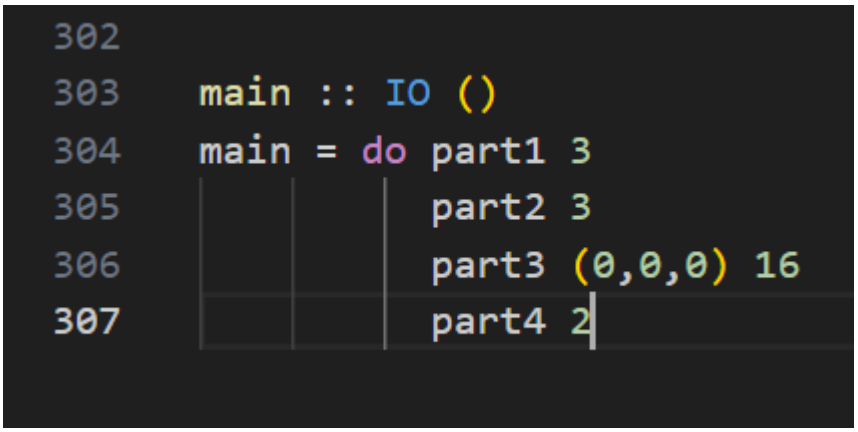
BLG458E Homework 1 Report

How to run the functions?

There are 2 versions of the homework. The only difference is hw1-v1.hs includes this main function:



and hw1-v2.hs doesn't include that. So, we can compile hw1-v1.hs and run the hw1-v1.exe . I compiled and ran it, and as a result, wanted stl files were generated.

If wanted, in cmd, from the interactive environment ghci we can load hw1-v2.hs and write these functions ourselves and generate wanted stl files.

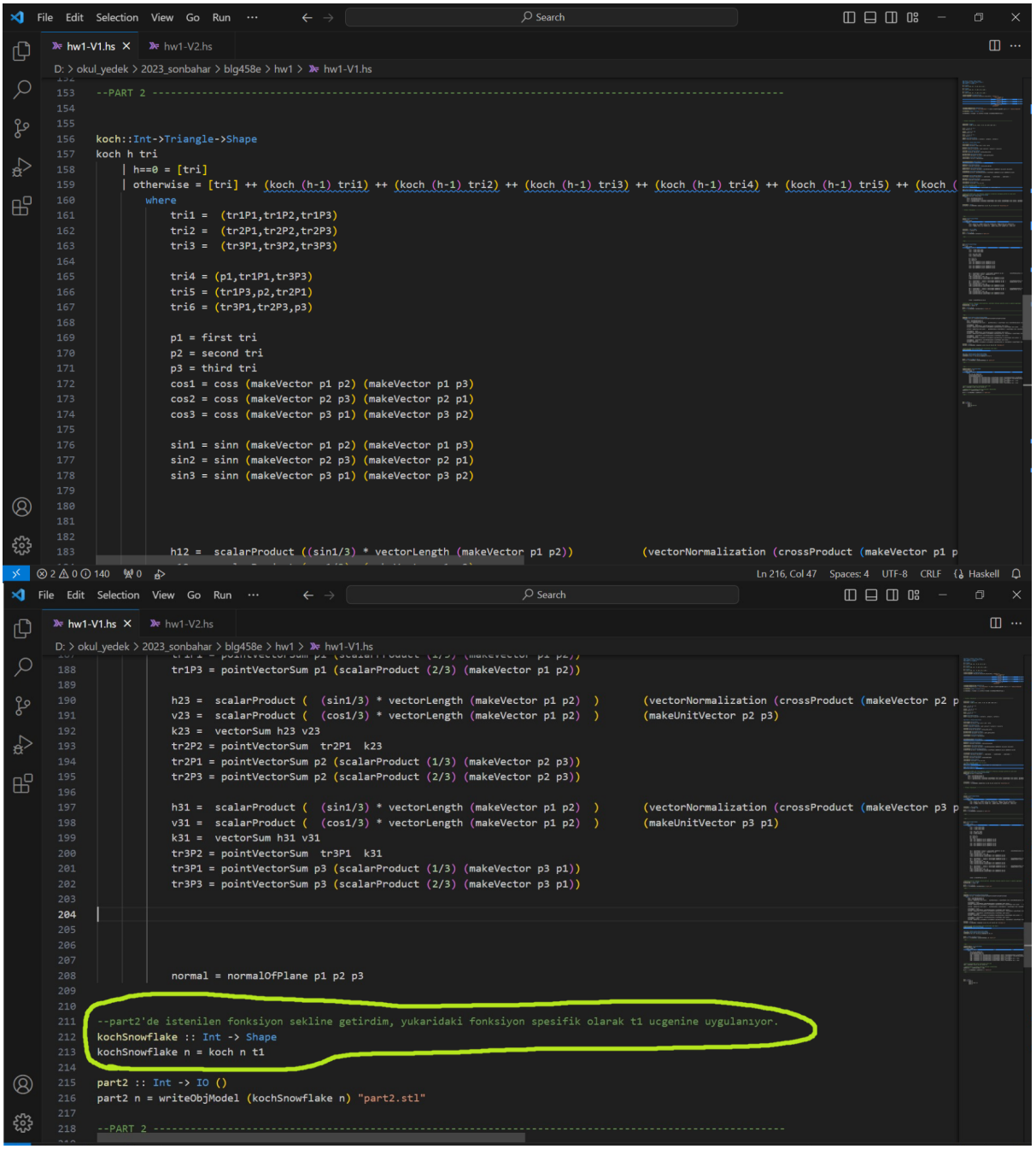
part1, part2, part3 and part4 functions are the stl file generator functions of the respective parts in the homework. The values given in the main function are for example purposes. You can use these functions yourself in the interactive environment ghci by loading hw1-v2.hs . After compiling hw1-v1.hs, the executable file is going to run the functions in the picture.

