BLG458E
Homework-1

22.11.2023
Res. Asst. Yusuf Huseyin Sahin
sahinyu@itu.edu.tr

*M*inu ees laual on hüperboloid
Insener Garini hüperboloid
Pimestav kiir
Silmipimestav kiir
Tunnete piir
Meie tunnete piir
See on ju hukatus

Elu ja surm
Meie elu ja surm
Taevalik hurm
Nõnda taevalik hurm
See on ju igavik hüperboloid

Vennaskond

- You should write all your code in Haskell language.
- Cheating is highly discouraged. If you are planning to use different libraries or functions, please ask me about it.
- In your report please describe how to run your functions.

1 Introduction: Points, Triangles and STL

Standard Triangle Language (STL) is a standard to store Mesh objects. In STL meshes, the object is represented as a list of triangles. An example mesh object, containing only one triangle is given below.

```
solid Object01
  facet
    outer loop
      vertex -178.53533151862527 152.7531362784853 713.0844789743423
      vertex -173.53533201862527 152.7531362784853 713.0844789743423
```

```
      vertex -173.53533201862527 152.7531362784853 708.0844789743423
    endloop
  endfacet
endsolid Object01
```

Here, a solid object named **Object01** is created. Inside this object, there is only one **facet**. The facet is created from three **vertices** having xyz coordinates.

I have provided a skeleton code with the homework for basic operations on STLs. According to the code, a **Point** is defined as a tuple of three floats and a **Triangle** is defined as a tuple of three points. A **Shape** is defined as a list of Triangles. **createObjectModelString** function creates the STL structure for the given Shape. **writeObjModel** function writes the object as an STL file.

```
type Point = (Float,Float, Float)
type Triangle = (Point, Point, Point)
type Shape = [Triangle]

t1 :: Triangle
t1 = ((0,0,10),(0,-5,0),(0,5,0))

t2 :: Triangle
t2 = ((0,0,20),(0,-5,10),(0,5,10))

t3 :: Triangle
t3 = ((0,0,30),(0,-5,20),(0,5,20))


createTriangleDef :: Triangle -> String
createTriangleDef ((x1,y1,z1),(x2,y2,z2),(x3,y3,z3)) = "  facet\n" ++
                                                        "    outer loop\n"
    ++
                                                        "      vertex " ++ (
    show x1) ++ " " ++ (show y1) ++ " " ++ (show z1) ++ "\n" ++
                                                        "      vertex " ++ (
    show x2) ++ " " ++ (show y2) ++ " " ++ (show z2) ++ "\n" ++
                                                        "      vertex " ++ (
    show x3) ++ " " ++ (show y3) ++ " " ++ (show z3) ++ "\n" ++
                                                        "    endloop\n" ++
                                                        "  endfacet\n"


createObjectModelString :: Shape -> String
createObjectModelString n = "solid Object01\n" ++ concat [createTriangleDef
     y | y<-n] ++ "endsolid Object01"

writeObjModel :: Shape -> String -> IO ()
writeObjModel x filename = do writeFile filename (createObjectModelString x
    )
```

Using the given functions, the line **writeObjModel [t1,t2,t3] "test.stl"** will create the 3D object. The object could be visualized using CloudCompare[1]. The result is given

[1] https://www.danielgm.net/cc/

in the Figure 1.

