorr);


            x1 = x2;
            y1 = y2;


            // draw and switch
    }


    dda(x1,y1, obj->matrix[0][0], obj->matrix[0][1],colorr);
}




int main(){




    int r =4 ;
    // cout<<"\ngive the number of points : ";
```

```
    // cin>>r;
     my_matrix *obj;


if(r>=1){
    cout<<"matrix made"<<endl;
    obj  = new my_matrix(r,1);
    obj->print_obj();
}else{
    cout<<"invalid input";
    return 0;
}


int temp  =0;
 int x =0;
 int y = 0;




// while(temp < r){

//     cout<<"x , y : ";
//     cin>>x>>y;


//     obj->matrix[temp][0] = x;
//     obj->matrix[temp][1] = y;


// temp +=1;


// }
//x
obj->matrix[0][0] = 100;
obj->matrix[1][0] = 200;
obj->matrix[2][0] = 200;
obj->matrix[3][0] = 100;


//y
obj->matrix[0][1] =0;
obj->matrix[1][1] =0;
obj->matrix[2][1] =100;
obj->matrix[3][1] =100;
cout<<endl;


int sx = 2 ;
int sy = 1;
// cout<<"shear factor in x : ";
// cin>>sx;


// cout<<"shear factor in y : ";
// cin>>sy;


my_matrix *shear_matrix = new my_matrix(3,0);;
create_shear_matrix(sx,sy,shear_matrix);


cout<<"\nyour input :"<<endl;
obj->print_obj();




my_matrix *result =  new my_matrix(r,0);  // 4  x 3     3 x 3 -->  4 x 3
matrix_multiplication(obj,shear_matrix,result);
```

```
    int gd = DETECT, gm;
    char pathtodriver[] = "";
    initgraph(&gd, &gm, pathtodriver);


    draw_object(obj,BLUE);
    draw_object(result,GREEN);


    result->print_obj();







getch();
closegraph();


return 0;
}




//======================


//6) Write a program to apply various 2D transformations on a 2D object
(use homogenous
//coordinates).


#include <iostream>
#include <conio.h>
#include <math.h>
#include "dda.cpp"


#include <graphics.h>


using namespace std;







class my_matrix{


    public:


    int** matrix ;
    int row;
    int col;


    my_matrix(){


    }


    my_matrix(int r,int v){
```

```
        row = r;
        col = 3; // for homogeneous


        matrix = new int*[r];


         // Allocate memory for each row
           for (int i = 0; i < r; ++i) {
              matrix[i] = new int[3]; // Create an array of integers for each
row
           }


             // Initialize each element to 1
           for (int i = 0; i < r; ++i) {
              for (int j = 0; j < 3; ++j) {
                 matrix[i][j] = v;
              }
           }




    }


    void print_obj(){


  cout<<"--------------------------------------OBJECT--------------------------
------------------\n";
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < col; ++j) {
           std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }

    cout<<"------------------------------------------------------------------------
------------------\n";
    }


    void deallocate_matrix(){
         // Free dynamically allocated memory
    for (int i = 0; i < row; ++i) {
       delete[] matrix[i];
    }
    delete[] matrix;
    }




};



void create_scaling_matrix(int tx,int ty, my_matrix *obj){
    obj->matrix[0][0] = 1;
    obj->matrix[1][1] = 1;
    obj->matrix[2][2] = 1;


    obj->matrix[2][0] = tx;
    obj->matrix[2][1] = ty;
```

```
// [ 1 0 0]
// [ 0 1 0]
// [tx ty 1]



}



void matrix_multiplication(my_matrix *obj , my_matrix *scaling ,
my_matrix *result){
    // Function to perform matrix multiplication


     if (obj->col != scaling->row) {
        std::cerr << "Error: Matrix dimensions mismatch for
multiplication." << std::endl;
        exit(1);
     }




     for (int i = 0; i < obj->row; ++i) {
        for (int j = 0; j < scaling ->col; ++j) {
           result -> matrix[i][j] = 0;


           for (int k = 0; k < obj -> col ; ++k) {
              result -> matrix[i][j] +=  obj->matrix[i][k] * scaling -> matrix[k]
[j];
           }


        }
     }

}




void draw_object(my_matrix *obj,int colorr){
   int x1=0;
   int y1=0;


   int x2=0;
   int y2=0;




    x1 = obj->matrix[0][0];
    y1 = obj->matrix[0][1];


    for (int i = 1; i < obj->row; ++i) {


        x2 = obj->matrix[i][0];
        y2 = obj->matrix[i][1];


        dda(x1,y1,x2,y2,colorr);


        x1 = x2;
        y1 = y2;
```

```
            // draw and switch
        }


        dda(x1,y1, obj->matrix[0][0], obj->matrix[0][1],colorr);
}



int main(){



    int r = 4 ;
    cout<<"\ngive the number of points : ";
   // cin>>r;
     my_matrix *obj;


if(r>=1){
    cout<<"matrix made"<<endl;
    obj  = new my_matrix(r,1);
    obj->print_obj();
}else{
    cout<<"invalid input";
    return 0;
}


int temp  =0;
 int x =0;
 int y = 0;



// while(temp < r){

//     cout<<"x , y : ";
//     cin>>x>>y;


//     obj->matrix[temp][0] = x;
//     obj->matrix[temp][1] = y;


// temp +=1;


// }
//x
obj->matrix[0][0] = 100;
obj->matrix[1][0] = 200;
obj->matrix[2][0] = 200;
obj->matrix[3][0] = 100;


//y
obj->matrix[0][1] =0;
obj->matrix[1][1] =0;
obj->matrix[2][1] =100;
obj->matrix[3][1] =100;
cout<<endl;


int tx = 0 ;
int ty = 0;
cout<<"translation factor in x : ";
cin>>tx;
```

```
cout<<"translation factor in y : ";
cin>>ty;



my_matrix *translation_matrix = new my_matrix(3,0);


create_scaling_matrix(tx,ty,translation_matrix);


cout<<"translation matrix::"<<endl;
translation_matrix->print_obj();


cout<<"\nyour input :"<<endl;
obj->print_obj();



my_matrix *result =  new my_matrix(r,0);  // 4  x 3     3 x 3 -->  4 x 3


matrix_multiplication(obj,translation_matrix,result);


cout<<"==============================RESULT============
======================\n";
result->print_obj();



   int gd = DETECT, gm;
   char pathtodriver[] = "";
   initgraph(&gd, &gm, pathtodriver);


   draw_object(obj,RED);
   draw_object(result,BLUE);







getch();
closegraph();



return 0;
}



//============================
```