Part 1 - Sierpinski’s Triangle

```
129
130  --PART 1 ------------------------------------------------------------------------------------------------------
131
132  sierpin::Int->Triangle->Shape
133  sierpin h tri
134      | h==0 = [tri]
135      | otherwise = (sierpin (h-1) tri1) ++ (sierpin (h-1) tri2) ++ (sierpin (h-1) tri3)
136          where
137              tri1 = (first tri , middle (first tri) (second tri), middle (first tri) (third tri))
138              tri2 = (middle (first tri) (second tri) , second tri , middle (second tri) (third tri))
139              tri3 = ( middle (first tri) (third tri) , middle (third tri) (second tri) , third tri)
140
141
142  sierpinski :: Int -> Shape
143  sierpinski n = sierpin n t1
144
145  part1 :: Int -> IO ()
146  part1 n = writeObjModel (sierpinski n) "part1.stl"
147
148
149  --PART 1 ------------------------------------------------------------------------------------------------------
```

The marked function is the wanted function of part 1 of the homework. While I was doing part 1, I have not used the power of linear algebra so my code here is pretty primitive, just a recursion and list concatenation.

Part 2 - Koch’s Snowflake



```haskell
--PART 2 ------------------------------------------------------------------------------------------

koch::Int->Triangle->Shape
koch h tri
    | h==0 = [tri]
    | otherwise = [tri] ++ (koch (h-1) tri1) ++ (koch (h-1) tri2) ++ (koch (h-1) tri3) ++ (koch (h-1) tri4) ++ (koch (h-1) tri5) ++ (koch (
        where
            tri1 =  (tr1P1,tr1P2,tr1P3)
            tri2 =  (tr2P1,tr2P2,tr2P3)
            tri3 =  (tr3P1,tr3P2,tr3P3)

            tri4 = (p1,tr1P1,tr3P3)
            tri5 = (tr1P3,p2,tr2P1)
            tri6 = (tr3P1,tr2P3,p3)

            p1 = first tri
            p2 = second tri
            p3 = third tri
            cos1 = coss (makeVector p1 p2) (makeVector p1 p3)
            cos2 = coss (makeVector p2 p3) (makeVector p2 p1)
            cos3 = coss (makeVector p3 p1) (makeVector p3 p2)

            sin1 = sinn (makeVector p1 p2) (makeVector p1 p3)
            sin2 = sinn (makeVector p2 p3) (makeVector p2 p1)
            sin3 = sinn (makeVector p3 p1) (makeVector p3 p2)



            h12 =  scalarProduct ((sin1/3) * vectorLength (makeVector p1 p2))          (vectorNormalization (crossProduct (makeVector p1 p
```

```haskell
            tr1P3 = pointVectorSum p1 (scalarProduct (2/3) (makeVector p1 p2))

            h23 =  scalarProduct (  (sin1/3) * vectorLength (makeVector p1 p2)  )       (vectorNormalization (crossProduct (makeVector p2 p
            v23 =  scalarProduct (  (cos1/3) * vectorLength (makeVector p1 p2)  )       (makeUnitVector p2 p3)
            k23 =  vectorSum h23 v23
            tr2P2 = pointVectorSum  tr2P1  k23
            tr2P1 = pointVectorSum p2 (scalarProduct (1/3) (makeVector p2 p3))
            tr2P3 = pointVectorSum p2 (scalarProduct (2/3) (makeVector p2 p3))

            h31 =  scalarProduct (  (sin1/3) * vectorLength (makeVector p1 p2)  )       (vectorNormalization (crossProduct (makeVector p3 p
            v31 =  scalarProduct (  (cos1/3) * vectorLength (makeVector p1 p2)  )       (makeUnitVector p3 p1)
            k31 =  vectorSum h31 v31
            tr3P2 = pointVectorSum  tr3P1  k31
            tr3P1 = pointVectorSum p3 (scalarProduct (1/3) (makeVector p3 p1))
            tr3P3 = pointVectorSum p3 (scalarProduct (2/3) (makeVector p3 p1))



            normal = normalOfPlane p1 p2 p3

--part2'de istenilen fonksiyon sekline getirdim, yukaridaki fonksiyon spesifik olarak t1 ucgenine uygulanıyor.
kochSnowflake :: Int -> Shape
kochSnowflake n = koch n t1

part2 :: Int -> IO ()
part2 n = writeObjModel (kochSnowflake n) "part2.stl"

--PART 2 ------------------------------------------------------------------------------------------
```

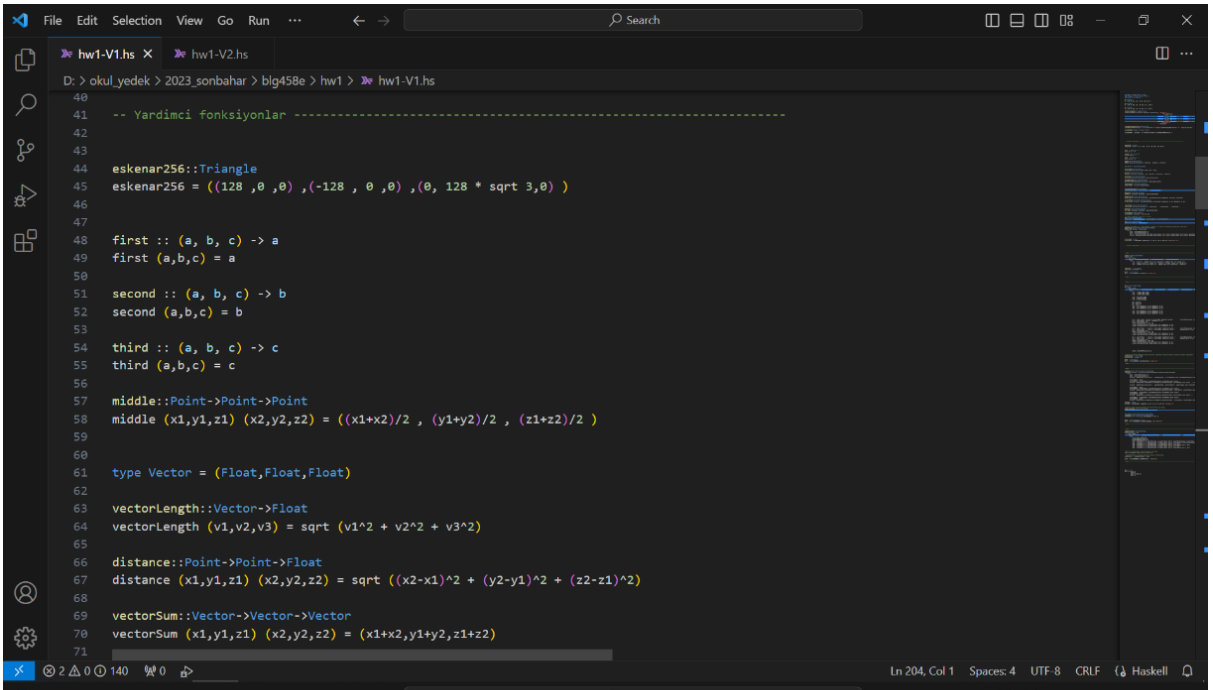
From the moment I understood the weakness of my flesh, it disgusted me. I craved the strength and certainty of Linear Algebra. I aspired to the purity of the Blessed Vector.

