CS 538: Assignment 03

Programming Assignments (90%)

Setup

- **This assignment is in C++.**
- **Pull and merge from my repository to get the latest version of repository.**

Rasterize.cpp

For this assignment, you will concentrate on implementing DDA and Midpoint line drawing algorithms, as well as a basic fill algorithm using barycentric coordinates.

For the line drawing, if wireframe is true, draw white lines; otherwise, you should linearly interpolate the color from startVert.color to endVert.color.

When swapping is required, you may use **std::swap(a,b).**

Note that these drawing functions are meant to generate fragments to add to the list of fragments!

- **void drawLineDDA(Vert &startVert, Vert &endVert, vector<Fragment> &fragList, bool wireframe)**
    - Round the starting and ending positions to the nearest integer (roundV() function and Vec3i) → **start** and **end**
    - Compute the integer differences in x and y → **dx** and **dy**
    - Set the number of steps as either abs(dx) or abs(dy), depending on which is greater → **steps**
    - Compute the floating point increments in x and y → **xInc** and **yInc**
    - Get the floating point starting x and y coordinates from start → **x** and **y**
    - Get the starting color from startVert.color → **color**
    - Compute the color increment as the ending color minus the starting color divided by the number of steps → **colorInc**
    - If wireframe is true, change color to white (1.0f, 1.0f, 1.0f, 1.0f) and colorInc to all zeros.
    - Each fragment will consist of the following:
        - The current position rounded to the nearest integer (with z = 0)
        - The current color
    - Add the first fragment to fragList
    - For each step:
        - Increment x and y by xInc and yInc, respectively
        - Increment color by colorInc
        - Add the next fragment to fragList

- **void drawLineMid(Vert &startVert, Vert &endVert, vector<Fragment> &fragList, bool wireframe)**
    - Round the starting and ending positions to the nearest integer (roundV() function and Vec3i) → **start** and **end**
    - Compute the integer differences in x and y → **dx** and **dy**
    - If abs(dy) is greater than abs(dx)
        - Note that swapping x and y will need to be performed → **swap**
        - Swap dx and dy
        - Swap the x and y coordinates of start
        - Swap the x and y coordinates of end
    - Set the starting and ending color → **startColor** and **endColor**
    - If x is going in the NEGATIVE direction (dx < 0)
        - Swap start and end
        - Swap startColor and endColor
        - Negate dx
        - Negate dy
    - Set starting y as start.y → **y**
    - Set y increment to +1 → **yInc**
    - If y is going in the NEGATIVE direction (dy < 0)
        - Set the y increment to -1 → **yInc**
        - Negate dy
        - Swap y coordinates of start and end
    - Create an ImplicitLine<float> from start and end → **line**
    - Calculate initial value of decision variable as line.eval(start.x + 1.0f, start.y + 0.5f) → **d**
    - Calculate number of steps as end.x – start.x
    - Get the starting color from startVert.color → **color**
    - Compute the color increment as the ending color minus the starting color divided by the number of steps → **colorInc**
    - If wireframe is true, change color to white (1.0f, 1.0f, 1.0f, 1.0f) and colorInc to all zeros.
    - For each x value from start.x to end.x (inclusive):
        - Create a Fragment → **f**
        - If we need to swap x and y → set f.pos equal to (y,x,0)
        - Otherwise → set f.pos equal to (x,y,0)
        - Set f.color to the current color
        - Add the fragment to fragList
        - Increment color by colorInc
        - If the decision variable is negative (d < 0)
            - Increment y by yInc
            - Increment d by (dx – dy)
        - Otherwise:
            - Decrement d by dy

- **void fillTriangle(vector<Vert> &vertices, Face &face, vector<Fragment> &fragList)**
    - Create a floating-point bounding box (BoundBoxf and the computeBounds function) from the vertices and the face
    - Convert to an integer bounding box (BoundBoxi and convertBoundBox)
    - Grab the first three vertices that make up this particular face → **vA, vB, vC**
    - Create a BaryData instance from vA, vB, and vC → **bd**
    - Loop through each position covered by the (integer) bounding box:
        - Compute the barycentric coordinates (barycentric() function) → bary
        - If each barycentric coordinate is STRICTLY greater than 0:
            - Create a Fragment using the computeFragment() function
            - Add the fragment to fragList

Screenshots (10%)

