International Journal of Mathematics and Computer Research

ISSN: 2320-7167

Volume 12 Issue 06 June 2024, Page no. – 4298-4310

Index Copernicus ICV: 57.55, Impact Factor: 8.316

DOI: 10.47191/ijmcr/v12i6.07

Applications of Matrices in Computer Graphics

Muhammad Hanif

Department of Applied Mathematics, Noakhali Science and Technology University, Noakhali 3814, Noakhali, Bangladesh

ARTICLE INFO

Published Online:
15 June 2024

Corresponding Author:
Muhammad Hanif

ABSTRACT

In this paper, the usage of matrix in computer graphics is shown. A brief overview of geometric transformations in computer graphics is given. The "Matrix - Computer Graphics" application software is created for the representation and easier understanding of relations between geometric transformations and matrix calculus. We introduce and explore the basic matrix operation such as translation, rotation, scaling, reflection, shearing in 2D and 3D and at end we explain about the parallel and perspective view. These concepts commonly appear in video game graphics. The work has also shown some programming in using C/C++ code.

KEYWORDS: 2D and 3D matrix transformations, Computer graphics, Programming C/C+, application software.

Mathematics Subject Classification Number 2020: 05C85, 05C90, 15A09, 15A10, 15A12

I. INTRODUCTION

In mathematics, a matrix (plural: matrices) is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. It is necessary to enclose the elements of a matrix in parentheses or brackets. Each value in a matrix is called an element. Provided that they have the same size, two matrices can be added or subtracted element by element. The rule of matrix multiplication, however, is that two matrices can be multiplied only when the number of columns in the first equals the number of rows in the second. A major application of matrices is to represent linear transformations such as rotation, scaling, reflection, shear etc. The term computer graphics includes almost everything on computers that is not text or sound. Today almost every computer can do some graphics, and people have even come to expect to control their computer through icons and pictures rather than just by typing. Here in our lab at the Program of Computer Graphics, we think of computer graphics as drawing pictures on computers, also called rendering. The pictures can be photographs, drawings, movies, or simulations - pictures of things, which do not yet exist and maybecould never exist. Or they may be pictures from places we cannot see directly, such as medical images from inside our body. We spend much of our time improving the way computer pictures can simulate real world scenes.

We want images on computers to not just look more realistic, but also to be more realistic in their colors, the way objects and rooms are lighted, and the way different materials appear. We call this work “realistic image synthesis”. Computer graphics is a computing field that involves the creating, storing and processing of image content via computer. It has a wide application in science, engineering, art and especially in the field of entertainment, movies and video games. [2, 3]. It is important to note that computer graphics applications are used for various educational purposes. For example, computer graphics applications can, in the education process, help people with learning difficulties [10], assist students in understanding computer science algorithms [4], as well as increasing our' interest in programming and computer graphics [1]. The usefulness of a matrix in computer graphics is its ability to convert geometric data into different coordinate systems. A matrix is composed of elements arranged in rows and columns. In simple terms, the elements of a matrix are coefficients that represents the scale or rotation a vector will undergo during a transformation. The use of matrices in computer graphics is widespread. Many industries like architecture, cartoon, automotive that were formerly done by hand drawing now are done routinely with the aid of computer graphics. Video gaming industry, maybe the earliest industry to rely heavily on computer graphics, is now representing rendered polygon in 3- Dimensions. In video gaming industry, matrices are major mathematic tools to

construct and manipulate a realistic animation of a polygonal figure. Examples of matrix operations include translations,

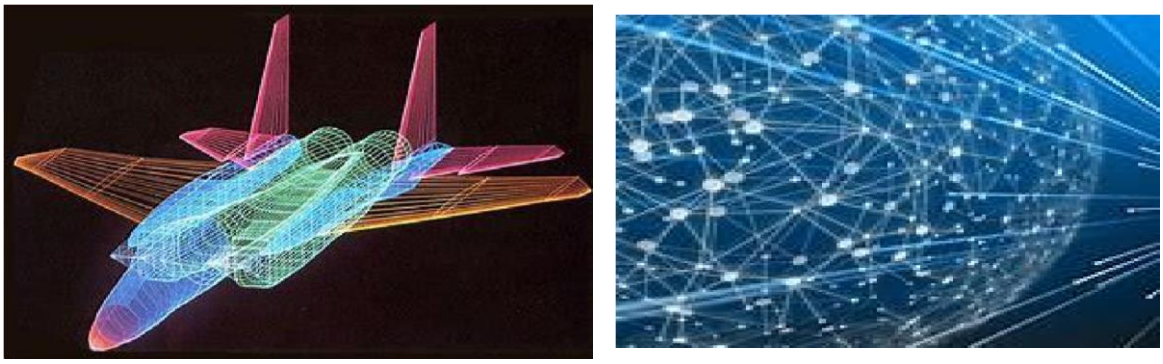


Figure 1. Images on computer graphics based on matrix theory

rotations, and scaling.

