Project 2 Report – Pythagorean Tree Fractal

D11315807
Ardiawan Bagus Harisa
Department of CSIE

In this project, we are asked to create a Pythagorean fractal using three rectangles, with these specifications:

1. Different size: 3:4:5, 5:12:13, free ratio → choose using bottom right dropdown button. In case of 3:4:5 and 5:12:13, the alpha angle will be determined automatically.
2. Size of rectangles (default 100x100) → choose using size up-down button.
3. Color gradient → choose using color dropdown button.

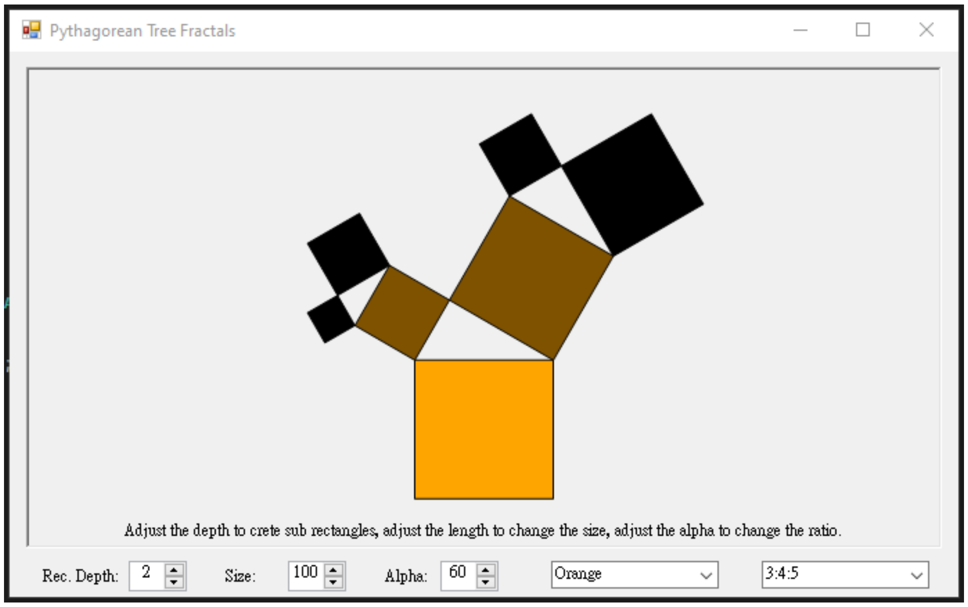
This time, I implement the homework using .NET C# in windows form app. This tutorial by Rod Stephens really gives the great step through the fractal generation process.

Setup

To be able to run the project, you just need to use Visual Studio (2022), .NET C#, and windows form SDK installed. Then you can just run the debug or release version of the .exe file.

How to Use

Here is the visual look of the Pythagorean fractal. In here, you can set the recursive depth of the tree, the size of the rectangle, the alpha (angle), and the color.



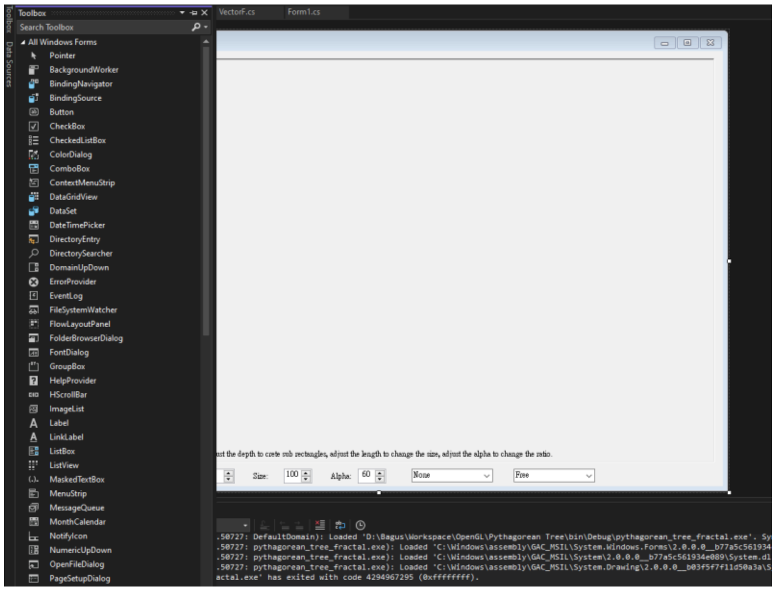
Adjust the depth to crete sub rectangles, adjust the length to change the size, adjust the alpha to change the ratio.

Rec. Depth: 2 Size: 100 Alpha: 60 Orange 3:4:5

Step-by-Step

Form1.cs

First, I create the form window and attach some UI elements from the toolbox such as text label, combo box dropdown, numeric up and down arrow, and a picture box. The picture box is the main element to perform the drawing of the fractal.

