Instituto Politécnico Nacional
Escuela Superior de Cómputo
Computer Graphics
1 Introduction to Computer Graphics
Malagón Baeza Alan Adrian
Rosas Trigueros Jorge Luis
27/03/2021
09/04/2021

Theoretical framework

Drawing graphics primitives in C# with Windows Forms is a common task for creating visual applications. Graphics primitives are the basic building blocks of any visual representation, and the ability to manipulate them through transformations and animations is essential for creating dynamic and engaging user interfaces. This theoretical framework will explore the concepts of graphics primitives, transformation matrices, and animation, and how they relate to C# and Windows Forms.

In computer graphics, a primitive is a basic element that can be used to create more complex images. Examples of graphics primitives include points, lines, and polygons. In C# with Windows Forms, graphics primitives are represented by objects of the System.Drawing namespace, such as the Point, Line, and Rectangle classes. These primitives can be drawn onto a graphical surface using the Graphics object, which is provided by the framework.

Transformation matrices are mathematical objects that represent a transformation in a 2D or 3D space. In computer graphics, these matrices are used to perform operations such as translation, rotation, scaling, and skewing on graphics primitives. In C# with Windows Forms, transformation matrices are represented by the Matrix class, which provides methods for performing these operations on graphics primitives.

In 2D graphics, transformation matrices can be used to manipulate the position, size, and orientation of graphics primitives. Translation matrices are used to move primitives in the x and y directions, rotation matrices are used to rotate primitives around a fixed point, and scaling matrices are used to increase or decrease the size of primitives. Skewing matrices can also be used to deform primitives along one or more axes.

Animation is the process of creating the illusion of motion by rapidly displaying a sequence of static images. In C# with Windows Forms, animations can be created by updating the position, size, and orientation of graphics primitives over time. This can be achieved by using a timer to update the transformation matrix of the primitive at regular intervals, which will cause it to move, rotate, or scale in the desired manner. By combining multiple transformations, complex animations can be created, such as a rotating and scaling triangle that moves in a specific direction.

Material and equipment

We will use C# (Visual Studio 2019), and XAML. A personal computer with AMD Ryzen 5 5600H and 16 GB of RAM.

Practice development

Catalog with 9 graphics primitives

The next code is a partial class for a Windows Forms application, specifically for a form called "Form1". The code is auto-generated by the Visual Studio designer when creating a new form.

The code defines a class with a private member variable called **components** which is an instance of the **System.ComponentModel.IContainer** interface. The **components** object is used to manage and dispose of the resources that the form uses, such as controls and images.

The class also defines a method called **Dispose** which is used to dispose of the **components** object when the form is closed or disposed.

Finally, the class contains a region of code generated by the designer, which includes the **InitializeComponent** method. This method sets up the form's visual appearance and properties, including its size and title. It also assigns event handlers for the **Load** and **Paint** events of the form, which will be executed when the form is loaded and painted respectively.
Regenerate response

```
namespace WindowsFormsLabSession_1
{
    partial class Form1
    {
        /// <summary>
        /// Variable del diseñador necesaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se estén usando.
        /// </summary>
        /// <param name="disposing">true si los recursos
administrados se deben desechar; false en caso contrario.</param>
        protected override void Dispose(bool disposing)
```

```
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Código generado por el Diseñador de Windows Forms

        /// <summary>
        /// Método necesario para admitir el Diseñador. No se puede
modificar
        /// el contenido de este método con el editor de código.
        /// </summary>
        private void InitializeComponent()
        {
            this.SuspendLayout();
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F,
20F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(1123, 869);
            this.Name = "Form1";
            this.Text = "Form1";
            this.Load += new System.EventHandler(this.Form1_Load);
            this.Paint += new
System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
            this.ResumeLayout(false);

        }

        #endregion
    }
}
```

The next code is a C# program that uses the .NET Framework's Windows Forms library to draw various shapes on a form. The form is defined in the Form1 class, which inherits from the Form class provided by the library.

The *InitializeComponent()* method is called in the constructor of the Form1 class to initialize the components of the form.

The *Form1_Paint()* method is an event handler that is called whenever the form is painted. This method creates a Graphics object, which is used to draw the shapes. It also creates several Pen objects, each with a different color, that are used to draw the shapes.

The code draws several shapes, including a rectangle, a triangle, a circle, a rhomb, a trapeze, a semicircle, a pie, a line, and a pixel. The coordinates and dimensions of these shapes are calculated based on the size of the form.

Each shape is drawn using the Graphics object and the appropriate Pen object. The *DrawLine()* method is used to draw straight lines, the *DrawRectangle()* method is used to draw rectangles, the *DrawEllipse()* method is used to draw circles, the *DrawPolygon()* method is used to draw the rhomb, the *DrawPath()* method is used to draw the trapeze, the *DrawArc()* method is used to draw the semicircle and pie, and the *DrawLine()* method is used to draw the line.

The last shape drawn is a pixel, which is filled using a Brush object with a DarkOrange color. The *FillRectangle()* method is used to fill the pixel.