In this part, I understood I need the power of linear algebra so I wrote some helper code to implement it. By the way, the marked function is the wanted function of part 2 of the homework. With this code we can use any triangle for Koch's Snowflake anywhere in 3D space.

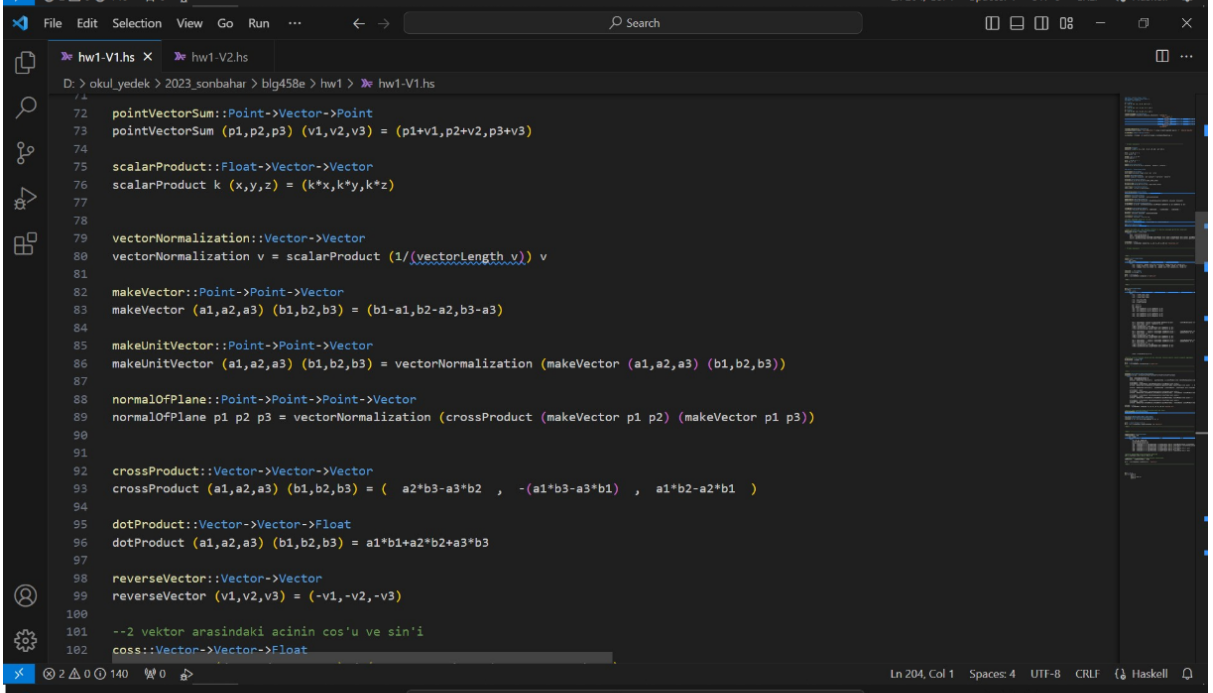
To generate the new triangles, first we need to identify their points so we can draw them on the space. 2 points for the base is easy. Finding the third sharp point is the hard part. For this first we find the normal of the

plane of the triangle. After that, we take the cross product of the normal of the triangle with a vector that was generated from the triangle's two vertices. By doing this, we find the direction of the vector which is orthogonal to the edge of the triangle to reach the third point we seek. After some vector summation we reach our goal.