In this paper, we apply the matrix based linear transformations in computer graphics and understand about modeling, and interactive control of 2D or 3D computer graphics and create the various formats for storing and working with image in computer graphics.

II. METHODOLOGY

We will use the linear transformation of matrix theory to our work. The following linear equations can be defined:

Translation: A translation basically means adding a vector to a point, making a point transforms to a new point. Translation is a geometric transformation that moves the current point $T = \begin{bmatrix} X \\ Y \end{bmatrix}$ to the point $T' = \begin{bmatrix} X' \\ Y' \end{bmatrix}$ for a given vector $\vec{t} = t_x\vec{i} + t_y\vec{j} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$. Translation is shown in Figure 2

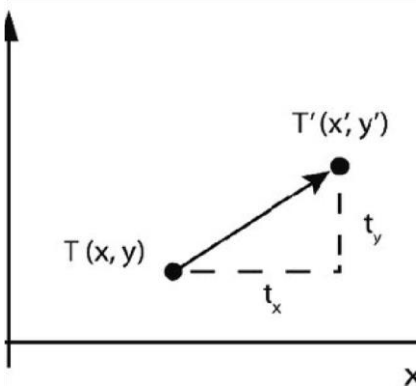


Figure 2: Translation

Therefore, point T' can be written as:

$T'=TT+tt$

and in matrix notation as:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}.$$

Rotation: Let (0, 0) and θ be a point and an angle given respectively. Rotation around the point (0, 0) by the angle θ is a transformation that maps every point T in the plane to the point T' in such a manner that the following properties are true:

$|OT| = |OT'|$ and $\angle TO'T = \theta$

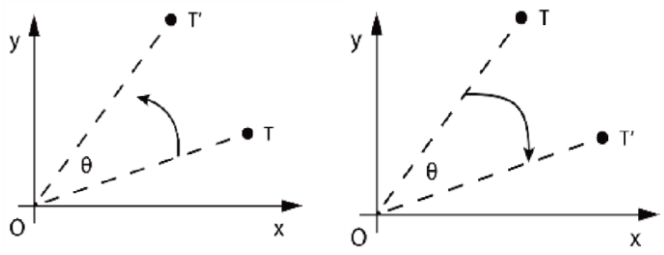


Figure 3: Rotation

By convention, rotation by the positive angle θ goes counter-clockwise (Fig. 3a), while rotation by the negative angle θ goes clockwise (Fig. 3b). Rotation of the point (x,y) located at the angle ϕ from the horizontal x coordinate with a distance r from the origin of the coordinate system leads to a new point $T'(x',y')$. By using standard trigonometry, coordinates of the point (x,) can be expressed as:

$$x = r\cos\phi$$
$$y = r\sin\phi$$

Similarly, coordinates of the point $T'(x',y')$ can be expressed as:

$$x' = r\cos(\phi + \theta)$$
$$y' = r\sin(\phi + \theta)$$

By using the above equations, as well as addition formulas, the following formulas can be obtained:

$$x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta = x\cos\theta - y\sin\theta$$
$$y' = r\cos\phi\sin\theta + r\sin\phi\cos\theta = x\sin\theta + y\cos\theta$$

which can be in a matrix form written as:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling: Scaling is the process of changing the size and proportion of the image. It is the process of expanding or compressing the dimension of an object. There are two factors used in scaling transformation i.e. S_X and S_Y Scaling operation can achieved by multiplying each vertex coordinate $T(x,y)$ of the polygon by scaling factor. and to produce the transformed coordinates as $T'(x'$, $y')$.

$$x' = Sx \cdot x$$
$$y' = Sy \cdot y$$

The scaling factors S_X , S_Y scales the object in X and Y direction respectively. In the matrix form, above equations can be written as:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

The scaling process of the of the square is shown in Fig. 4 .

