<u>PS1</u>: *Recursive Graphics (Sierpinski's Triangle)*

In this assignment, we were to create a Sierpinski triangle by using recursion to draw the triangle. We needed be able to plot the base triangle and then recessively drawing smaller or bigger triangles outside or inside the base triangle. (Depending if you choose to make the base triangle small or large. We would then create our own original shape doing the same procedure as the Sierpinski.

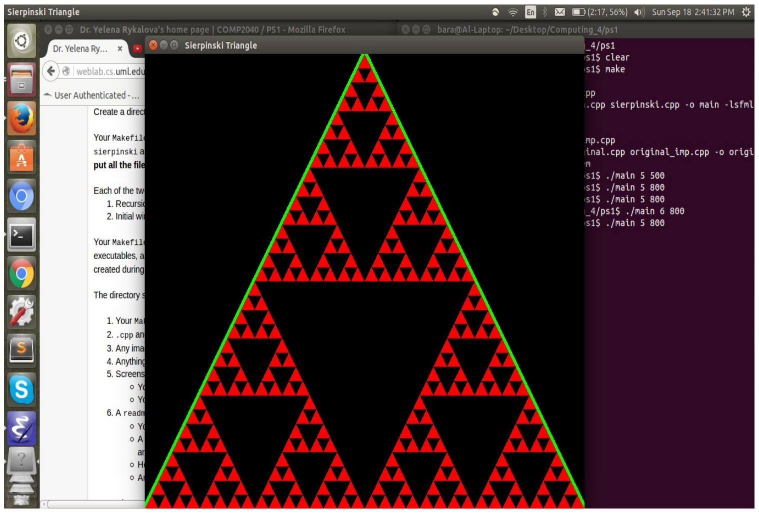
Sierpinski Triangle:

In the Sierpinski implantation, I created a class called Sierpinski that inherited the drawable library. This was used to be able to have a type convex shape to draw the base triangle and recursive triangle. The draw function was used to call each time we were to need to draw the function. When the user enter, I needed to set the point using the setPoint function in the sf::Convex Shape. In the recusrive function, I would draw the points by dividing the 2 points to get the middle point. After drawing it, I would set the top, left, and right to different points to get the next mid points. Then take the amount of recursion depth and recursively call the function until 0.

Original:

I did everything the same idea as the Sierpinski triangle but I instead used 4 points instead of 3. I also changed the coordinates of the square to fit the screen.

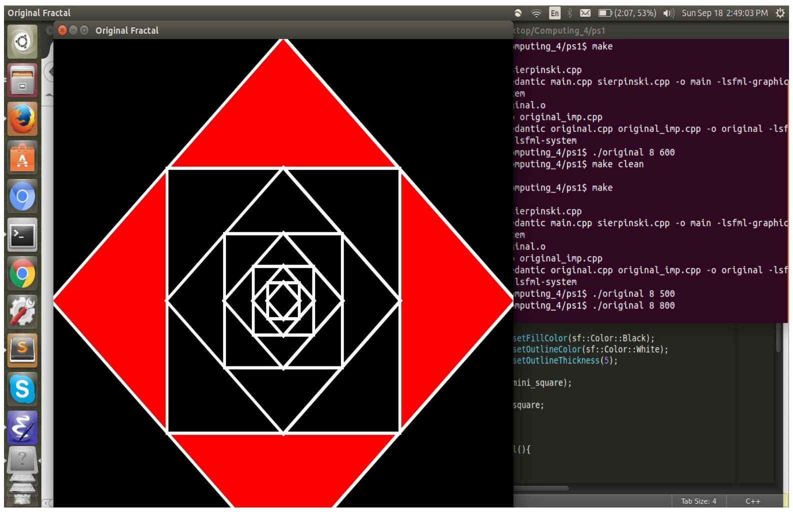
In this assignment, I learned to figure out how to better use the sf::Convex Shape library, using functions like setPoint, getPoint, setFillColor, etc. I was able to better understand recursion and use it with other functions. Also learned the full use of type Vector2f.