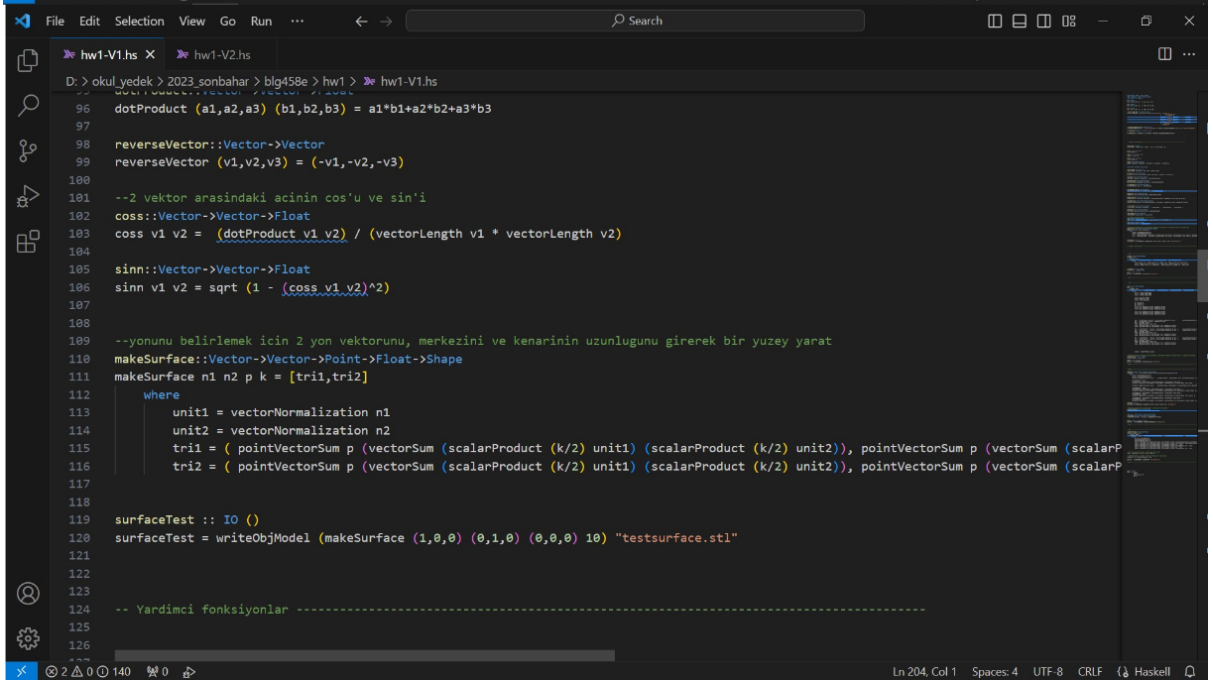
Below are the codes for linear algebra, making a square surface and a couple of more mathematical things.



```
-- Yardimci fonksiyonlar ---------------------------------------------------------------------


eskenar256::Triangle
eskenar256 = ((128 ,0 ,0) ,(-128 , 0 ,0) ,(0, 128 * sqrt 3,0) )


first :: (a, b, c) -> a
first (a,b,c) = a

second :: (a, b, c) -> b
second (a,b,c) = b

third :: (a, b, c) -> c
third (a,b,c) = c

middle::Point->Point->Point
middle (x1,y1,z1) (x2,y2,z2) = ((x1+x2)/2 , (y1+y2)/2 , (z1+z2)/2 )


type Vector = (Float,Float,Float)

vectorLength::Vector->Float
vectorLength (v1,v2,v3) = sqrt (v1^2 + v2^2 + v3^2)

distance::Point->Point->Float
distance (x1,y1,z1) (x2,y2,z2) = sqrt ((x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2)

vectorSum::Vector->Vector->Vector
vectorSum (x1,y1,z1) (x2,y2,z2) = (x1+x2,y1+y2,z1+z2)
```

