דו"ח פרויקט "מיני פרויקט במבוא להנדסת תוכנה"

פירוט לגבי הפונקציות:

במחלקות הבסיסיות יצרנו פונקציות שיעזרו לנו בפעולות חיבור, חיסור, כפל וחילוק.

יצרנו פעולות של setters רק כשהיה מותר מבחינת ההוראות.

רוב הפונקציות מחזירות משתנים חדשים, חוץ מפעולת הנרמול normalize שמנרמלת את הווקטור שנשלח לפונקציה ומחזירה אותו בעצמו.

השתמשנו בשמות משמעותיים לצורך הבנת המשתנים, שמות שדומים לכתוב במצגות המעבדה.

ישנו תיעוד לרוב הפונקציות ולמחלקות

מיני פרויקט 1-שיפור תמונה

למדנו על שיפורים שונים במהלך השיעורים, ביניהם:

הבעיה:

קצוות חדים- קצוות הגופים בתמונה אינם חלקים. התמונות נראות מאד מפוקסלות. בעיה זו נוצרת כתוצאה מכך שקבענו את הצבע בפיקסל להיות הצבע שמתקבל מהקרן יחידה שנשלחת מהמצלמה דרך מרכז הפיקסל לאובייקטים בסצנה. בקצוות הגופים ישנו מצב שחלק מהפיקסל מכסה גוף מסוים וחלק ממנו לא.

ולכן הפתרון הוא:

Super- sampling :

בתחילה ירינו רק קרן אחת דרך מרכז הפיקסל, ואז בקצוות הצורות היו נראים חיתוכי משולשים קטנים בגלל הפרשי הצבע של הצורה והרקע, לכן השיפור שנעשה הוא: במקום לשלוח דרך כל פיקסל בודד קרן אחת בלבד, נשלח הרבה קרניים דרך חלקים שונים בפיקסל כאשר אצלנו הגדרנו רשת שדרכה יעבור קרניים בצורה מסודרתמכל תת פיקסל בודד. לאחר מכן נחשב את הממוצע המשוקלל של הצבעים שהתקבלו מכל הקרניים שנוצרו לתוך צבע אחד ואיתו נצבע את הפיקסל. יש חשיבות גם לאזור שבו דוגמים.אנחנו נדגום מכל תת רשת בצורה רנדומלית.

המימוש בפועל:

במחלקת camera יש שדה בוליאני שמציין האם בעצם יש לירות אלומת קרניים במקום קרן בודדת-

כלומר האם אנו מעוניינים בשיפור.

בפונק RenderImage נשאל אם מעוניינים בשיפור ואם כן נשלח לפונק במחלקת camera שתפקידה ליצור רשימת קרניים

```
//AntiAliacing only
else if (camera.isAntialiacing()&&!camera.isDepthOfField()) {//anti true

    List<Ray> rays = camera.constructRaysThroughPixel(nX, nY, j, i);
    Color pixelcolor=AddColor(rays);
    pixelcolor = pixelcolor.scale(1d / rays.size());
    imageWriter.writePixel(j, i, pixelcolor);
}
```

הפונק' במחלקת camera:

נחלק כל פיקסל לרשת פיקסלים שגודלה יהיה בגודל מספר הקרנים כלומר האורך והרוחב שלהם הם שורש של מספר הקרניים הנשלחות ביצירת תמונה. נרצה לירות קרניים מכל תת פיקסל ברשת כאשר מיקום הנקודה בה עוברת הקרן בתת פיקסל תיהיה רנודמלית.

חישוב הנקודה הרנדומלית יחושב על פי נקודת המרכז של כל תת פיקסל כאשר נוסיף לא ערך רנודמלי בגבולות התת פיקסל הספציפי.