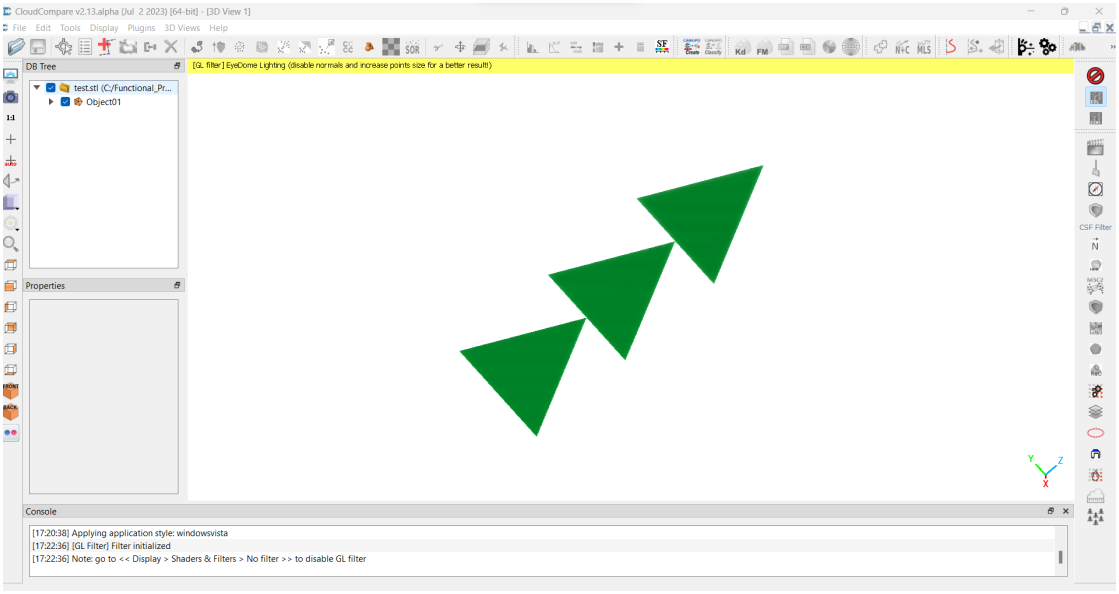

Figure 1: An STL object shown in CloudCompare.

2 (30 pts): Part 1 - Sierpiński's triangle

Write a "$Int- > Shape$" function which generates Sierpiński's triangle fractal starting from the triangle t1. The function's only parameter is the iteration count.

3 (30 pts): Part 2 - Koch's snowflake

Write a "$Int- > Shape$" function which generates Koch's snowflake fractal starting from the triangle t1. The function's only parameter is the iteration count.

4 (20 pts): Part 3 - Cube at a specified position

Write a "$Point- > Float- > Shape$" function which creates a cube centered at the given position and side length. Remember that, a cube could be created using 12 triangles.

5 (20 pts): Part 4 - Cube Pattern

Write a "$Int- > Shape$" function which generates the following fractal:

- Start with a single cube.
- For every iteration, place a smaller cube to the center of each face of every cube. The edge length of these new cubes should be half of the cubes from the previous iteration.

The first two iterations are given in Figure 2.

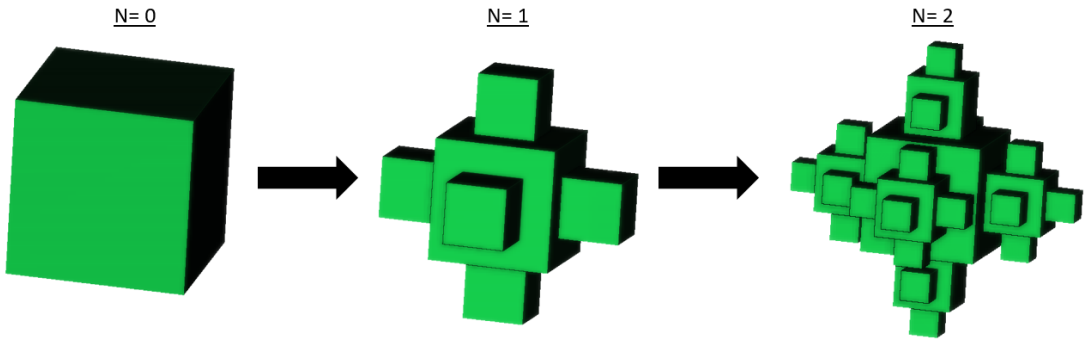

Figure 2: 3D Cube pattern for Part 4.

Hint: You do not need to consider the "inner" cubes.