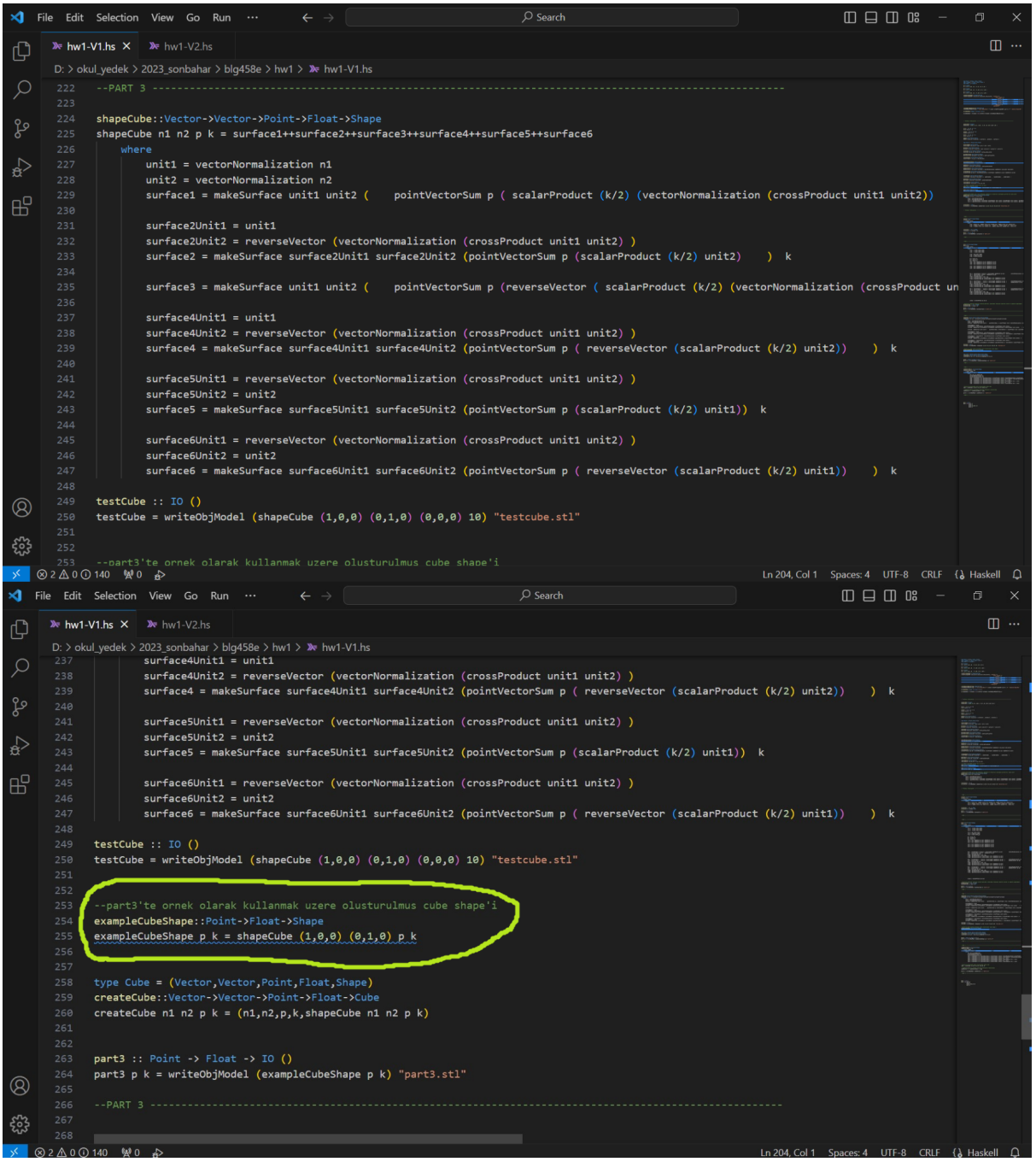


```
pointVectorSum::Point->Vector->Point
pointVectorSum (p1,p2,p3) (v1,v2,v3) = (p1+v1,p2+v2,p3+v3)

scalarProduct::Float->Vector->Vector
scalarProduct k (x,y,z) = (k*x,k*y,k*z)


vectorNormalization::Vector->Vector
vectorNormalization v = scalarProduct (1/(vectorLength v)) v

makeVector::Point->Point->Vector
makeVector (a1,a2,a3) (b1,b2,b3) = (b1-a1,b2-a2,b3-a3)

makeUnitVector::Point->Point->Vector
makeUnitVector (a1,a2,a3) (b1,b2,b3) = vectorNormalization (makeVector (a1,a2,a3) (b1,b2,b3))

normalOfPlane::Point->Point->Point->Vector
normalOfPlane p1 p2 p3 = vectorNormalization (crossProduct (makeVector p1 p2) (makeVector p1 p3))


crossProduct::Vector->Vector->Vector
crossProduct (a1,a2,a3) (b1,b2,b3) = (  a2*b3-a3*b2  ,  -(a1*b3-a3*b1)  ,  a1*b2-a2*b1  )

dotProduct::Vector->Vector->Float
dotProduct (a1,a2,a3) (b1,b2,b3) = a1*b1+a2*b2+a3*b3

reverseVector::Vector->Vector
reverseVector (v1,v2,v3) = (-v1,-v2,-v3)

--2 vektor arasindaki acinin cos'u ve sin'i
coss::Vector->Vector->Float
```



```
dotProduct (a1,a2,a3) (b1,b2,b3) = a1*b1+a2*b2+a3*b3

reverseVector::Vector->Vector
reverseVector (v1,v2,v3) = (-v1,-v2,-v3)

--2 vektor arasindaki acinin cos'u ve sin'i
coss::Vector->Vector->Float
coss v1 v2 =  (dotProduct v1 v2) / (vectorLength v1 * vectorLength v2)

sinn::Vector->Vector->Float
sinn v1 v2 = sqrt (1 - (coss v1 v2)^2)


--yonunu belirlemek icin 2 yon vektorunu, merkezini ve kenarinin uzunlugunu girerek bir yuzey yarat
makeSurface::Vector->Vector->Point->Float->Shape
makeSurface n1 n2 p k = [tri1,tri2]
    where
        unit1 = vectorNormalization n1
        unit2 = vectorNormalization n2
        tri1 = ( pointVectorSum p (vectorSum (scalarProduct (k/2) unit1) (scalarProduct (k/2) unit2)), pointVectorSum p (vectorSum (scalarP
        tri2 = ( pointVectorSum p (vectorSum (scalarProduct (k/2) unit1) (scalarProduct (k/2) unit2)), pointVectorSum p (vectorSum (scalarP


surfaceTest :: IO ()
surfaceTest = writeObjModel (makeSurface (1,0,0) (0,1,0) (0,0,0) 10) "testsurface.stl"



-- Yardimci fonksiyonlar ---------------------------------------------------------------------------
```