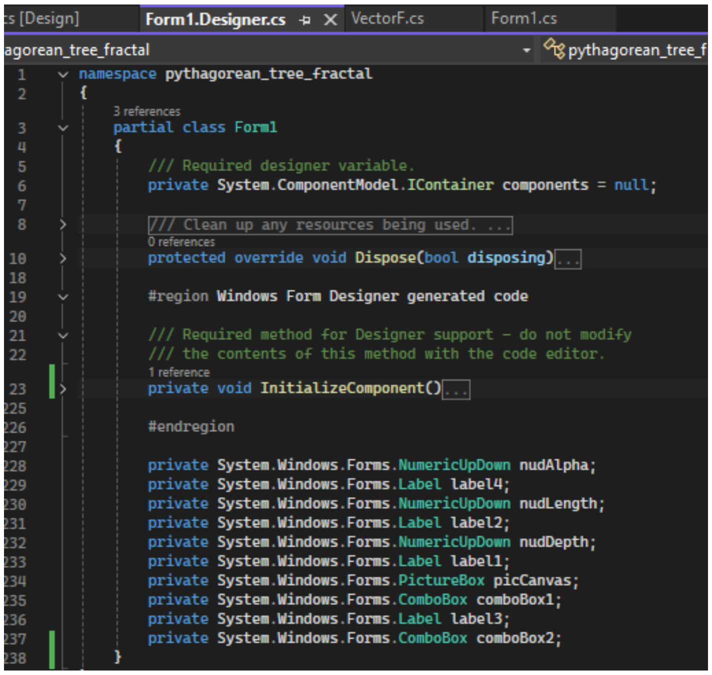


ust the depth to crete sub rectangles, adjust the length to change the size, adjust the alpha to change the ratio.

Size: 100 Alpha: 60 None Free

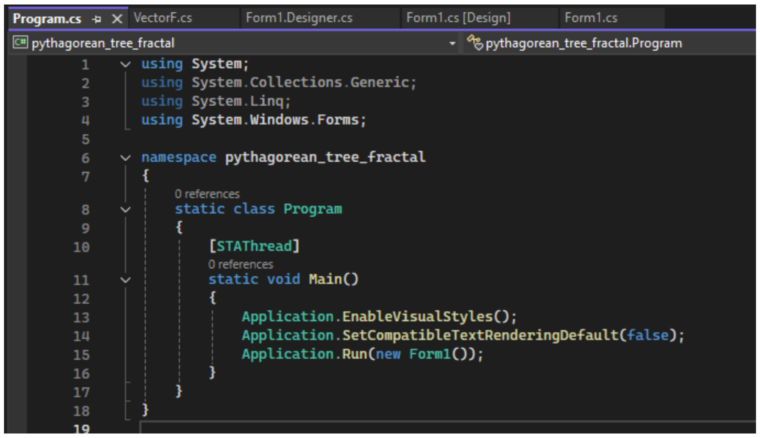
```
.50727: DefaultDomain): Loaded 'D:\Bagus\Workspace\OpenGL\Pythagorean Tree\bin\Debug\pythagorean_tree_fractal.exe'. Sy
.50727: pythagorean_tree_fractal.exe): Loaded 'C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934
.50727: pythagorean_tree_fractal.exe): Loaded 'C:\Windows\assembly\GAC_MSIL\System\2.0.0.0__b77a5c561934e089\System.dl
.50727: pythagorean_tree_fractal.exe): Loaded 'C:\Windows\assembly\GAC_MSIL\System.Drawing\2.0.0.0__b03f5f7f11d50a3a\S
actal.exe' has exited with code 4294967295 (0xffffffff).
```

And here is the code view of that form. Basically, in windows form app, there are two files (views): code and designer views. The code is basically generated version of the visual designer form. In the following code, we can see the list of the UI components.



```csharp
namespace pythagorean_tree_fractal
{
    3 references
    partial class Form1
    {
        /// Required designer variable.
        private System.ComponentModel.IContainer components = null;

        /// Clean up any resources being used. ...
        0 references
        protected override void Dispose(bool disposing)...

        #region Windows Form Designer generated code

        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        1 reference
        private void InitializeComponent()...

        #endregion

        private System.Windows.Forms.NumericUpDown nudAlpha;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.NumericUpDown nudLength;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.NumericUpDown nudDepth;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.PictureBox picCanvas;
        private System.Windows.Forms.ComboBox comboBox1;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.ComboBox comboBox2;
    }
```