Sierpinski Triangle
(2:17, 56%) Sun Sep 18 2:41:32 PM
Dr. Yelena Rykalova's home page | COMP2040 / PS1 - Mozilla Firefox
bara@Al-Laptop: ~/Desktop/Computing_4/ps1
Sierpinski Triangle

```
_4/ps1
s1$ clear
s1$ make

pp
.cpp sierpinski.cpp -o main -lsfml

mp.cpp
inal.cpp original_imp.cpp -o origi
m
s1$ ./main 5 500
s1$ ./main 5 800
s1$ ./main 5 800
_4/ps1$ ./main 6 800
s1$ ./main 5 800
```



```
mputing_4/ps1$ make

ierpinski.cpp
dantic main.cpp sierpinski.cpp -o main -lsfml-graphic
em
inal.o
 original_imp.cpp
dantic original.cpp original_imp.cpp -o original -lsf
lsfml-system
mputing_4/ps1$ ./original 8 600
mputing_4/ps1$ make clean

mputing_4/ps1$ make

ierpinski.cpp
dantic main.cpp sierpinski.cpp -o main -lsfml-graphic
em
inal.o
 original_imp.cpp
dantic original.cpp original_imp.cpp -o original -lsf
lsfml-system
mputing_4/ps1$ ./original 8 500
mputing_4/ps1$ ./original 8 800
```

```
setFillColor(sf::Color::Black);
setOutlineColor(sf::Color::White);
setOutlineThickness(5);

mini_square);

square;


l(){
```

```
cc = g++
#all


all : main original

main : main.o sierpinski.o
        $(cc) -Wall -Werror -ansi -pedantic main.cpp sierpinski.cpp -o sierpinski -lsfml-graphics -lsfml-window -lsfml-system

main.o : sierpinski.hpp
        $(cc) -c main.cpp -o main.o

sierpinski : sierpinski.cpp sierpinski.hpp
        $(cc) -c sierpinski.cpp -o sierpinski.o


original : original.o original_imp.o
        $(cc) -Wall -Werror -ansi -pedantic original.cpp original_imp.cpp -o original -lsfml-graphics -lsfml-window -lsfml-system

original.o : original.hpp
        $(cc) -c original.cpp -o original.o

original_imp : original_imp.cpp original.hpp
        $(cc) -c original_imp.cpp -o original_imp.o

clean:
        rm *.o main original

run:
                ./sierpinski 5 800
                ./original 8 400

debug: cc += -g
debug: main original
```

```
/*
        Name: Albara Mehene
        Date:9/17/2016
        Computing IV
*/
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include "sierpinski.hpp"

int main(int argc, char* argv[]){

        if(argc < 3){

                        std::cout<<"sierpinski [recursion-depth] [side-length]"<< std::endl;
                        return -1;
        }
        //Atoi connverts strings to integers
        int depth = atoi(argv[1]);
        int side = atoi(argv[2]);

         sf::RenderWindow
                  window(sf::VideoMode(side,(int)(.5*sqrt(3.)*(float)side)), "Sierpinski Triangle");

        Sierpinski sierpinski(depth, side);


         window.setFramerateLimit(1);

          while(window.isOpen())
         {
                 sf::Event event;
                 while(window.pollEvent(event))
                 {
                               if(event.type == sf::Event::Closed)
                                        window.close();
                 }
                 window.clear();
                 window.draw(sierpinski);
                 window.display();
         }

         return 0;
}
```

```cpp








#include <cmath>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

class Sierpinski : public sf::Drawable
{
        public:

                Sierpinski(int N, int size_tri);

                void sierpinski(sf::ConvexShape mid_triangle, int recursion, sf::RenderTarget& target) const;

                sf::ConvexShape filledtriangle(sf::Vector2f left_tri, sf::Vector2f bottom_tri, sf::Vector2f right_tri,sf::RenderTarget& target) const;
                //destructor;
                ~Sierpinski();

        private:
                sf::ConvexShape triangle;

                int depth_;
                int side_;


                void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;


};
```

```
#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include "sierpinski.hpp"

void Sierpinski::draw(sf::RenderTarget& target, sf::RenderStates states) const{
        target.draw(triangle, states);
        sierpinski(triangle, depth_, target);

}

Sierpinski::Sierpinski(int N, int size_tri){

        side_ = size_tri;
        depth_ = N;


        triangle.setPointCount(3);
        triangle.setPoint(0, sf::Vector2f(0,side_ *(sqrt(3)/2)));//left
        triangle.setPoint(1, sf::Vector2f(side_,side_*(sqrt(3)/2)));//right
        triangle.setPoint(2, sf::Vector2f((side_/2),0));//top


        triangle.setFillColor(sf::Color::Red);
        triangle.setOutlineColor(sf::Color::Green);
        triangle.setOutlineThickness(5);
}


void Sierpinski::sierpinski(sf::ConvexShape mid_triangle, int recursion,sf::RenderTarget& target) const{
        sf::Vector2f left,right,top;
        sf::Vector2f mid_lefttop, mid_leftright, mid_topright;
        sf::ConvexShape temp1_tri, temp2_tri, temp3_tri;

        if(recursion == 0){
                return;
        }
        else{
                left = mid_triangle.getPoint(0);
                right = mid_triangle.getPoint(1);
                top = mid_triangle.getPoint(2);

                mid_lefttop.x = (left.x + top.x)/2;
                mid_lefttop.y = (left.y + top.y)/2;

                mid_leftright.x = (left.x + right.x)/2;
                mid_leftright.y = (left.y + right.y)/2;

                mid_topright.x = (top.x + right.x)/2;
                mid_topright.y = (top.y + right.y)/2;

                temp1_tri = filledtriangle(mid_lefttop,mid_leftright, mid_topright,target);
                temp2_tri = temp1_tri;