Overall, this code demonstrates how to use the Windows Forms library to draw shapes on a form using a Graphics object and various Pen and Brush objects.

```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace WindowsFormsLabSession_1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
```

```
        {
            Graphics g = e.Graphics;
            Pen myDrawingPen = new Pen(Color.Blue);
            Pen myDrawingPen1 = new Pen(Color.Red);
            Pen myDrawingPen2 = new Pen(Color.Firebrick);
            Pen myDrawingPen3 = new Pen(Color.Green);
            Pen myDrawingPen4 = new Pen(Color.DarkOrchid);
            Pen myDrawingPen5 = new Pen(Color.CornflowerBlue);
            Pen myDrawingPen6 = new Pen(Color.Indigo);
            Pen myDrawingPen7 = new Pen(Color.DarkGoldenrod);
            Pen myDrawingPen8 = new Pen(Color.DarkOrange);

            // Form Division
            // |  |  |  |  |
            // -------------
            // |  |  |  |  |
            g.DrawLine(myDrawingPen, 0, Size.Height / 2,
Size.Width, Size.Height / 2);
            g.DrawLine(myDrawingPen, Size.Width / 4, 0, Size.Width
/ 4, Size.Height);
            g.DrawLine(myDrawingPen, 2 * Size.Width / 4, 0, 2 *
Size.Width / 4, Size.Height);
            g.DrawLine(myDrawingPen, 3 * Size.Width / 4, 0, 3 *
Size.Width / 4, Size.Height);

            // Rectangle

            g.DrawRectangle(myDrawingPen1, 10, 10, (Size.Width / 4)
- 20, (Size.Height / 2) - 20);

            // Triangle

            g.DrawLine(myDrawingPen2, (Size.Width / 4) + 10,
(Size.Height / 2) - 10, 2 * (Size.Width / 4) - 10, (Size.Height /
2) - 10);
            g.DrawLine(myDrawingPen2, (Size.Width / 4) + 10,
(Size.Height / 2) - 10, ((Size.Width / 4) / 2) + (Size.Width / 4),
10);
            g.DrawLine(myDrawingPen2, 2 * (Size.Width / 4) - 10,
(Size.Height / 2) - 10, ((Size.Width / 4) / 2) + (Size.Width / 4),
10);
```

```
            // Circle

            g.DrawEllipse(myDrawingPen3, (Size.Width / 2) + 10, 10,
(Size.Width / 4) - 20, (Size.Height / 2) - 20);

            // Rhomb

            Point[] points = new Point[4];
            points[0] = new Point((3 * (Size.Width / 4)) +
Size.Width / 8, 10);
            points[1] = new Point(Size.Width - 20, (Size.Height /
4));
            points[2] = new Point((3 * (Size.Width / 4)) +
Size.Width / 8, (Size.Height / 2) - 10);
            points[3] = new Point((Size.Width / 4) * 3 + 10,
(Size.Height / 4));
            g.DrawPolygon(myDrawingPen4, points);


            // Trapeze

            GraphicsPath path = new GraphicsPath();
            path.AddLine(10, (Size.Height / 2) + 10, 10,
Size.Height - 50);
            path.AddLine(10, Size.Height - 50,(Size.Width/4)-10,
Size.Height-100);
            path.AddLine((Size.Width/4)-10, Size.Height-100,
(Size.Width / 4) - 10, (Size.Height/2) + 50);
            path.AddLine((Size.Width / 4) - 10, (Size.Height/2) +
50, 10, (Size.Height / 2) + 10);
            g.DrawPath(myDrawingPen5, path);

            // Semicircle

            g.DrawArc(myDrawingPen6,
(Size.Width/4)+10,(Size.Height/2)+10,(Size.Width/4)-20,(Size.Height
/2)-60,90,180);
            g.DrawLine(myDrawingPen6, (Size.Width / 4)+ (Size.Width
/ 8), (Size.Height / 2) + 10, (Size.Width / 4) + (Size.Width / 8),
Size.Height - 50);

            // Pie
```

```
            g.DrawPie(myDrawingPen7, (Size.Width / 2) + 10,
(Size.Height / 2) + 10, (Size.Width / 4) - 20, (Size.Height / 2) -
60, 0, 270);

            // Line

            g.DrawLine(myDrawingPen8,((Size.Width*3)/4)+10,
(Size.Height / 2) + 10, Size.Width-30,Size.Height-50);

            // Pixel
            Brush brush = new SolidBrush(Color.DarkOrange);
            g.FillRectangle(brush, Size.Width - 30, Size.Height/2 +
10, 1, 1);
        }
    }
}
```

When starting the program, the result obtained is like the one in the following figure.