Part 3 - Cube at a specified position

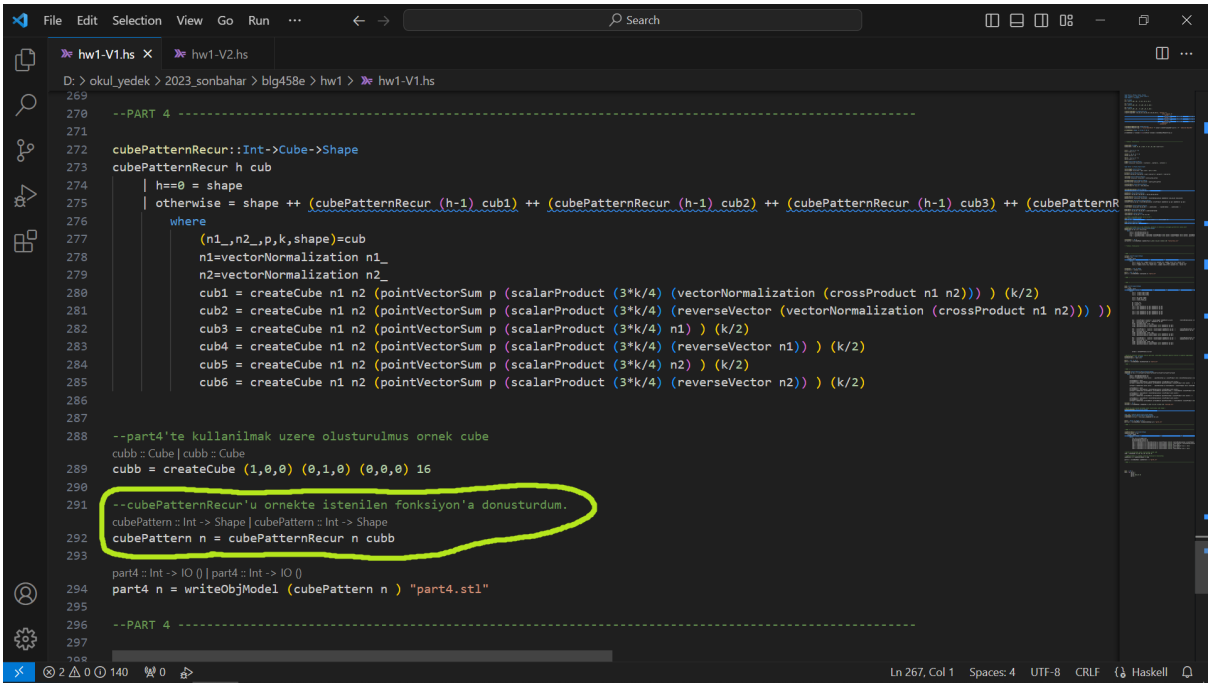


```haskell
--PART 3 ----------

shapeCube::Vector->Vector->Point->Float->Shape
shapeCube n1 n2 p k = surface1++surface2++surface3++surface4++surface5++surface6
    where
        unit1 = vectorNormalization n1
        unit2 = vectorNormalization n2
        surface1 = makeSurface unit1 unit2 (    pointVectorSum p ( scalarProduct (k/2) (vectorNormalization (crossProduct unit1 unit2))

        surface2Unit1 = unit1
        surface2Unit2 = reverseVector (vectorNormalization (crossProduct unit1 unit2) )
        surface2 = makeSurface surface2Unit1 surface2Unit2 (pointVectorSum p (scalarProduct (k/2) unit2)    ) k

        surface3 = makeSurface unit1 unit2 (    pointVectorSum p (reverseVector ( scalarProduct (k/2) (vectorNormalization (crossProduct un

        surface4Unit1 = unit1
        surface4Unit2 = reverseVector (vectorNormalization (crossProduct unit1 unit2) )
        surface4 = makeSurface surface4Unit1 surface4Unit2 (pointVectorSum p ( reverseVector (scalarProduct (k/2) unit2))    ) k

        surface5Unit1 = reverseVector (vectorNormalization (crossProduct unit1 unit2) )
        surface5Unit2 = unit2
        surface5 = makeSurface surface5Unit1 surface5Unit2 (pointVectorSum p (scalarProduct (k/2) unit1))  k

        surface6Unit1 = reverseVector (vectorNormalization (crossProduct unit1 unit2) )
        surface6Unit2 = unit2
        surface6 = makeSurface surface6Unit1 surface6Unit2 (pointVectorSum p ( reverseVector (scalarProduct (k/2) unit1))    ) k

testCube :: IO ()
testCube = writeObjModel (shapeCube (1,0,0) (0,1,0) (0,0,0) 10) "testcube.stl"


--part3'te ornek olarak kullanmak uzere olusturulmus cube shape'i
exampleCubeShape::Point->Float->Shape
exampleCubeShape p k = shapeCube (1,0,0) (0,1,0) p k

type Cube = (Vector,Vector,Point,Float,Shape)
createCube::Vector->Vector->Point->Float->Cube
createCube n1 n2 p k = (n1,n2,p,k,shapeCube n1 n2 p k)


part3 :: Point -> Float -> IO ()
part3 p k = writeObjModel (exampleCubeShape p k) "part3.stl"

--PART 3 ----------
```

The marked function here is the wanted function of part 3 of the homework. To create a cube we need to specify 2 vectors to describe its orientation and 1 point to describe its place in space.(1,0,0) and (0,1,0) are the vectors of the example cube.

shapeCube function generates the shape of the cube, createCube specifies every aspect of the cube for the 3D space. By the way, I created a new type just for the cube because its shape alone is not enough to describe it.

Part 4 - Cube Pattern

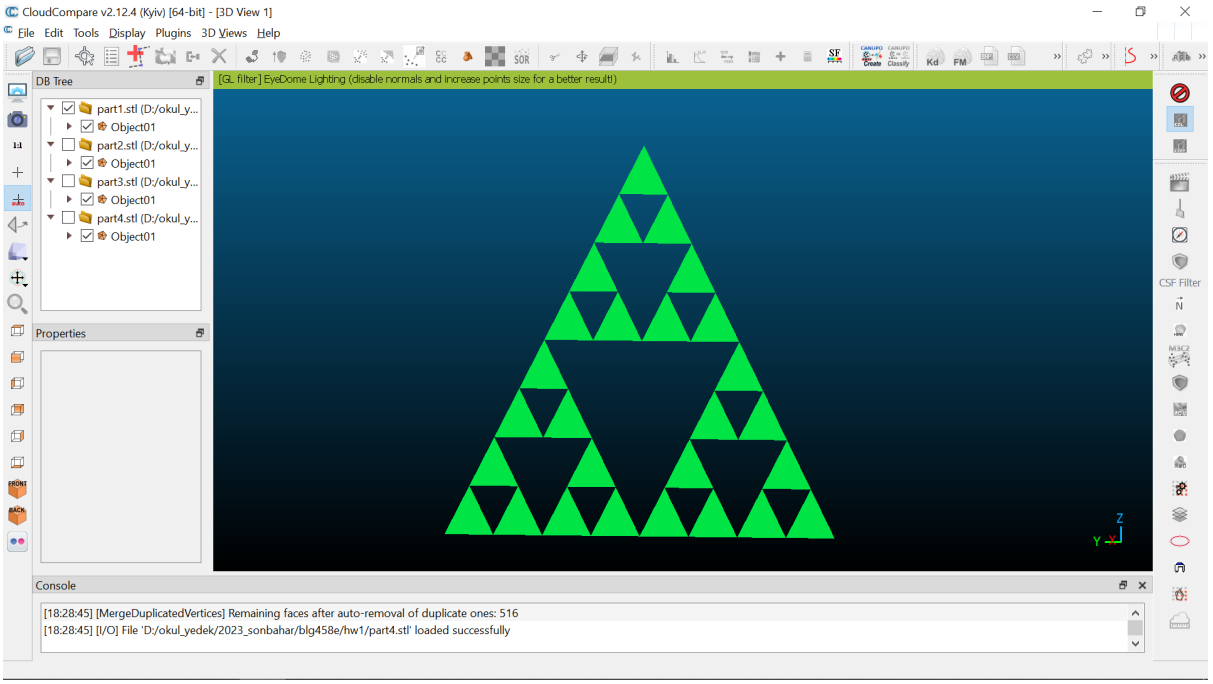


Like before, the marked function here is the wanted function of part 4 of the homework. Like said in the code's comments, cubb here is just a standard cube I defined so that we could generate a pattern on the base of it.