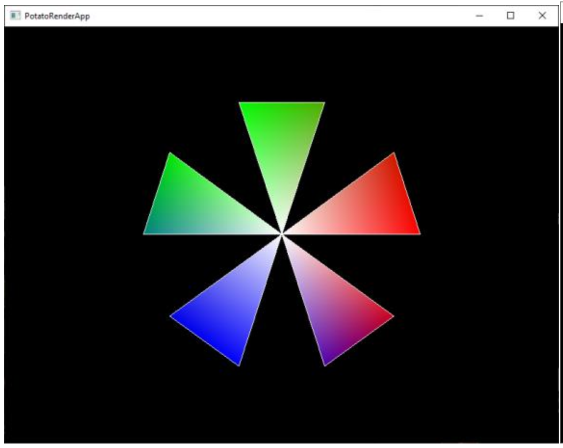
To change how the object is rendered, you will want to look at the following settings in include/Settings.hpp:

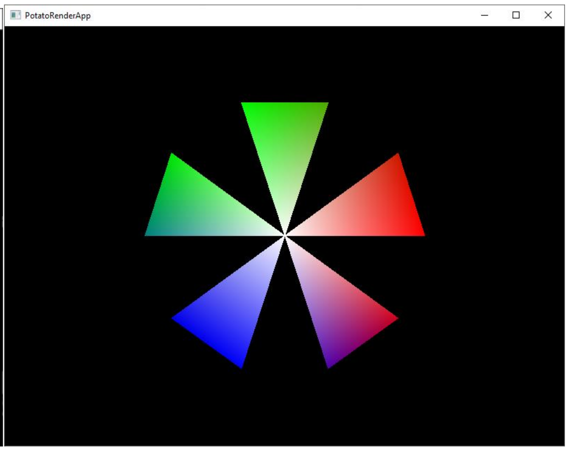
- **GEO_FAN_BLADE_CNT**
    - How many "blades" of the geometric fan
- **DRAW_LINES_AS_WIREFRAME**
    - If true, draw lines as white; otherwise, interpolate color
- **LINE_ALGORITHM**
    - Either LINE_DDA or LINE_MID, depending on which algorithm is desired

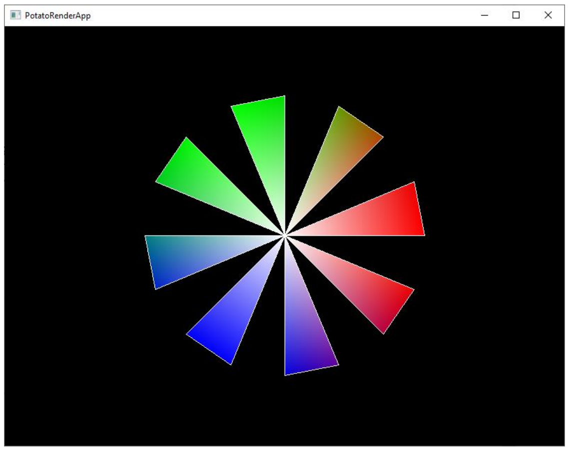
For this assignment, **take FOUR screenshots:**

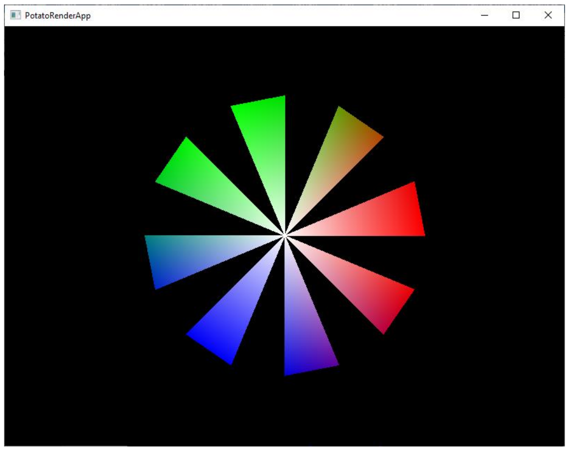
- 5 blades, wireframe, DDA
- 5 blades, no wireframe, DDA
- 8 blades, wireframe, midpoint
- 8 blades, no wireframe, midpoint

Save these into the **screenshots** folder









Grading

Your OVERALL assignment grade is weighted as follows:

- 90% - Programming
- 10% - Screenshots