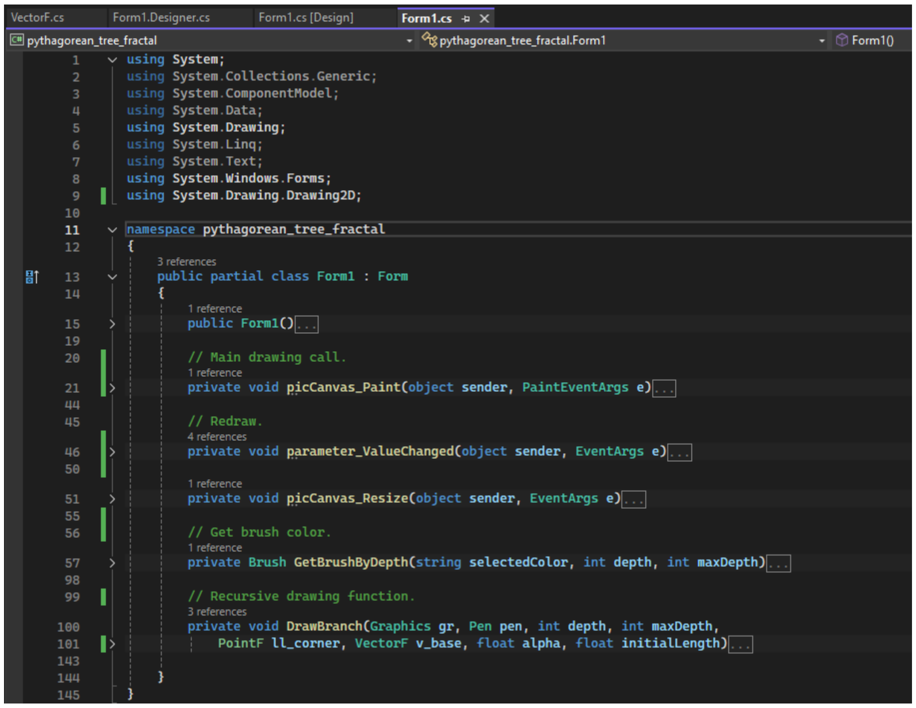
Generally, in Windows Form App, we can create as many forms as we want, then just call them onto the main program as follows. In here, I call the Form1 that I just created in the main function as entry point of the application. First, I enable the visual style of the form to follow the current modern style by calling the EnableVisualStyles(). Then, I set the application to use the newer version of text rendering (using GDI+ from .NET). Finally, I call the form that I created where I implemented the logic of the fractal.



```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace pythagorean_tree_fractal
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

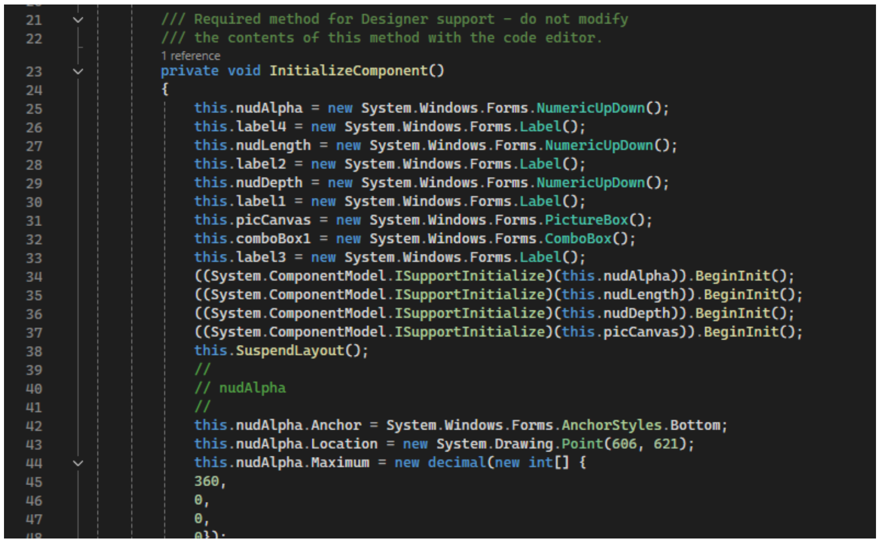
So, here is the steps to create the fractal:

1. Initialize all of the UI element.
2. Setup all of the logic variables including drawing function call.
    a. Here is where the recursive function to draw the tree is called.
3. I redraw if there is any change on the UI elements.



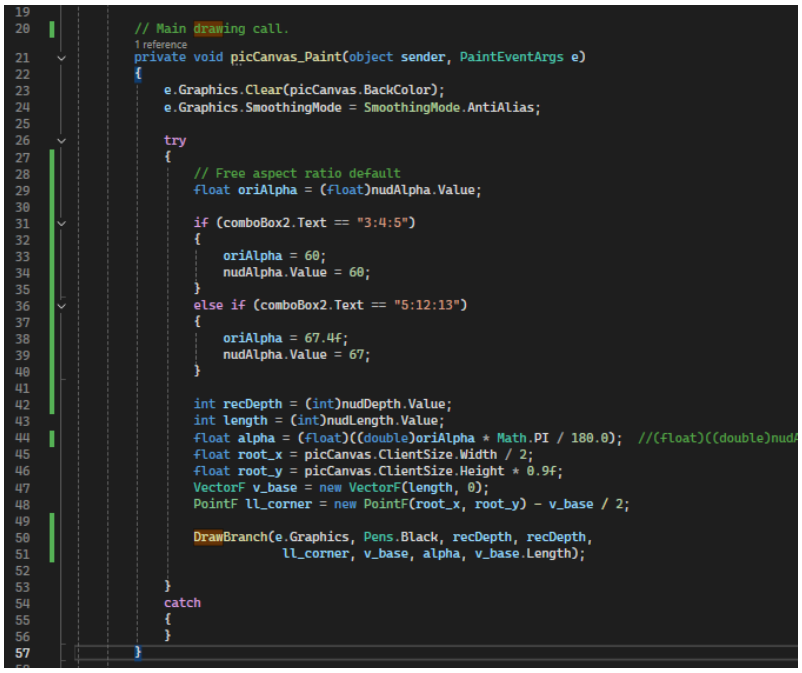
```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace pythagorean_tree_fractal
{
    public partial class Form1 : Form
    {
        public Form1()[...]