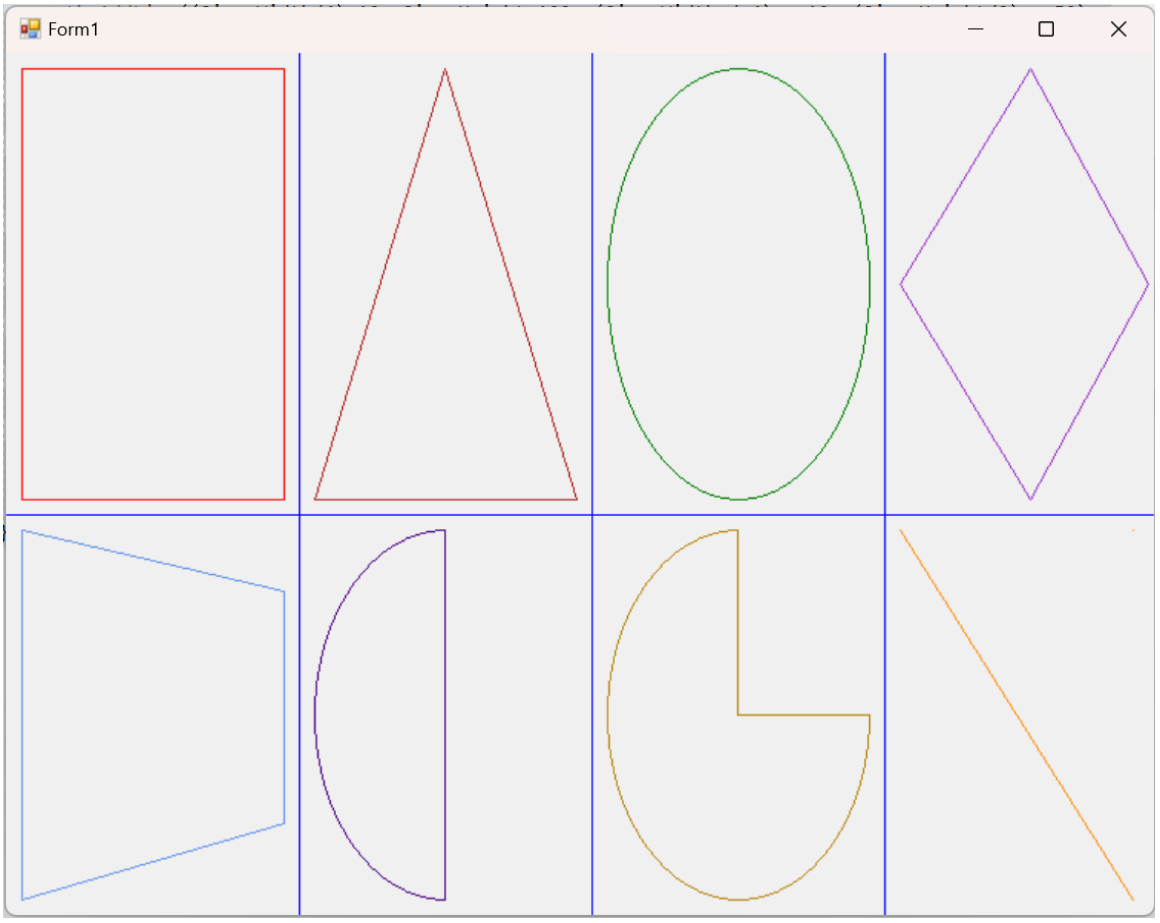

Figure 1. Program that shows a catalog with 9 graphics primitives.

Effect of a transformation matrix in 2D

The next code is XAML code for a WPF application's main window. It defines the user interface of the window. Here is a brief explanation of what the code does:

- The Window tag defines the main window of the application. It has several attributes, including the title, height, and width of the window.

- The Viewbox tag is used to resize the content of the window based on the size of the window.

- The Grid tag defines a grid layout with two columns. The first column is 150 pixels wide, and the second column is 280 pixels wide.

- The first grid in the first column contains a grid with two columns and eight rows. Each row contains a TextBlock and a TextBox control. These controls are used to input the values for a 2x2 matrix and two offset values.

- The two buttons are also defined in this grid. The first button applies the input values to a rectangle in the canvas, and the second button closes the window.
- The second grid is defined in the second column and contains a canvas with a rectangle in it. The rectangle is initially displayed as a black border with a light coral fill and an opacity of 0.5. The rectangle is centered in the canvas and has a width of 50 pixels and a height of 70 pixels.

- The rectangle also has a MatrixTransform applied to it, which allows it to be transformed using matrix operations. The matrix is initially set to the identity matrix, which means that the rectangle is not transformed. The matrix can be updated by the user using the input controls in the first grid.

- The Canvas also contains a TextBlock control, which displays the text "Original shape" above the rectangle.

