שלב 1

עשינו פונקציה בcamera בשם constructMultipleRaysThroughPixel

שמחזירה רשימה של 100 קרניים. מכפילים 2 מספרים רנדומליים בטווח של הגובה והרוחב ב2 הווקטורים וכך הם זזים

כל פעם בתוך הפיקסל עצמו .הפונקציה ממש דומה לconstructRaythroughPixel

רק שהיא מחזירה רשימה של קרניים שזזו בתוך הפיקסל.

*/\*constructs 100 rays throgh same pixel each time moving a bit within the pixel\*/* **public** List<Ray> constructMultipleRaysThroughPixel(**int** nX, **int** nY, **int** j, **int** i, **double** screenDistance,  
 **double** screenWidth, **double** screenHeight)  
 {  
 **if** (*isZero*(screenDistance))*//screen distance can't be zero* {  
 **throw new** IllegalArgumentException(**"distance cannot be 0"**);  
 }  
  
 List<Ray> rays = **new** ArrayList<>();*//creating a new list of rays* Point3D Pc = **new** Point3D( **\_p0**.add(**\_vTo**.scale(screenDistance)));  
 **double** Ry = screenHeight/nY; *//height of the pixel* **double** Rx = screenWidth/nX;*//Width of the pixel* **double** yi = ((i - nY/2d)\*Ry); *// yi distance of original pixel from (0,0) on Y axis***double** xj= ((j - nX/2d)\*Rx);*//xj distance of original pixel from (0,0) on X axis*

**int** multipleRays=100;*//num of rays* **for** (**int** count = 0; count < multipleRays; ++count) {  
  
 **double** minY =-Ry;*//minimum height of the pixel* **double** maxY=Ry;*//maximum height of the pixel* **double** minX=-Rx;*//minimum width of the pixel* **double** maxX=Rx;*//maximum width of the pixel* **double** randomY = ***rnd***.nextDouble() ;*//generates random number between 0-1* **double** randomX= ***rnd***.nextDouble();*//generates random number between 0-1* **double** y=minY+(randomY\*(maxY-minY));*//y will be number between -Ry and Ry* **double** x=minX+(randomX\*(maxX-minX));*//x will be number between -Rx and Rx* Point3D Pij = Pc;  
  
 **if** (!Util.*isZero*(x + xj))*//if x+xj doesnt equel zero do the next line* {  
 Pij = Pij.add(**\_vRight**.scale(x + xj));*//will move the ray along the width of the pixel.* }  
 **if** (!Util.*isZero*(y + yi))*//if y+yi doesnt equel zero do the next line* {  
 Pij = Pij.add(**\_vUp**.scale(-y -yi ));*//will move the ray along the hight of the pixel.* }  
 Vector Vij = Pij.subtract(**\_p0**);  
 Ray r=**new** Ray(**\_p0**,Vij);  
  
 rays.add(r);*//adds new ray to list* }  
  
 **return** rays;*//returns full list of rays*}  
  
  
}

ובrender הוספנו משתנה בשם \_superSampleingDensity ששווה לאפס. בrenderimage כשרוצים שזה לא יעשה supersampeling צריך שהמשתנה לא יהיה שווה אפס ואם רוצים שזה כן יעשה צריך שהוא יהיה שווה 0.

ברגע שהמשתנה שווה 0 הrenderImage יעביר אותנו לפונקצייה בשם renderingSupersamplingRays ש מקבל רשימה של קרניים. אם הfindintersection שווה לnull הפיקסל יהיה הצבע רקע ואם לא זה יעשה ממוצע של הקרניים ויחזיר את הצבע של הפיקסל.

*/\*\*this function does the sumpersameling it gets a list of rays then caculates the ray color if the find intersection= null then its the backround color  
 \* and if not it averages out the rays and returns the color of the pixel  
 \** ***@param background*** *color of backround  
 \** ***@param camera*** *camera  
 \** ***@param distance*** *distance  
 \** ***@param height*** *View Plane height in size units  
 \** ***@param nx*** *amount of pixels by Width  
 \** ***@param ny*** *amount of pixels by height  
 \** ***@param width*** *View Plane width in size units\*/* **private void** renderingSupersamplingRays(Camera camera, java.awt.Color background, **double** distance, **double** width, **double** height, **int** nx, **int** ny) {  
 **for** (**int** row = 0; row < ny; ++row) {  
 **for** (**int** column = 0; column < nx; ++column) {  
 List<Ray> rays = camera.constructMultipleRaysThroughPixel(nx, ny, column, row, distance, width, height);*//calling the function constructMultipleRaysThroughPixel and sending the row and collum which are the i and j* Color color= Color.***BLACK***;  
 **double** counter =0;  
 **for** (Ray ray : rays){  
 GeoPoint intersectionPoint=findClosestIntersection(ray);  
 **if** (intersectionPoint == **null**){  
 color = color.add(**\_scene**.getBackground());*//if findClosestIntersection=null it means there is no shape so the color is the backround color* }  
 **else**{  
 counter++;  
 color = color.add(calcColor(intersectionPoint, ray));*//caculates the color of the ray* }  
 }  
 Color color1;  
 **if** (counter==0)  
 color1=**new** Color(background);*//if counter=0 it means there are no shapes so its the backround color* **else** {  
 color1=color;  
 color1=color1.scale(1d/rays.size());*//gets the average of the rays color* }  
 **\_imageWriter**.writePixel(column, row, color1.getColor());*//gives caculated color to pixel* }  
 }  
 }