        // Main drawing call.
        private void picCanvas_Paint(object sender, PaintEventArgs e)[...]

        // Redraw.
        private void parameter_ValueChanged(object sender, EventArgs e)[...]

        private void picCanvas_Resize(object sender, EventArgs e)[...]

        // Get brush color.
        private Brush GetBrushByDepth(string selectedColor, int depth, int maxDepth)[...]

        // Recursive drawing function.
        private void DrawBranch(Graphics gr, Pen pen, int depth, int maxDepth,
            PointF ll_corner, VectorF v_base, float alpha, float initialLength)[...]

    }
}
```

1. **Initialize UI element**
   Basically, if we create the windows form using visual editor, the UI elements will all be automatically generated by the system. So, I don't need to do much. Probably just check and rename.



```
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        1 reference
        private void InitializeComponent()
        {
            this.nudAlpha = new System.Windows.Forms.NumericUpDown();
            this.label4 = new System.Windows.Forms.Label();
            this.nudLength = new System.Windows.Forms.NumericUpDown();
            this.label2 = new System.Windows.Forms.Label();
            this.nudDepth = new System.Windows.Forms.NumericUpDown();
            this.label1 = new System.Windows.Forms.Label();
            this.picCanvas = new System.Windows.Forms.PictureBox();
            this.comboBox1 = new System.Windows.Forms.ComboBox();
            this.label3 = new System.Windows.Forms.Label();
            ((System.ComponentModel.ISupportInitialize)(this.nudAlpha)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.nudLength)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.nudDepth)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.picCanvas)).BeginInit();
            this.SuspendLayout();
            //
            // nudAlpha
            //
            this.nudAlpha.Anchor = System.Windows.Forms.AnchorStyles.Bottom;
            this.nudAlpha.Location = new System.Drawing.Point(606, 621);
            this.nudAlpha.Maximum = new decimal(new int[] {
            360,
            0,
            0,
            0});
```

2. **Setup logical variables and call the drawing function**
   In this picCanvas_Paint(), I intend to call the recursive function to draw the fractal tree. First, I make sure to clean up the background color of the canvas that later will be used as the main medium to draw. Second, I assign the parameters of recursive depth, the size of the triangle, the angle of the triangle, the position of the tree, and finally just call the recursive drawing function.

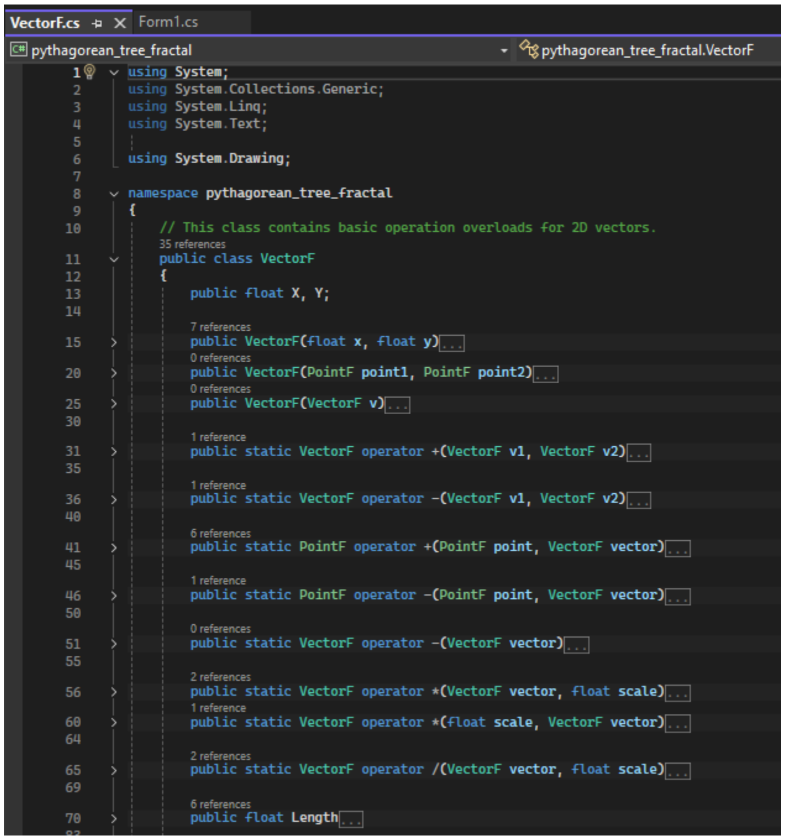