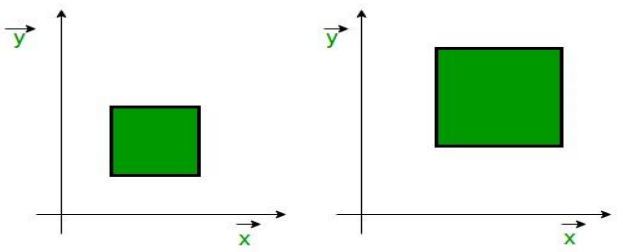


Figure 4. Proportional scaling of the square

Reflection: Reflection is a transformation which generates the mirror image of an object. The mirror image for two dimensional reflections is generated by rotating the object 180 degree on the x-axis of reflection. It is a special case of a scaling where the size of the object does not change. For the scaling factors $Sx = -1$ and $Sy = 1$, reflecting is about the y axis. If the scaling factors are $Sx = 1$ and $Sy = -1$, there is a reflection about the x axis. For the scaling factors $Sx = Sy = -1$, reflection is about the x and y axes. In Fig. 5, reflections about the x and y axes of Super Mario, the main character in the Super Mario video games as an original object, are shown.

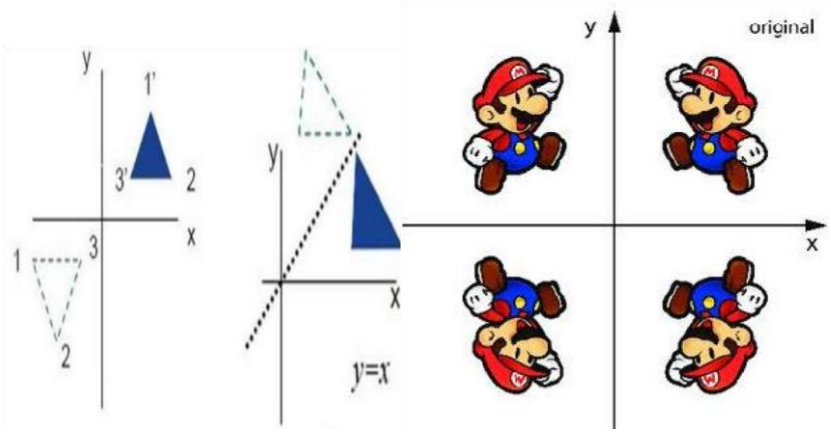


Figure 5. Reflection

Shear: A transformation that slants the shape of an object is called the shear transformation. Also skewing the object with respect to the x or y axis is called shear transformation. Skewing by the angle α with respect to the x axis and by the angle β with respect to the y axis is illustrated in Fig. 6.

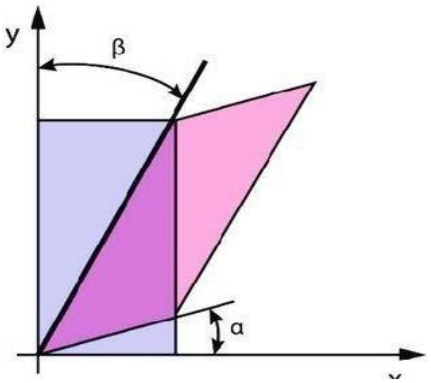


Figure 6: shear Coordinates of the point $T'(x', y)$ can be defined as

$x' = x + y \cdot \tan\beta$ $y' = y + x \cdot \tan\alpha$.

In the matrix form, point TT' can be represented as:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} 1 & tan\beta \\ tan\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

III. IMPLEMENTATION

Since the dawn of computers, Hollywood has greatly demonstrated a Hacker or a Programmer as someone sitting on a computer typing random keys on computer which ultimately compiles to a Falling matrix like simulation. Here, we will try to implement a similar falling matrix simulation on the console using C++.

Figure 7: Falling matrix

The idea here is to print random characters over a defined width, where the two successive characters may or may not have certain amount of gap defined randomly. A certain amount of delay between printing successive lines has to be implemented in order to have a 'falling effect'.

Cubic Bezier Curve Implementation

A Bezier curve is a mathematically defined curve used in two-dimensional graphic applications like adobe Illustrator, Ink scape etc. The curve is defined by four points: **the initial position** and **the terminating position** i.e **P0** and **P3** respectively (which are called "anchors") and **two separate middle points** i.e **P1** and **P2**(which are called "handles") in our example. Bezier curves are frequently used in computer graphics, animation, modeling etc.

Bezier curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as –

$$P(u) = \sum_{i=0}^{n} P_i \, B_i^n(u)$$

P_i

$$B_i^n(u) = \binom{n}{i}(1-u)^{n-i}u^i$$

Where is the set of points and $B_i^n(u)$ represents the Bernstein polynomials. i.e.

Blending Function which are given by –

Where n is the polynomial order, i is the index, and u/t is the variable which has from 0 to 1.

Let us define our cubic Bezier curve mathematically.

So a Bezier curve id defined by a set of control points p_0 to p_n where n is called its order (n = 1 for linear, n = 2 for quadratic, etc.). The first and last control points are always the endpoints of the curve; however, the intermediate control points (if any) generally do not lie on the curve.