Overall, this XAML code defines a window with a simple user interface that allows the user to input values for a 2x2 matrix and two offset values, which are then applied to a rectangle in a canvas. The window also allows the user to close the window or reset the input values.

```
<Window x:Class="WpfAppLabSession_1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfAppLabSession_1"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">

    <Viewbox Stretch="Uniform">
        <Grid Width="430" Height="300"
              HorizontalAlignment="Left"
              VerticalAlignment="Top">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150"/>
                <ColumnDefinition Width="280"/>
            </Grid.ColumnDefinitions>
            <Grid Width="140" Height="300" Margin="5,10,5,5">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="60"/>
                    <ColumnDefinition Width="70"/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                </Grid.RowDefinitions>
                <TextBlock HorizontalAlignment="Right"
                           Grid.Column="0" Grid.Row="0"
                           Margin="5,5,10,5">M11</TextBlock>
                <TextBox Name="tbM11" Grid.Column="1"
                         Grid.Row="0"
                         TextAlignment="Center">1</TextBox>
```

```
            <TextBlock HorizontalAlignment="Right"
                       Grid.Column="0" Grid.Row="1"
                       Margin="5,5,10,5">M12</TextBlock>
            <TextBox Name="tbM12" Grid.Column="1"
                     Grid.Row="1"
                     TextAlignment="Center">0</TextBox>
            <TextBlock HorizontalAlignment="Right"
                       Grid.Column="0" Grid.Row="2"
                       Margin="5,5,10,5">M21</TextBlock>
            <TextBox Name="tbM21" Grid.Column="1"
                     Grid.Row="2"
                     TextAlignment="Center">0</TextBox>
            <TextBlock HorizontalAlignment="Right"
                       Grid.Column="0" Grid.Row="3"
                       Margin="5,5,10,5">M22</TextBlock>
            <TextBox Name="tbM22" Grid.Column="1"
                     Grid.Row="3"
                     TextAlignment="Center">1</TextBox>
            <TextBlock HorizontalAlignment="Right"
                       Grid.Column="0" Grid.Row="4"
                       Margin="5,5,10,5">OffsetX</TextBlock>
            <TextBox Name="tbOffsetX" Grid.Column="1"
                     Grid.Row="4"
                     TextAlignment="Center">0</TextBox>
            <TextBlock HorizontalAlignment="Right"
                       Grid.Column="0" Grid.Row="5"
                       Margin="5,5,10,5">OffsetY</TextBlock>
            <TextBox Name="tbOffsetY" Grid.Column="1"
                     Grid.Row="5"
                     TextAlignment="Center">0</TextBox>
            <Button Click="BtnApply_Click"
                    Margin="15,20,15,5" Grid.Row="6"
                    Height="25" Grid.ColumnSpan="2"
                    Grid.Column="0">Apply</Button>
            <Button Click="BtnClose_Click"
                    Margin="15,0,15,5" Grid.Row="7"
                    Height="25" Grid.ColumnSpan="2"
                    Grid.Column="0">Close</Button>
        </Grid>
        <Border Margin="10" Grid.Column="1"
                BorderBrush="Black"
                BorderThickness="1"
```

```
                    HorizontalAlignment="Left">
                <Canvas Name="canvas1" Grid.Column="1"
                        ClipToBounds="True" Width="270"
                        Height="280">
                    <TextBlock Canvas.Top="53"
                               Canvas.Left="90">Original
shape</TextBlock>
                    <Rectangle Canvas.Top="70"
                               Canvas.Left="100"
                               Width="50" Height="70"
                               Stroke="Black"
                               StrokeThickness="2"
                               StrokeDashArray="3,1"/>
                    <Rectangle Name="rect" Canvas.Top="70"
                               Canvas.Left="100"
                               Width="50" Height="70"
                               Fill="LightCoral" Opacity="0.5"
                               Stroke="Black"
                               StrokeThickness="2">
                        <Rectangle.RenderTransform>
                            <MatrixTransform
                                x:Name="matrixTransform"/>
                        </Rectangle.RenderTransform>
                    </Rectangle>
                </Canvas>
            </Border>
        </Grid>
    </Viewbox>
</Window>
```

This code is a C# code for a WPF (Windows Presentation Foundation) application. It includes three namespaces: System, System.Windows, and System.Windows.Media.

The namespace WpfAppLabSession_1 contains a partial class called MainWindow, which is the main window of the application.

The MainWindow class has a constructor that calls the InitializeComponent method, which is used to initialize the components of the window. Additionally, it contains two methods that handle button click events: BtnApply_Click and BtnClose_Click.

The BtnApply_Click method parses the values entered in several text boxes and assigns them to a new Matrix object. The Matrix object is then assigned to a

MatrixTransform object, which is used to transform an element in the window. This method is triggered when the user clicks on the "Apply" button.

The BtnClose_Click method simply closes the application window when the user clicks on the "Close" button.