```
        // Main drawing call.
        1 reference
        private void picCanvas_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.Clear(picCanvas.BackColor);
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;

            try
            {
                // Free aspect ratio default
                float oriAlpha = (float)nudAlpha.Value;

                if (comboBox2.Text == "3:4:5")
                {
                    oriAlpha = 60;
                    nudAlpha.Value = 60;
                }
                else if (comboBox2.Text == "5:12:13")
                {
                    oriAlpha = 67.4f;
                    nudAlpha.Value = 67;
                }

                int recDepth = (int)nudDepth.Value;
                int length = (int)nudLength.Value;
                float alpha = (float)((double)oriAlpha * Math.PI / 180.0);  //(float)((double)nudA
                float root_x = picCanvas.ClientSize.Width / 2;
                float root_y = picCanvas.ClientSize.Height * 0.9f;
                VectorF v_base = new VectorF(length, 0);
                PointF ll_corner = new PointF(root_x, root_y) - v_base / 2;

                DrawBranch(e.Graphics, Pens.Black, recDepth, recDepth,
                        ll_corner, v_base, alpha, v_base.Length);

            }
            catch
            {
            }
        }
```

There you can see where I set the ratio of the triangle according to the user preference.

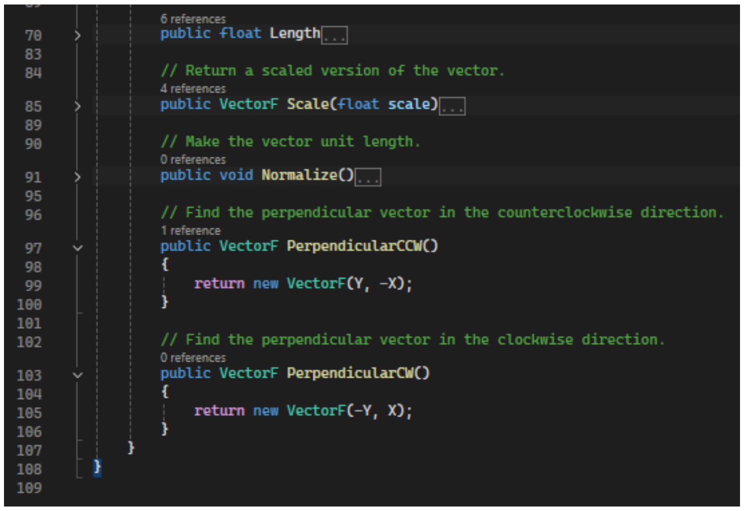
3. **Draw the tree**

To be able to draw the tree with modularity, I have a class to handle the vector operations, call VectorF. This class literally just handle the overload operations of vector and point.



```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Drawing;

namespace pythagorean_tree_fractal
{
    // This class contains basic operation overloads for 2D vectors.
    35 references
    public class VectorF
    {
        public float X, Y;

        7 references
        public VectorF(float x, float y)...
        0 references
        public VectorF(PointF point1, PointF point2)...
        0 references
        public VectorF(VectorF v)...

        1 reference
        public static VectorF operator +(VectorF v1, VectorF v2)...

        1 reference
        public static VectorF operator -(VectorF v1, VectorF v2)...

        6 references
        public static PointF operator +(PointF point, VectorF vector)...

        1 reference
        public static PointF operator -(PointF point, VectorF vector)...

        0 references
        public static VectorF operator -(VectorF vector)...

        2 references
        public static VectorF operator *(VectorF vector, float scale)...
        1 reference
        public static VectorF operator *(float scale, VectorF vector)...

        2 references
        public static VectorF operator /(VectorF vector, float scale)...