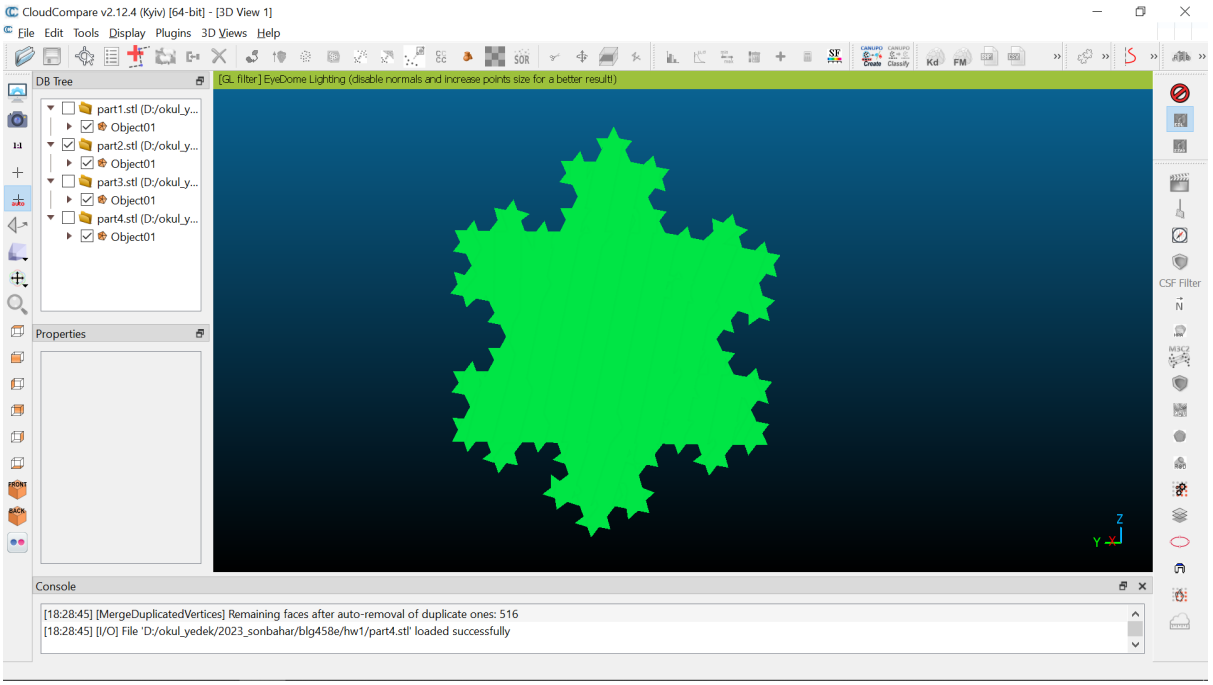
In all parts of the homework, recursion functions can take any triangle or any kind of cube and generate their pattern with them. To generate concrete examples, I defined some default cubes. For the triangles, like asked in the homework, I used triangle t1.

Below are the resulting stl files showed in CloudCompare:

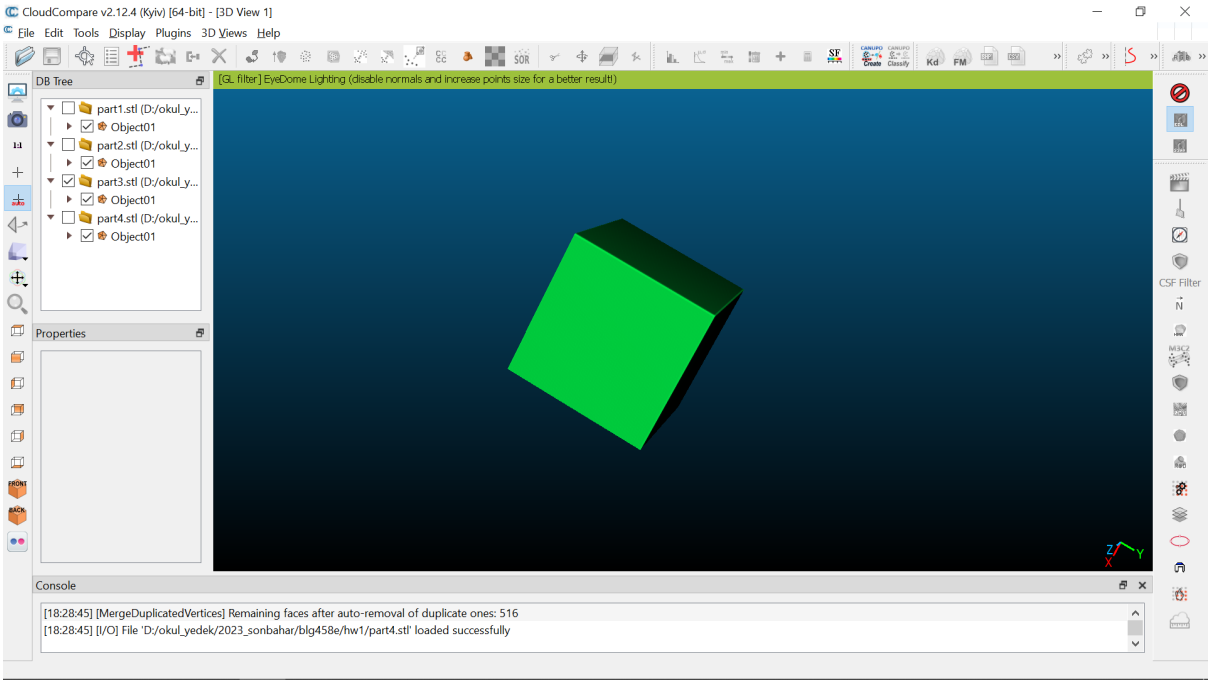
Part 1


CloudCompare v2.12.4 (Kyiv) [64-bit] - [3D View 1]
File Edit Tools Display Plugins 3D Views Help
DB Tree
part1.stl (D:/okul_y...
Object01
part2.stl (D:/okul_y...
Object01
part3.stl (D:/okul_y...
Object01
part4.stl (D:/okul_y...
Object01
Properties
[GL filter] EyeDome Lighting (disable normals and increase points size for a better result!)
CSF Filter
Console
[18:28:45] [MergeDuplicatedVertices] Remaining faces after auto-removal of duplicate ones: 516
[18:28:45] [I/O] File 'D:/okul_yedek/2023_sonbahar/blg458e/hw1/part4.stl' loaded successfully

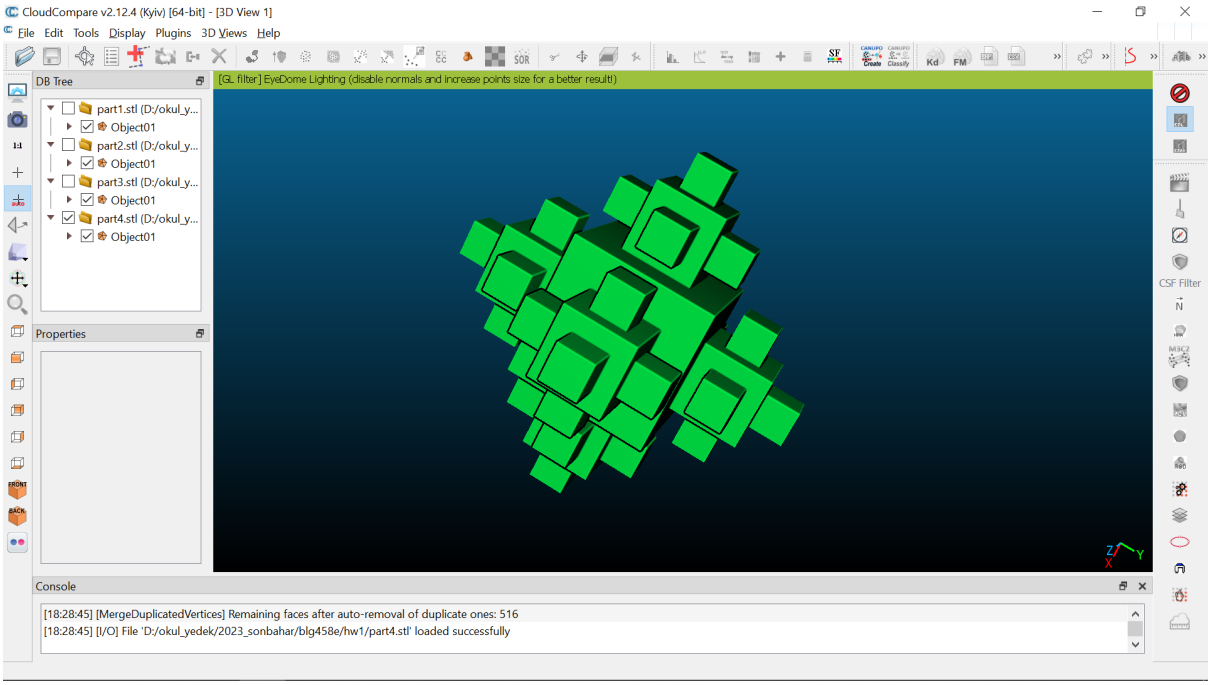
Part 2


CloudCompare v2.12.4 (Kyiv) [64-bit] - [3D View 1]
File Edit Tools Display Plugins 3D Views Help
DB Tree
part1.stl (D:/okul_y...
Object01
part2.stl (D:/okul_y...
Object01
part3.stl (D:/okul_y...
Object01
part4.stl (D:/okul_y...
Object01
Properties
[GL filter] EyeDome Lighting (disable normals and increase points size for a better result!)
CSF Filter
Console
[18:28:45] [MergeDuplicatedVertices] Remaining faces after auto-removal of duplicate ones: 516
[18:28:45] [I/O] File 'D:/okul_yedek/2023_sonbahar/blg458e/hw1/part4.stl' loaded successfully

Part 3


CloudCompare v2.12.4 (Kyiv) [64-bit] - [3D View 1]
File Edit Tools Display Plugins 3D Views Help
DB Tree
part1.stl (D:/okul_y...
Object01
part2.stl (D:/okul_y...
Object01
part3.stl (D:/okul_y...
Object01
part4.stl (D:/okul_y...
Object01
Properties
[GL filter] EyeDome Lighting (disable normals and increase points size for a better result!)
CSF Filter
Console
[18:28:45] [MergeDuplicatedVertices] Remaining faces after auto-removal of duplicate ones: 516
[18:28:45] [I/O] File 'D:/okul_yedek/2023_sonbahar/blg458e/hw1/part4.stl' loaded successfully

Part 4


CloudCompare v2.12.4 (Kyiv) [64-bit] - [3D View 1]
File Edit Tools Display Plugins 3D Views Help
DB Tree
part1.stl (D:/okul_y...
Object01
part2.stl (D:/okul_y...
Object01
part3.stl (D:/okul_y...
Object01
part4.stl (D:/okul_y...
Object01
Properties
[GL filter] EyeDome Lighting (disable normals and increase points size for a better result!)
CSF Filter
Console
[18:28:45] [MergeDuplicatedVertices] Remaining faces after auto-removal of duplicate ones: 516
[18:28:45] [I/O] File 'D:/okul_yedek/2023_sonbahar/blg458e/hw1/part4.stl' loaded successfully