Overall, this code demonstrates the use of event handling and WPF graphics transformations in a C# application.

```
using System;
using System.Windows;
using System.Windows.Media;

namespace WpfAppLabSession_1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        public void BtnApply_Click(object sender, EventArgs e)

        {

            Matrix m = new Matrix();

            m.M11 = Double.Parse(tbM11.Text);

            m.M12 = Double.Parse(tbM12.Text);

            m.M21 = Double.Parse(tbM21.Text);

            m.M22 = Double.Parse(tbM22.Text);

            m.OffsetX = Double.Parse(tbOffsetX.Text);

            m.OffsetY = Double.Parse(tbOffsetY.Text);

            matrixTransform.Matrix = m;

        }
```

```
        public void BtnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

When starting the program, the result obtained is like the one in the following figure.

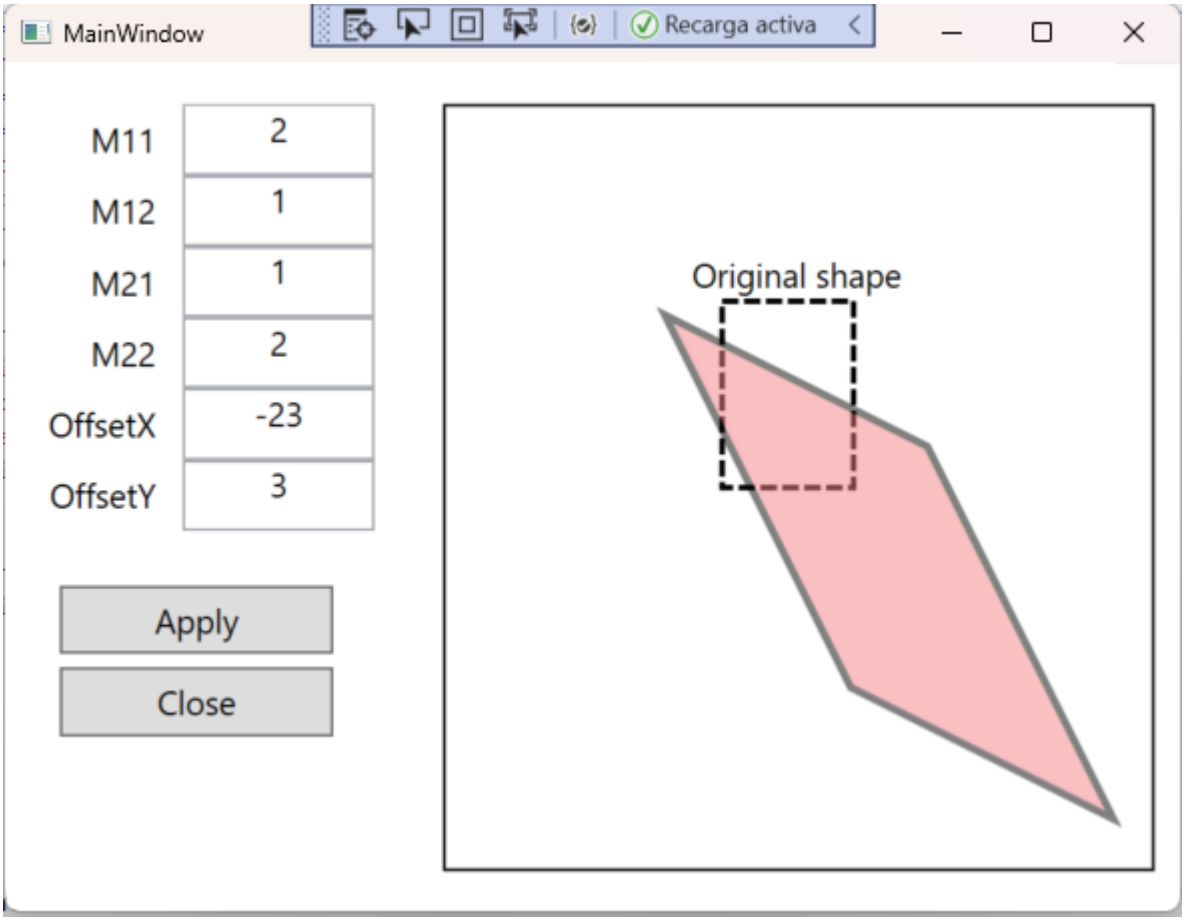

Figure 2. Program to test the effect of a transformation matrix in 2D.

Animation of triangle

The next code defines a partial class **Form1** in the **WindowsFormsCGLS1** namespace. The **partial** keyword means that this class is part of a class definition that spans multiple files.

This class extends the System.Windows.Forms.Form class and defines a form that can be displayed on the screen. The form contains a **Timer** component and two event handlers: **Form1_Paint** and **Form1_KeyPress**.

The **Dispose** method is used to free up any resources used by the form, such as memory allocated for controls, when the form is closed.

The **InitializeComponent** method is a generated method that initializes the form's controls and sets its properties. In this case, it sets the form's size, name, and text, as well as adding event handlers for the **timer1** control, **Paint** and **KeyPress** events.

The **timer1** control is used to periodically execute code in the **timer1_Tick** event handler.

Overall, this code is part of a Windows Forms application that implements a graphical user interface (GUI) with a timer and event handlers for painting and key input.

```
namespace WindowsFormsCGLS1
{
    partial class Form1
    {
        /// <summary>
        /// Variable del diseñador necesaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se estén usando.
        /// </summary>
        /// <param name="disposing">true si los recursos administrados se
deben desechar; false en caso contrario.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Código generado por el Diseñador de Windows Forms

        /// <summary>
        /// Método necesario para admitir el Diseñador. No se puede
modificar
```