        6 references
        public float Length...
```

Here are some functions to handle the calculation of the angle to create the triangle. CW stands for clockwise, and CCW stands for counter clockwise.

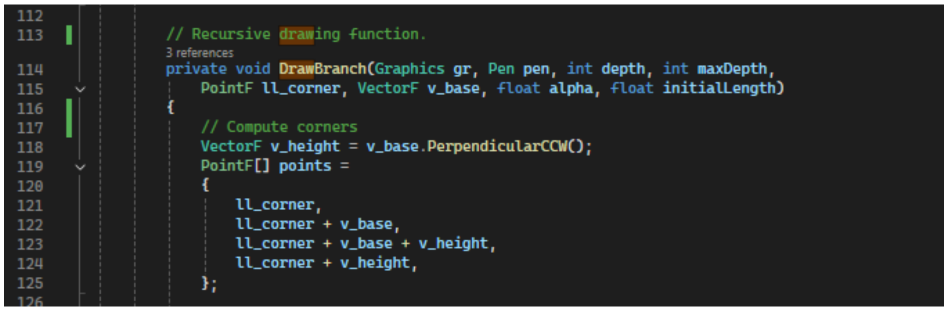


```csharp
        6 references
        public float Length...

        // Return a scaled version of the vector.
        4 references
        public VectorF Scale(float scale)...

        // Make the vector unit length.
        0 references
        public void Normalize()...

        // Find the perpendicular vector in the counterclockwise direction.
        1 reference
        public VectorF PerpendicularCCW()
        {
            return new VectorF(Y, -X);
        }

        // Find the perpendicular vector in the clockwise direction.
        0 references
        public VectorF PerpendicularCW()
        {
            return new VectorF(-Y, X);
        }
    }
}
```

The graphics and pen type from the argument are data type in .NET graphic library to render the fractal tree. The use of Graphics gr is to draw shapes like polygons and lines. So, the graphics gr is passed to the main drawing canvas. The object pen will define the style of the lines to draw the outline of the rectangles.

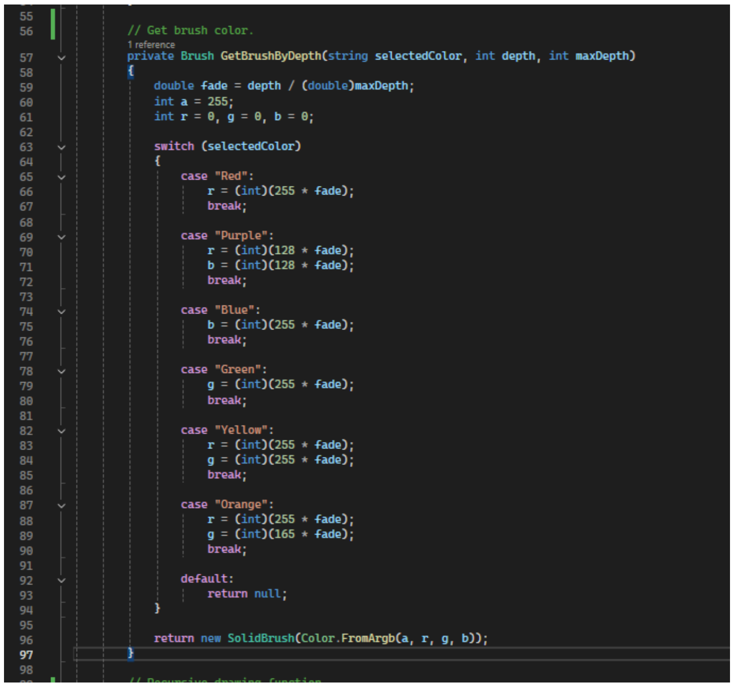
To render the rectangle, we need to calculate its corner points. First, we need to determine the base width and orientation of the rectangle, using vector, in the counter clockwise direction. Then we calculate the all four points in array of points. We use lower left corner as a base to compute the other. Here is the order:

1. Lower left
2. Lower right
3. Upper right
4. Upper left

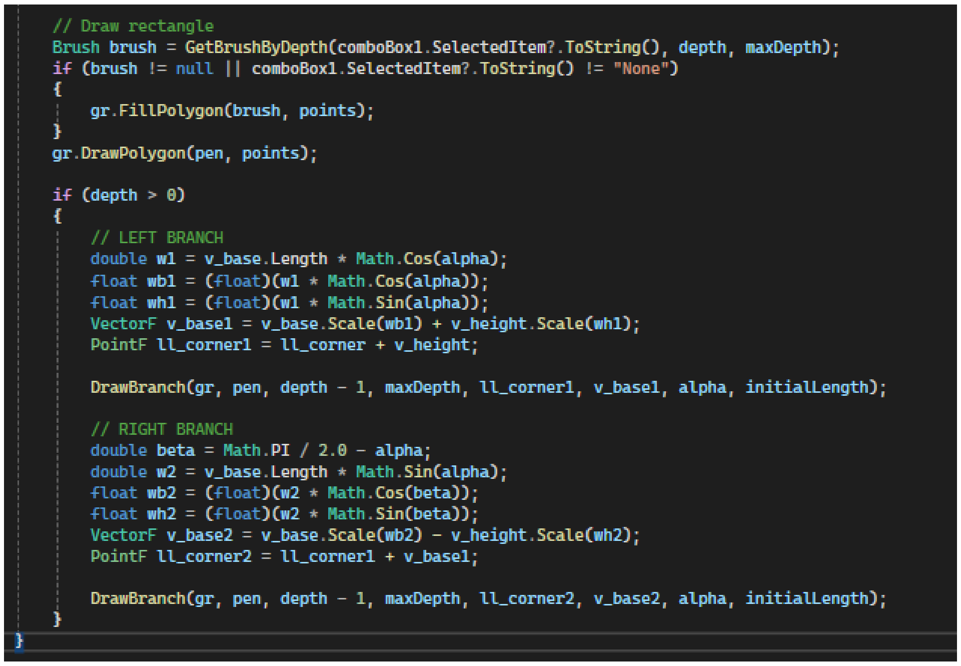