                temp3_tri = temp1_tri;

                temp1_tri.setPoint(2,left);
                temp2_tri.setPoint(0,right);
```

```cpp
                temp3_tri.setPoint(1,top);

                sierpinski(temp1_tri, recursion - 1, target);
                sierpinski(temp2_tri, recursion - 1, target);
                sierpinski(temp3_tri, recursion - 1, target);




        }



}

sf::ConvexShape Sierpinski::filledtriangle(sf::Vector2f left_tri, sf::Vector2f bottom_tri, sf::Vector2f right_tri,sf::RenderTarget& target) const{

        sf::ConvexShape small_triangle;

        small_triangle.setPointCount(3);
        small_triangle.setPoint(0, left_tri);
        small_triangle.setPoint(1, bottom_tri);
        small_triangle.setPoint(2, right_tri);

        small_triangle.setFillColor(sf::Color::Black);

        target.draw(small_triangle);

        return small_triangle;
}

Sierpinski::~Sierpinski(){

}
```

```
/*
        Name: Albara Mehene
        Date:9/18/2016
        Computing IV
*/

#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

#include "original.hpp"

int main(int argc, char* argv[]){

        if(argc < 3){

                        std::cout<<"Fractal [recursion-depth] [side-length]" << std::endl;
                        return -1;
        }
        //Atoi connverts strings to integers
        int depth_ = atoi(argv[1]);
        int side_ = atoi(argv[2]);

         sf::RenderWindow window(sf::VideoMode(side_,side_), "Original Fractal");

        Fractal frac(depth_, side_);

                window.setFramerateLimit(1);

                 while(window.isOpen())
                 {
                          sf::Event event;
                          while(window.pollEvent(event))
                          {
                                        if(event.type == sf::Event::Closed)
                                                 window.close();
                          }
                          window.clear();
                          window.draw(frac);
                          window.display();
                 }

                return 0;
        }
```

```
#include <cmath>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

class Fractal : public sf::Drawable
{
        public:

         Fractal(int n, int size_frac);

         void fractal_rec(sf::ConvexShape fractal_shape, int recursion, sf::RenderTarget &target) const;

         sf::ConvexShape filledFractal(sf::Vector2f point1,sf::Vector2f point2, sf::Vector2f point3, sf::Vector2f point4,sf::RenderTarget &target) const;

         ~Fractal();




        private:
        sf::ConvexShape square;

        int depth;
        int side;

        void virtual draw(sf::RenderTarget& target, sf::RenderStates states) const;

};
```

```cpp
#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>

#include "original.hpp"


void Fractal::draw(sf::RenderTarget& target, sf::RenderStates states) const{
        target.draw(square, states);
        fractal_rec(square, depth, target);
}

Fractal::Fractal(int N, int size_frac){
        side = size_frac;
        depth = N;

        square.setPointCount(4);
        square.setPoint(0, sf::Vector2f(side/2, 0));//top
        square.setPoint(1, sf::Vector2f(0, side/2));//left
        square.setPoint(2, sf::Vector2f(side/2, side));//bottom
        square.setPoint(3, sf::Vector2f(side, side/2));//right

        square.setFillColor(sf::Color::Red);
        square.setOutlineColor(sf::Color::White);
        square.setOutlineThickness(5);
}
void Fractal::fractal_rec(sf::ConvexShape fractal_shape, int recursion, sf::RenderTarget &target) const{
        sf::Vector2f left,top,right,bottom;
        sf::Vector2f mid_lefttop, mid_topright, mid_rightbottom, mid_bottomleft;

        sf::ConvexShape temp1_sqr, temp2_sqr, temp3_sqr, temp4,sqr;

        if(recursion == 0){
                return;
        }
        else{
                top = fractal_shape.getPoint(0);
                left = fractal_shape.getPoint(1);
                bottom = fractal_shape.getPoint(2);
                right = fractal_shape.getPoint(3);

                mid_lefttop.x = (left.x + top.x)/2;
                mid_lefttop.y = (left.y + top.y)/2;

                mid_topright.x = (top.x + right.x)/2;
                mid_topright.y = (top.y + right.y)/2;

                mid_rightbottom.x = (right.x + bottom.x)/2;
                mid_rightbottom.y = (right.y + bottom.y)/2;

                mid_bottomleft.x = (left.x + bottom.x)/2;
                mid_bottomleft.y = (left.y + bottom.y)/2;

                temp1_sqr = filledFractal(mid_lefttop, mid_topright, mid_rightbottom, mid_bottomleft, target);

                fractal_rec(temp1_sqr, recursion - 1, target);
        }

}
sf::ConvexShape Fractal::filledFractal(sf::Vector2f point1,sf::Vector2f point2, sf::Vector2f point3, sf::Vector2f point4,sf::RenderTarget &target) const{
        sf::ConvexShape mini_square;

        mini_square.setPointCount(4);
        mini_square.setPoint(0, point1);
        mini_square.setPoint(1, point2);
        mini_square.setPoint(2, point3);
        mini_square.setPoint(3, point4);

        mini_square.setFillColor(sf::Color::Black);
        mini_square.setOutlineColor(sf::Color::White);
        mini_square.setOutlineThickness(5);

        target.draw(mini_square);

        return mini_square;

}

Fractal::~Fractal(){

}
```