For cubic Bezier curve order (n) of polynomial is 3, index (i) vary from i = 0 to i = n
i.e. 3 and u will vary from $0 \leq u \leq 1$. Cubic Bezier Curve function is defined as:

$$P(u) = p_0 B_0^3(u) + p_1 B_1^3(u) + p_2 B_2^3(u) + p_3 B_{03}^3(u)$$

Cubic Bezier Curve blending function are defined as:

$$B_0^3(u) = \binom{3}{0}(1-u)^{3-0}u^0 \equiv 1(1-u)^3u^0$$
$$B_1^3(u) = \binom{3}{1}(1-u)^{3-1}u^1 \equiv 3(1-u)^2u^1$$
$$B_2^3(u) = \binom{3}{2}(1-u)^{3-2}u^2 \equiv 3(1-u)^1u^2$$
$$B_3^3(u) = \binom{3}{3}(1-u)^{3-3}u^3 \equiv 1(1-u)^0u^3$$

so,

$$P(u) = (1-u)^3p_0 + 3(1-u)^2u^1p_1 + 3(1-u)^1u^2p_2 + u^3p_3$$

And

$$P(u) = \{x(u), y(u)\}$$
$$x(u) = (1-u)^3x_0 + 3(1-u)^2u^1x_1 + 3(1-u)^1u^2x_2 + u^3x_3$$
$$y(u) = (1-u)^3y_0 + 3(1-u)^2u^1y_1 + 3(1-u)^1u^2y_2 + u^3y_3$$

So we will calculate curve x and y pixel by incrementing value of u by **0.0001**.

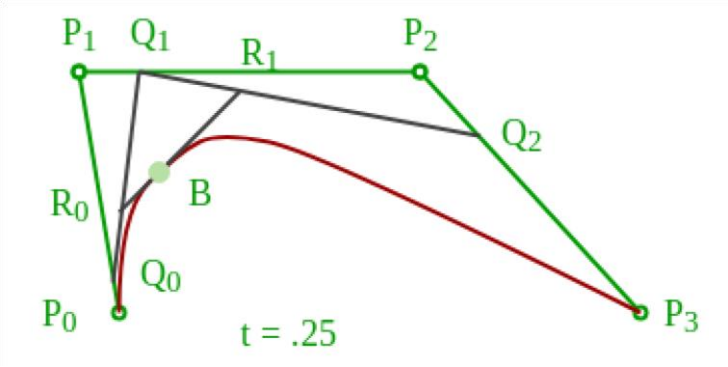


Figure 8. Construction of a cubic Bezier curve

Implementation test:

We are given with four control points $B_0[1,0]$, $B_1[2,2]$, $B_2[6,3]$, $B_3[8,2]$, so determine the five points that lie on the curve also draw the curve on the graph.

Solution:

Given curve has four control points hence it is a cubic Bezier curve, So, the parametric equation of cubic Bezier curve is

$$P(t) = [1\ 0] * (1-t)^3 + 3 * [2\ 2] * (1-t)^2 * t + 3 * [6\ 2] * (1-t)^1 * t^2 + [8\ 2] * t^3$$