```
        // Recursive drawing function.
        3 references
        private void DrawBranch(Graphics gr, Pen pen, int depth, int maxDepth,
            PointF ll_corner, VectorF v_base, float alpha, float initialLength)
        {
            // Compute corners
            VectorF v_height = v_base.PerpendicularCCW();
            PointF[] points =
            {
                ll_corner,
                ll_corner + v_base,
                ll_corner + v_base + v_height,
                ll_corner + v_height,
            };
```

Then, I draw the rectangle(s). First, I need to prepare the drawing tool: brush and pen. Those are used to define the style of the drawing. For instance, the color of the rectangle is defined by brush, which is defined by the dropdown button and the depth of the tree.



```
// Get brush color.
1 reference
private Brush GetBrushByDepth(string selectedColor, int depth, int maxDepth)
{
    double fade = depth / (double)maxDepth;
    int a = 255;
    int r = 0, g = 0, b = 0;

    switch (selectedColor)
    {
        case "Red":
            r = (int)(255 * fade);
            break;

        case "Purple":
            r = (int)(128 * fade);
            b = (int)(128 * fade);
            break;

        case "Blue":
            b = (int)(255 * fade);
            break;

        case "Green":
            g = (int)(255 * fade);
            break;

        case "Yellow":
            r = (int)(255 * fade);
            g = (int)(255 * fade);
            break;

        case "Orange":
            r = (int)(255 * fade);
            g = (int)(165 * fade);
            break;

        default:
            return null;
    }

    return new SolidBrush(Color.FromArgb(a, r, g, b));
}
```



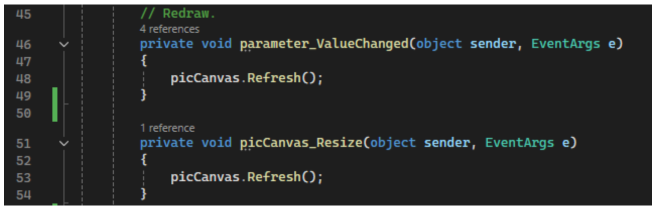
```
    // Draw rectangle
    Brush brush = GetBrushByDepth(comboBox1.SelectedItem?.ToString(), depth, maxDepth);
    if (brush != null || comboBox1.SelectedItem?.ToString() != "None")
    {
        gr.FillPolygon(brush, points);
    }
    gr.DrawPolygon(pen, points);

    if (depth > 0)
    {
        // LEFT BRANCH
        double w1 = v_base.Length * Math.Cos(alpha);
        float wb1 = (float)(w1 * Math.Cos(alpha));
        float wh1 = (float)(w1 * Math.Sin(alpha));
        VectorF v_base1 = v_base.Scale(wb1) + v_height.Scale(wh1);
        PointF ll_corner1 = ll_corner + v_height;

        DrawBranch(gr, pen, depth - 1, maxDepth, ll_corner1, v_base1, alpha, initialLength);

        // RIGHT BRANCH
        double beta = Math.PI / 2.0 - alpha;
        double w2 = v_base.Length * Math.Sin(alpha);
        float wb2 = (float)(w2 * Math.Cos(beta));
        float wh2 = (float)(w2 * Math.Sin(beta));
        VectorF v_base2 = v_base.Scale(wb2) - v_height.Scale(wh2);
        PointF ll_corner2 = ll_corner1 + v_base1;