```
        /// el contenido de este método con el editor de código.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.timer1 = new
System.Windows.Forms.Timer(this.components);
            this.SuspendLayout();
            //
            // timer1
            //
            this.timer1.Tick += new
System.EventHandler(this.timer1_Tick);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 20F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Name = "Form1";
            this.Text = "Form1";
            this.Paint += new
System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
            this.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.Form1_KeyPress);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.Timer timer1;
    }
}
```

The next code is a C# Windows Forms application that displays a triangle on a canvas and allows the user to interact with it using keyboard controls.

The application starts by defining a Form class named "Form1" and importing some necessary libraries such as System, System.Drawing, System.Drawing.Drawing2D, and System.Windows.Forms.

The class contains several public and private fields, including integer fields for tx, ty, dx, dy, r, and dr, as well as a float field for s. It also has a Matrix object named "myMatrix" and an array of Point objects named "triangle."

The constructor for Form1 initializes several of these fields and starts a timer. The timer's tick event is defined in the timer1_Tick method, which updates the values of tx, ty, and myMatrix using the current values of dx, dy, dr, and s. It also checks if the triangle has gone out of bounds and resets its position if necessary. Finally, it invalidates the canvas to trigger a repaint.

The Form1_Paint event handler method is called whenever the form needs to be redrawn, and it retrieves a Graphics object from the PaintEventArgs argument. It sets the Graphics object's Transform property to the value of myMatrix and fills the triangle using a SolidBrush object.

The Form1_KeyPress event handler method is called whenever the user presses a key on the keyboard. It checks which key was pressed and updates dx, dy, dr, or s accordingly. The keys 's', 'a', 'd', 'w', 'x', 'q', 'z', and 'c' are used to move the triangle in different directions, while the keys 'r' and 'R' are used to rotate it. The keys 'f' and 'F' are used to scale the triangle up and down, respectively.

Overall, the application provides a simple example of how to create an interactive graphical user interface using C# and Windows Forms.

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace WindowsFormsCGLS1
{
    public partial class Form1 : Form
    {
        public int tx, ty, dx, dy, r, dr;
        public float s = 1;


        public Matrix myMatrix;
        public Point[] triangle = {
            new Point(0,0),
            new Point(100,100),
            new Point(0,200)
        };

        private void timer1_Tick(object sender, EventArgs e)
        {
            tx += dx;
            ty += dy;
```

```
            myMatrix.Translate(dx, dy, MatrixOrder.Append);
            myMatrix.Rotate(dr);
            myMatrix.Scale(s, s);
            s = 1;
            if (tx > Size.Width)
            {
                myMatrix.Translate(-tx, 0, MatrixOrder.Append);
                tx = 0;
            }
            if (tx < 0)
            {
                myMatrix.Translate(Size.Width, 0,
MatrixOrder.Append);
                tx = Size.Width;
            }
            if (ty > Size.Height)
            {
                myMatrix.Translate(0, -ty, MatrixOrder.Append);
                ty=0;
            }
            if (ty < 0)
            {
                myMatrix.Translate(0, Size.Height,
MatrixOrder.Append);
                ty = Size.Height;
            }
            this.Invalidate();
        }

        public Form1()
        {
            InitializeComponent();
            this.DoubleBuffered = true; // prevent glitching
            this.KeyPreview = true;
            myMatrix = new Matrix();
            tx = 0;
            ty = 0;
            timer1.Start();
            // x = this.Width;
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
```

```
        {
            Graphics g = e.Graphics;
            Pen myDrawingPen = new Pen(Color.Blue);
            Brush myDrawingBrush = new SolidBrush(Color.Blue);

            // x =
            // g.DrawLine(myDrawingPen,0,0,x,Size.Height);
            g.Transform = myMatrix;
            System.Console.WriteLine(myMatrix.Elements.ToString());
            g.FillPolygon(myDrawingBrush, triangle);
        }

        private void Form1_KeyPress(object sender,
KeyPressEventArgs e)
        {
            System.Console.WriteLine(e.KeyChar);
            foreach (float v in myMatrix.Elements) {
                System.Console.WriteLine(v.ToString());
            }
            if (e.KeyChar == 's')
            {
                dx = 0;
                dy = 0;
            }
            if (e.KeyChar == 'a')
            {
                dx = -1;
                dy = 0;
            }
            if (e.KeyChar == 'd') {
                dx = 1;
                dy = 0;
            }
            if (e.KeyChar == 'w')
            {
                dx = 0;
                dy = -1;
            }
            if (e.KeyChar == 'x')
            {
                dx = 0;
                dy = 1;
```

```
            }
            if (e.KeyChar == 'q') {
                dx = -1;
                dy = -1;
            }
            if (e.KeyChar == 'z')
            {
                dx = -1;
                dy = 1;
            }
            if (e.KeyChar == 'e')
            {
                dx = 1;
                dy = -1;
            }
            if (e.KeyChar == 'c') {
                dx = 1;
                dy = 1;
            }
            if (e.KeyChar == 'r')
            {
                if (dr == 0)
                {
                    dr = 1;
                }
                else {
                    dr *= 2;
                }
            }
            if (e.KeyChar == 'R')
            {
                if (dr==1) {
                    dr = 0;
                }
                else
                {
                    dr /= 2;
                }
            }
            if (e.KeyChar == 'f')
            {
                s = 2;
```

```
            }
            if (e.KeyChar == 'F')
            {
                s = 0.5F;
            }
        }
    }
}
```

When starting the program, the result obtained is like the one in the following figure.