Let's assume five different values of t are **{0, 0.2, 0.5, 0.7, 1}.**

So, for t = 0 the coordinate will be,

$$P(0) = [1\ 0] * (1-0)^3 + 3 * [2\ 2] * (1-0)^2 * 0 + 3 * [6\ 2] * (1-0)^1 * 0^2 + [8\ 2] * 0^3$$

$$P(0) = [1\ 0]$$

So, for t=0.2 the coordinate will be,

$$P(0.2) = [1\ 0] * (1-0.2)^3 + 3 * [2\ 2] * (1-0.2)^2 * 0.2 + 3 * [6\ 3] * (1-0.2)^1 * 0.2^2 + [8\ 2] * 0.2^3$$

$$= [1\ 0] * .512 + [2\ 2] * 0.384 + [6\ 2] * 0.096 + [8\ 2] * 0.2^3$$

$$= [0.512\ 0] + [0.768\ 0.768] + [0.576\ 0.288] + [0.064\ 0.016]$$

$$P(0.2) = [1.92\ 1.072]$$

So, for t=0.5 the coordinate will be,

$$P(0.5) = [1\ 0] * (1-0.5)^3 + 3 * [2\ 2] * (1-0.5)^2 * 0.5 + 3 * [6\ 3] * (1-0.5)^1 * 0.5^2 + [8\ 2] * 0.5^3$$

$$P(0.5) = [4.125\ 2.125]$$

So, for t=0.7 the coordinate will be,

$$P(0.7) = [1\ 0] * (1-0.7)^3 + 3 * [2\ 2] * (1-0.7)^2 * 0.7 + 3 * [6\ 3] * (1-0.7)^1 * 0.7^2 + [8\ 2] * 0.7^3$$

$$P(0.7) = [5.79\ 2.387]$$

So, for t=1.0 the coordinate will be,

$$P(1.0) = [1\ 0] * (1-1)^3 + 3 * [2\ 2] * (1-1)^2 * 1 + 3 * [6\ 2] * (1-1)^1 * 1^2 + [8\ 2] * 1^3$$

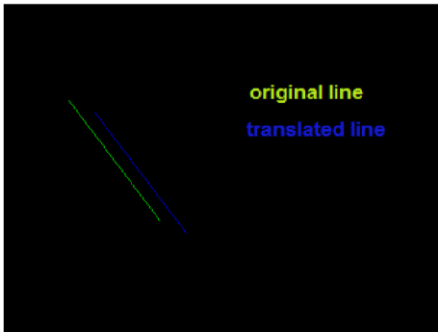
$$P(1.0) = [8\ 2]$$

So, for t = 0 the coordinate will be,

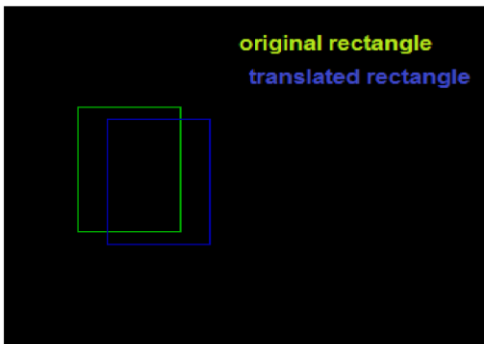
So, for t=0.7 the coordinate will be,

IV RESULT AND DISCUSSION

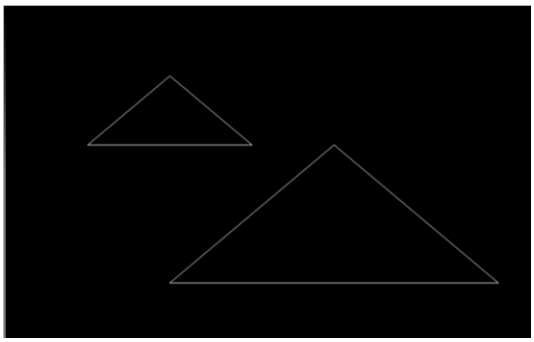
We have tried to get closer to who are dealing with computer science, especially computer graphics, the mathematical background of computer graphics in which the term matrix is in the center. Although computer graphics is nowadays usually 3D, in this paper we have looked at 2D transformations, 3D transformation and some program in order to show. In simpler way the relationship between the matrix calculus and geometric transformations used in computer graphics. For this purpose, the "Matrix in Computer Graphics" application gives a contribution in the visualization and easier learning of relationships between the matrices and their applications on display and positioning, as well as on transforming different objects on a computer. We have found the graphs of our results are as follows:



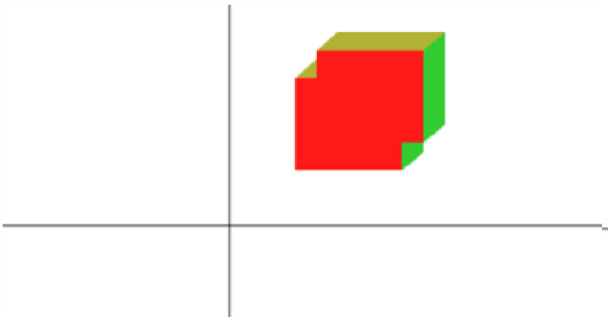
Line translation



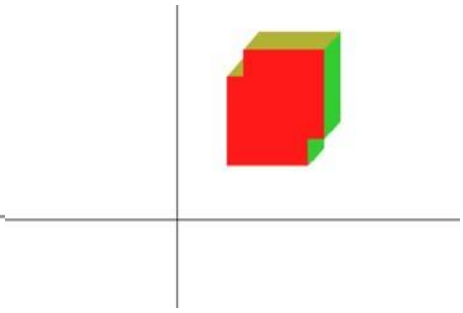
Rectangle translation



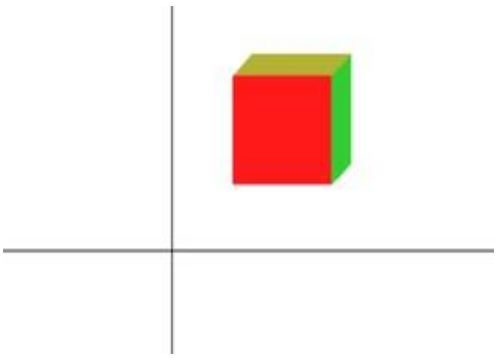
Scaling of Objects



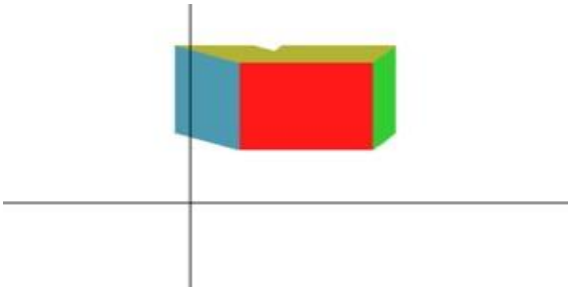
Scaling of Objects



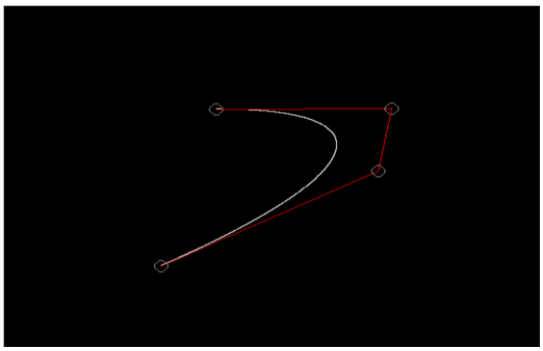
3D Translation



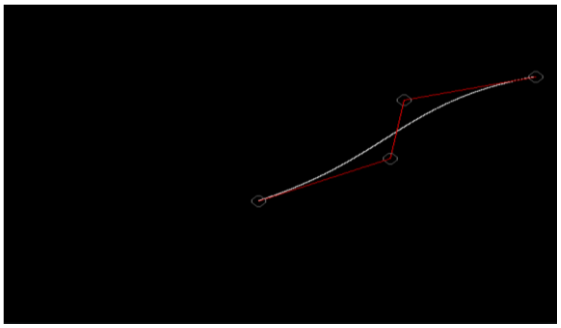
3D Scaling



3D Rotation



Cubic Bezier cure



Cubic Bezier cure

V. FUTURE WORK

We have the chance to talk about computer graphics and math that are often overlooked as unrelated. It is hoped that this activity on matrix using in computer graphics not only offers teachers with greater opportunities to either introduce or consolidate certain mathematical concepts and algorithms, but also to convince their students that mathematics plays an important role in various walks of life and hence is a useful and meaningful field of study. There are huge opportunities of matrices to implement to computer graphics and finding more possibly use results in future.

ACKNOWLEDGEMENT

The Research Cell of Noakhali Science and Technology University, Noakhali 3814, Bangladesh for providing the financial support to work with this article.

REFERENCES

1. Bruno, J., Jan, V., & Slaby, A. (2009).Computer Graphics in Computer Science Education. *Problems of Education in the 21st Century*, *11*, 60-68.
2. Hearn, D. D., Baker, M. P., & Carithers, W. (2014).*Computer Graphics with Open GL*, 4th Edition. Pearson Education Limited.
3. Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., & Foley, J. D. (2014). Computer Graphics: Principles and Practice, 3rd Edition. Addison-Wesley.
4. Naps, T. L. (2005). JHAVÉ: Supporting Algorithm Visualization. IEEE Computer Graphics and Applications, 25(5), 49-55. https://doi.org/10.1109/MCG.2005.110
5. Petricek, T. & Skeet, J. (2010). Real-World Functional Programming: With Examples in F# and C#, 1st Edition. Manning Publications.
6. Salomon, D. (2011).The computer graphics manual, Volume 2. Springer-Verlag London Limited. https://doi.org/10.1007/978-0-85729-886-7
7. Sharp, J. (2013).Microsoft Visual C# 2013 Step by Step (Step by Step Developer), 1st Edition. Microsoft Press.
8. Skala, V. (2006). Length, Area and Volume Computation in Homogeneous Coordinates. *International Journal of Image and Graphics*, *06*(04), 625-639. https://doi.org/10.1142/S0219467806002422
9. Srivastav, M. K. (2016). Transformation of an Object in Computer Graphics: A Case Study of Mathematical Matrix Theory. Elixir Comp. Engg. 100, 43396-43399.
10. Vera, L., Campos, R., Herrera, G., & Romero, C. (2007). Computer graphics applications in the education process of people with learning difficulties. Computer& graphics, 31,649658. https://doi.org/10.1016/j.cag.2007.03.003