        DrawBranch(gr, pen, depth - 1, maxDepth, ll_corner2, v_base2, alpha, initialLength);
    }
}
```

Then using the following recursive concept, I draw the rectangles.

1. Draw 1 rectangle
2. If depth > 0, draw two more rectangle
3. Recursive call

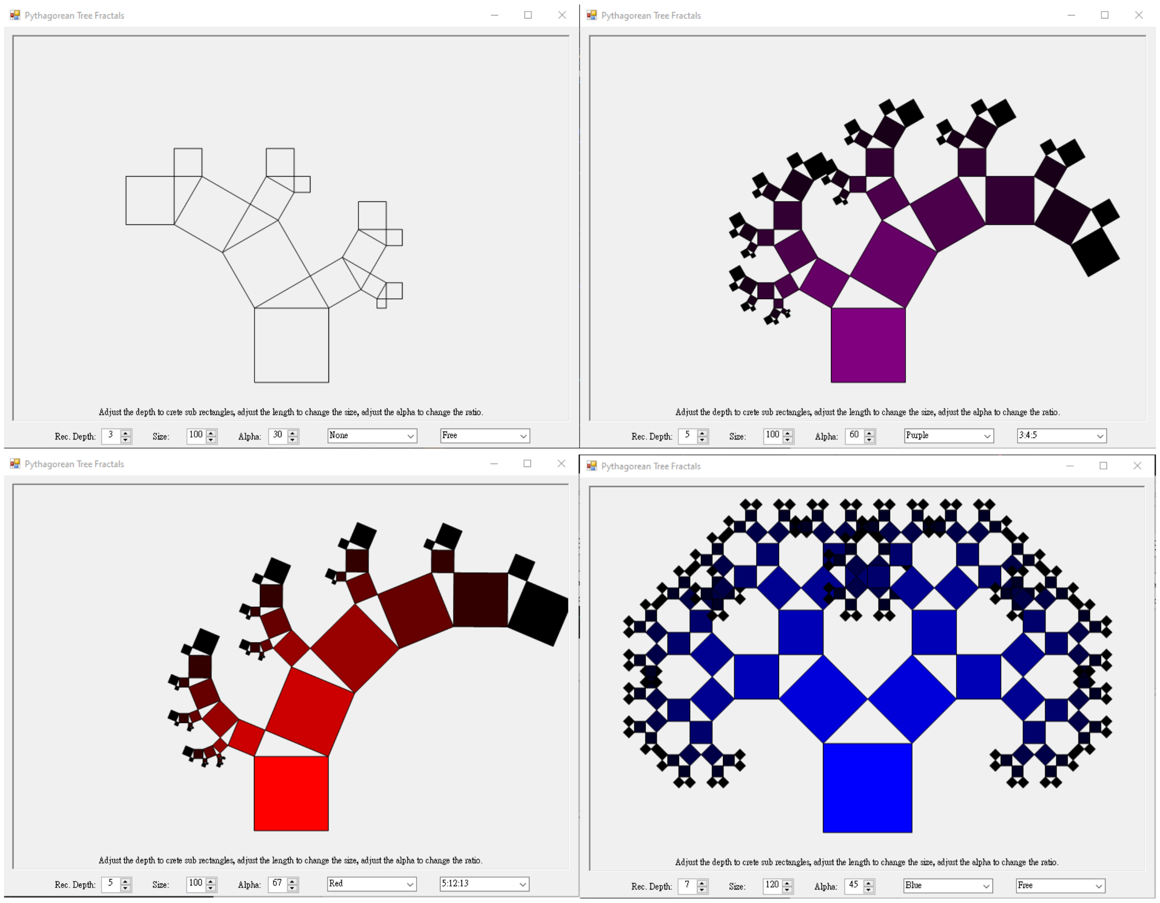
So, first, draw a rectangle as the base, as I mentioned before. Then, if the depth is more than zero, meaning I should draw two branches. Then, draw the left branch of the tree by calculating the base size and its relative position and orientation by calling its recursive function. Then, perform the similar call to the other (right) branch. The process will continue until there is no more depth to perform (depth = 0).

And these two functions will be called to redraw in case any update from the form (i.e., user edits the value).



```
// Redraw.
4 references
private void parameter_ValueChanged(object sender, EventArgs e)
{
    picCanvas.Refresh();
}

1 reference
private void picCanvas_Resize(object sender, EventArgs e)
{
    picCanvas.Refresh();
}
```

Simulations


Pythagorean Tree Fractals
Pythagorean Tree Fractals
Pythagorean Tree Fractals
Pythagorean Tree Fractals

Adjust the depth to crete sub rectangles, adjust the length to change the size, adjust the alpha to change the ratio.

Rec. Depth: 3 Size: 100 Alpha: 30 None Free

Adjust the depth to crete sub rectangles, adjust the length to change the size, adjust the alpha to change the ratio.

Rec. Depth: 5 Size: 100 Alpha: 60 Purple 3:4:5

Adjust the depth to crete sub rectangles, adjust the length to change the size, adjust the alpha to change the ratio.

Rec. Depth: 5 Size: 100 Alpha: 67 Red 5:12:13

Adjust the depth to crete sub rectangles, adjust the length to change the size, adjust the alpha to change the ratio.

Rec. Depth: 7 Size: 120 Alpha: 45 Blue Free

The complete project can be downloaded at my GitHub:
https://github.com/ardiawanbagusharisa/cgopengl/tree/main/Pythagorean%20Tree%20(WFA)