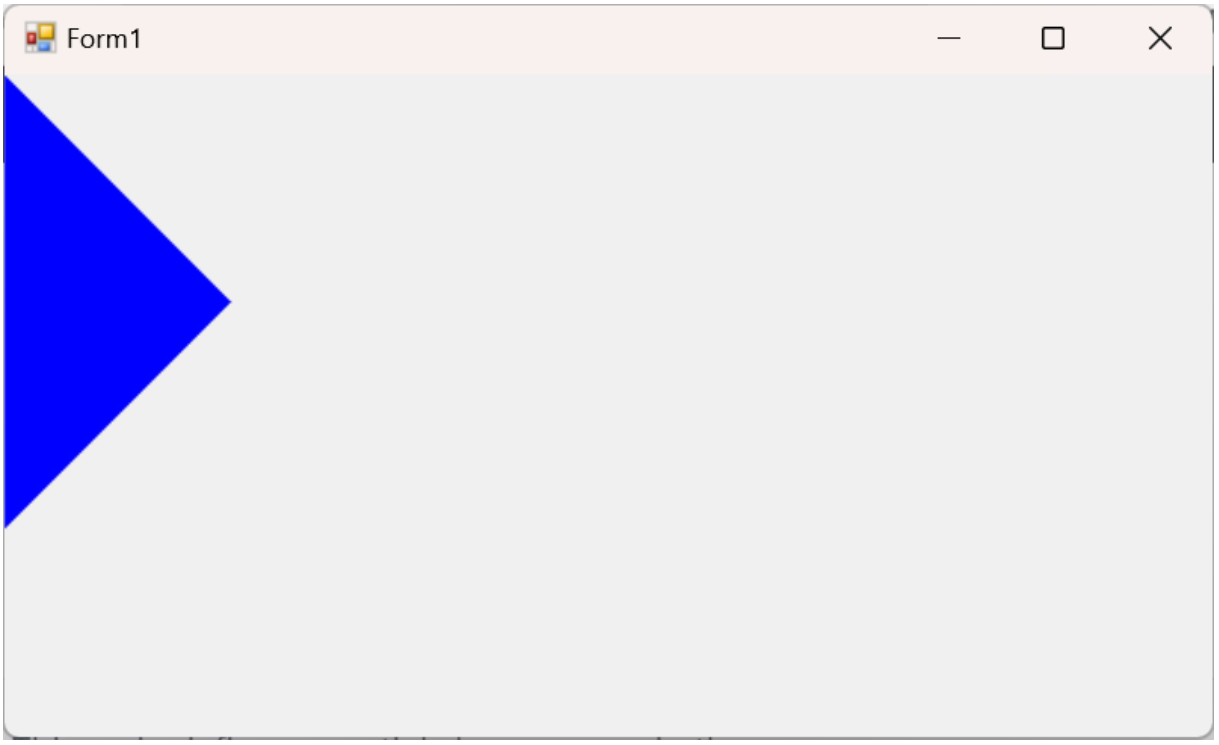

Figure 4. Program that animates a triangle

The following figures explain the operation of the keys.

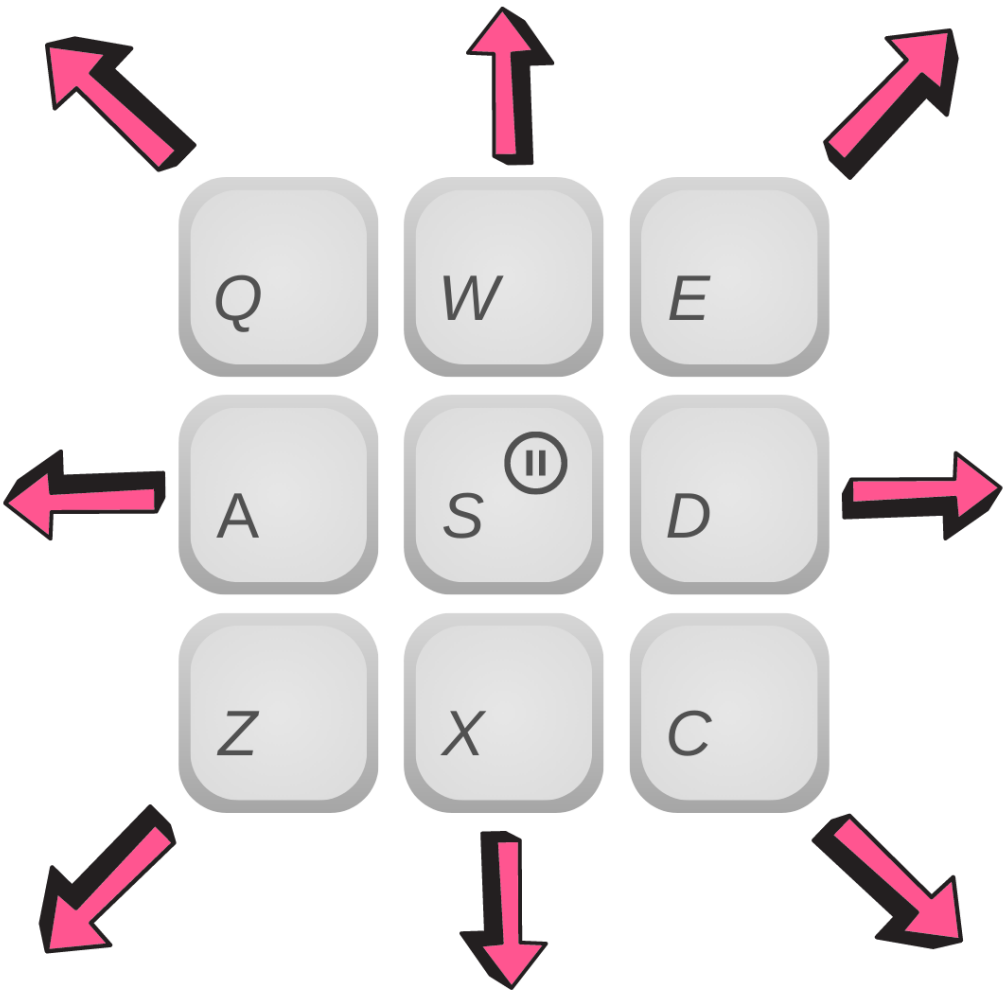

Figure 5. Keys to control direction of movement.