Appendix

Implementation of falling matrix

```
// C++ program for implementation of falling matrix.
#include<iostream>
#include<string>
#include<thread>
#include<cstdlib>
#include<ctime>
#include<chrono>

// Width of the matrix line
const int width = 70;
```

```
// Defines the number of flips in Boolean Array 'switches' const int flipsPerLine =5;

// Delay between two successive line print const int sleepTime = 100;

using namespace std;
   int main()
   {
      int i=0, x=0;

      // srand initialized with time function
      // to get distinct rand values at runtime
      srand(time(NULL));

      // Used to decide whether to print
      // the character in that particular iteration
      bool switches[width] = {0};

      // Set of characters to print from
      const string ch = "1234567890qwertyuiopasdfghjkl"
                        "zxcvbnm,./';[]!@#$%^&*()-=_+";
      const int l = ch.size();

      // Green font over black console, duh!
      system("Color 0A");

      // Indefinite Loop
      while (true)
      {
         // Loop over the width
         // Increment by 2 gives better effect
         for (i=0;i<width;i+=2)
         {
            // Print character if switches[i] is 1
            // Else print a blank character
            if (switches[i])
               cout << ch[rand() % l] << " ";
            else
               cout<<"  ";
         }
      // Flip the defined amount of Boolean values
      // after each line
      for (i=0; i!=flipsPerLine; ++i)
      {
         x = rand() % width;
         switches[x] = !switches[x];
      }

       // New Line
      cout << endl;

      // Using sleep_for function to delay,
      // chrono milliseconds function to convert to milliseconds
      this_thread::sleep_for(chrono::milliseconds(sleepTime));
   }
   return 0;
}
```