בפונקציה יש שימוש בפונקציה getPij שמחזירה נקודת מרכז פיקסל.

```java
//random with grid
public List<Ray> constructRaysThroughPixel(int nX, int nY, int j, int i) {
    Ray centerRay = constructRayThroughPixel(nX,nY,j,i);

    double Rx = _width / nX;//the length of pixel in X axis
    double Ry = _height / nY;//the length of pixel in Y axis

    Point3D Pij = getPij(nX, nY, j, i, _width, _height, _distance);
    Point3D tmp;
    //-----SuperSampling-----
    List<Ray> rays = new LinkedList<>();//the return list, construct Rays Through Pixels


    double n = Math.floor(Math.sqrt(sizeGrid));
    int delta = (int) (n / 2d);//the movement from the center of the pixel
    //length and width of each sub pixel
    double gapX = Rx / n;
    double gapY = Ry / n;

*************************************************************************
          |(-3,-3)|(-3,-2)|(-3,-1)|(-3, 0)|(-3, 1)|(-3, 2)||(-3, 3)
          |(-2,-3)|(-2,-2)|(-2,-1)|(-2, 0)|(-2, 1)|(-2, 2)||(-2, 3)
          |(-1,-3)|(-1,-2)|(-1,-1)|(-1, 0)|(-1, 1)|(-1, 2)||(-1, 3)
          |( 0,-3)|( 0,-2)|( 0,-1)|( 0, 0)|( 0, 1)|( 0, 2)||( 0, 3)
          |( 1,-3)|( 1,-2)|( 1,-1)|( 1, 0)|( 1, 1)|( 1, 2)||( 1, 3)
          |( 2,-3)|( 2,-2)|( 2,-1)|( 2, 0)|( 2, 1)|( 2, 2)||( 2, 3)
          |( 3,-3)|( 3,-2)|( 3,-1)|( 3, 0)|( 3, 1)|( 3, 2)||( 3, 3)
************************************************************************* */
```

חישוב נקודת מעבר הקרן עבור כל תת פיקסל

```java
for (int row = -delta; row <= delta; row++) {
    for (int col = -delta; col <= delta; col++) {
        tmp = new Point3D(Pij.getX(),Pij.getY(),Pij.getZ());//מרכז הפיקסל הספציפי
        if (!isZero(row)) {//אם אני לא בפיקסל המרכזי
            tmp = tmp.add(_vRIGHT.scale(row * (double) Math.random()*((gapX))-(gapX)/2));//(double) Math.random()*((gapX))-gapX/2))
            //tmp = tmp.add(_vRIGHT.scale(row * gapX));
        }
        if (!isZero(col)) {
            tmp = tmp.add(_vRIGHT.scale(col * (double) Math.random()*((gapY))-(gapY)/2));//(double) Math.random()*((gapY))-gapY/2));
            //tmp = tmp.add(_vRIGHT.scale(col * gapY));
        }
        rays.add(new Ray(_p0, tmp.subtract(_p0).normalize()));
    }
}
return rays;
```

הפונקציה תחזיר לrenderImage רשימת קרניים.

כעת נחשב את צבע הפיקסל-ע"י חישוב צבע ממוצע.

נשלח לפונק שתוסיף לצבע המקורי את הצבעים שהגיעו משאר הקרניים :

```java
Color pixelcolor=AddColor(rays);
```

הפונקציה AddColor:

```
public Color AddColor(List<Ray> rays){
    Color pixelcolor = Color.BLACK;

    for (var r : rays) {
        Color rcolor = tracer.traceRay(r);
        pixelcolor = pixelcolor.add(rcolor);
    }
    return pixelcolor;
}
```