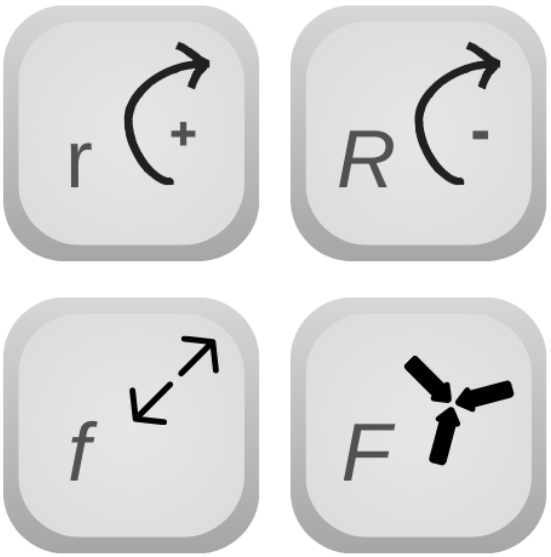

Figure 6.Keys to control scaling and rotation.

When pressing different keys, the result obtained is like the one in the following figure

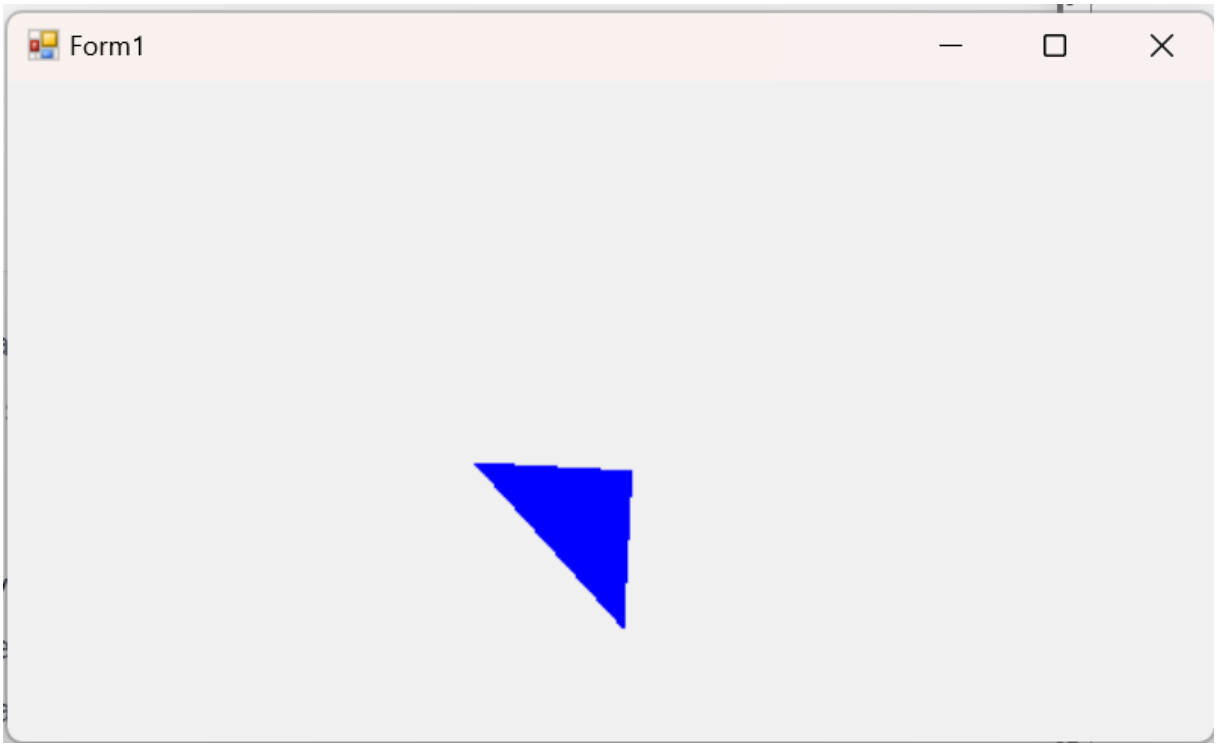

Figure 7. Program that animates a triangle when keys are pressed.

Conclusions and recommendations

Drawing graphics primitives in C# with Windows Forms is a powerful way to create visual representations for applications. By using transformation matrices and animation, these primitives can be manipulated in a variety of ways to create dynamic and engaging user interfaces. Understanding the concepts of graphics primitives, transformation matrices, and animation is essential for creating effective and visually appealing applications.

References

Microsoft. (n.d.). Graphics Class. Retrieved from https://docs.microsoft.com/en-us/dotnet/api/system.drawing.graphics?view=net-6.0

Wikipedia contributors. (2022, March 20). Transformation matrix. In Wikipedia. Retrieved April 9, 2023, from https://en.wikipedia.org/wiki/Transformation_matrix

Patel, D. (2017, October 12). C# Animation Tutorial. C# Corner. https://www.c-sharpcorner.com/article/c-sharp-animation-tutorial/