Implement Cubic Bezier Curve

```
// C program to implement
// Cubic Bezier Curve

/* install SDL library for running thing code*/
/* install by using this commamnd line : sudo apt-get install libsdl2-dev */
/* run this code using command : gcc fileName.c -lSDL2 -lm*/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<SDL2/SDL.h>

SDL_Window* window = NULL;
SDL_Renderer* renderer = NULL;
int mousePosX , mousePosY ;
int xnew , ynew ;
/*Function to draw all other 7 pixels present at symmetric position*/
void drawCircle(int xc, int yc, int x, int y)
{
        SDL_RenderDrawPoint(renderer,xc+x,yc+y) ;
        SDL_RenderDrawPoint(renderer,xc-x,yc+y);
        SDL_RenderDrawPoint(renderer,xc+x,yc-y);
        SDL_RenderDrawPoint(renderer,xc-x,yc-y);
        SDL_RenderDrawPoint(renderer,xc+y,yc+x);
        SDL_RenderDrawPoint(renderer,xc-y,yc+x);
        SDL_RenderDrawPoint(renderer,xc+y,yc-x);
        SDL_RenderDrawPoint(renderer,xc-y,yc-x);
}
/*Function for circle-generation using Bresenham's algorithm */
void circleBres(int xc, int yc, int r)
{
        int x = 0, y = r;
        int d = 3 - 2 * r;
        while (y >= x)
        {
                /*for each pixel we will draw all eight pixels */
                drawCircle(xc, yc, x, y);
                x++;

                  /*check for decision parameter and correspondingly update d, x, y*/
            if (d > 0)
                {
                        y--;
                        d = d + 4 * (x - y) + 10;
                }
                else
                        d = d + 4 * x + 6;
                drawCircle(xc, yc, x, y);
        }
}

/* Function that take input as Control Point x_coordinates and Control Point y_coordinates and draw bezier
curve */
void bezierCurve(int x[] , int y[])
```

```
{
        double xu = 0.0 , yu = 0.0 , u = 0.0 ;
        int i = 0 ;
        for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
        {
                xu = pow(1-u,3)*x[0]+3*u*pow(1-u,2)*x[1]+3*pow(u,2)*(1-u)*x[2]
                        +pow(u,3)*x[3];
                yu = pow(1-u,3)*y[0]+3*u*pow(1-u,2)*y[1]+3*pow(u,2)*(1-u)*y[2]
                        +pow(u,3)*y[3];
                SDL_RenderDrawPoint(renderer , (int)xu , (int)yu) ;
        }
}
int main(int argc, char* argv[])
{
        /*initialize sdl*/
        if (SDL_Init(SDL_INIT_EVERYTHING) == 0)
        {
                /*This function is used to create a window and default renderer.
            int SDL_CreateWindowAndRenderer(int width,int height,Uint32 window_flags
                ,SDL_Window** window ,SDL_Renderer** renderer) return 0 on success and
                                        -1 on error*/

            if(SDL_CreateWindowAndRenderer(640, 480, 0, &window, &renderer) == 0)
                {
                        SDL_bool done = SDL_FALSE;

                        int i = 0 ;
                        int x[4] , y[4] , flagDrawn = 0 ;

                        while (!done)
                        {
                                SDL_Event event;


                    /*set background color to black*/
                SDL_SetRenderDrawColor(renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);
                        SDL_RenderClear(renderer);

                    /*set draw color to white*/
      SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
 /* We are drawing cubic bezier curve which has four control points */
                    if(i==4)
                    {
                            bezierCurve(x , y) ;
                            flagDrawn = 1 ;
                    }

                    /*grey color circle to encircle control Point P0*/
      SDL_SetRenderDrawColor(renderer, 128, 128, 128, SDL_ALPHA_OPAQUE);
                    circleBres(x[0] , y[0] , 8) ;

                    /*Red Line between control Point P0 & P1*/
       SDL_SetRenderDrawColor(renderer, 255, 0, 0, SDL_ALPHA_OPAQUE);
                    SDL_RenderDrawLine(renderer , x[0] , y[0] , x[1] , y[1]) ;

            /*grey color circle to encircle control Point P1*/
      SDL_SetRenderDrawColor(renderer, 128, 128, 128, SDL_ALPHA_OPAQUE);
```

```
                circleBres(x[1] , y[1] , 8) ;

                /*Red Line between control Point P1 & P2*/
SDL_SetRenderDrawColor(renderer, 255, 0, 0, SDL_ALPHA_OPAQUE);
                SDL_RenderDrawLine(renderer , x[1] , y[1] , x[2] , y[2]) ;

                /*grey color circle to encircle control Point P2*/
SDL_SetRenderDrawColor(renderer, 128, 128, 128, SDL_ALPHA_OPAQUE);
                circleBres(x[2] , y[2] , 8) ;

                /*Red Line between control Point P2 & P3*/
SDL_SetRenderDrawColor(renderer, 255, 0, 0, SDL_ALPHA_OPAQUE);
                SDL_RenderDrawLine(renderer , x[2] , y[2] , x[3] , y[3]) ;
        /*grey color circle to encircle control Point P3*/
SDL_SetRenderDrawColor(renderer, 128, 128, 128, SDL_ALPHA_OPAQUE);
                circleBres(x[3] , y[3] , 8) ;

                /*We are Polling SDL events*/
                if (SDL_PollEvent(&event))
                {
                /* if window cross button clicked then quit from window */
                        if (event.type == SDL_QUIT)
                        {
                                done = SDL_TRUE;
                        }

                                                /*Mouse Button is Down */
                                                if(event.type == SDL_MOUSEBUTTONDOWN)
                                                {
                  /*If left mouse button down then store that point as control point*/
                                                if(event.button.button == SDL_BUTTON_LEFT)
                                                        {
                                /*store only four points because of cubic bezier curve*/
                                                                if(i < 4)
                                 printf("Control Point(P%d):(%d,%d)\n",i,mousePosX,mousePosY) ;

                   /*Storing Mouse x and y positions in our x and y coordinate array */
                                                                        x[i] = mousePosX ;
                                                                        y[i] = mousePosY ;
                                                                        i++ ;
                                                                }
                                                        }
                                                }
                                                /*Mouse is in motion*/
                                                if(event.type == SDL_MOUSEMOTION)
                                                {
                                                  /*get x and y positions from motion of mouse*/
                                                        xnew = event.motion.x ;
                                                        ynew = event.motion.y ;
                                                      int j ;
                                 /* change coordinates of control point after bezier curve has been drawn */
                                                        if(flagDrawn == 1)
                                                        {
                                                                for(j = 0 ; j < i ; j++)
                                                                {
/*Check mouse position if in b/w circle then change position of that control point to mouse new position
which are coming from mouse motion*/
```

```
if((float)sqrt(abs(xnew-x[j]) * abs(xnew-x[j]) + abs(ynew-y[j]) * abs(ynew-y[j])) < 8.0)
                                                                {
                                        /*change coordinate of jth control point*/
                                                x[j] = xnew ;
                                                y[j] = ynew ;
                                printf("Changed Control Point(P%d):(%d,%d)\n",j,xnew,ynew) ;
                                                                }
                                                        }
                                                }
                                /*updating mouse positions to positions coming from motion*/
                                                mousePosX = xnew ;
                                                mousePosY = ynew ;
                                        }
                                }
                                /*show the window*/
                                SDL_RenderPresen t(renderer);
                        }
                }

                /*Destroy the renderer and window*/
                if (renderer)
                {
                    SDL_DestroyRenderer(renderer);
                }
                if (window)
                {
               SDL_DestroyWindow(window);
                }
        }
        /*clean up SDL*/
        SDL_Quit();
        return 0;
}
```