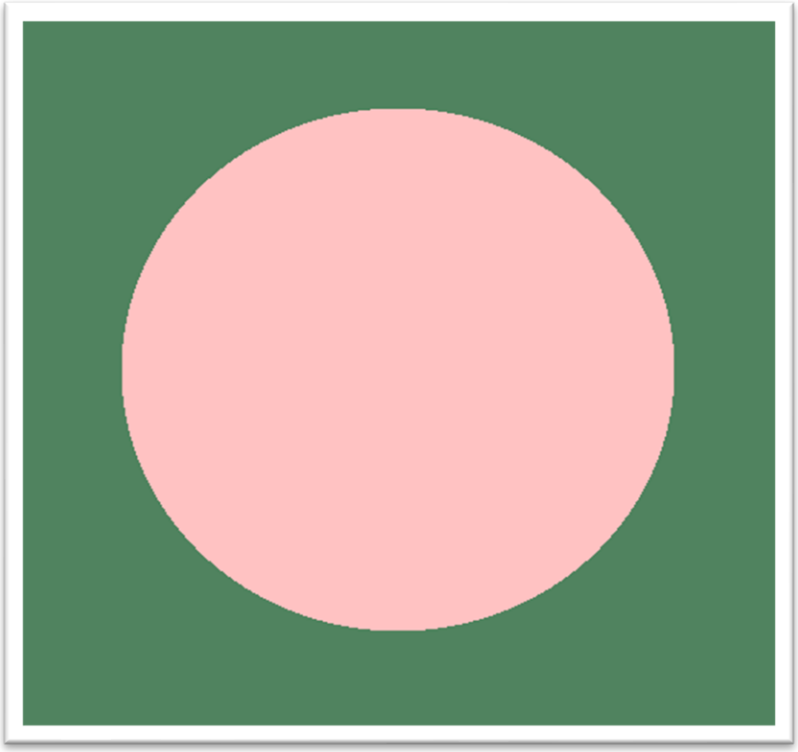
לאחר מכן נחשב צבע ממוצע ונצבע את הפיקסל במשטח הצפיה:

```
pixelcolor = pixelcolor.scale(1d / rays.size());
imageWriter.writePixel(j, i, pixelcolor);
```

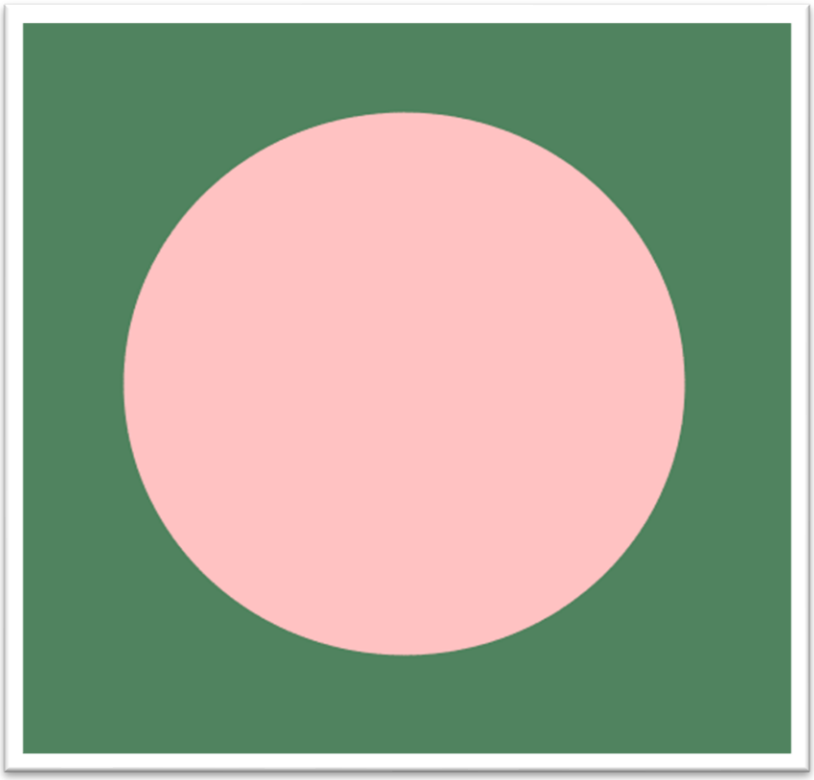
עד כאן המימוש!

תוצאות:

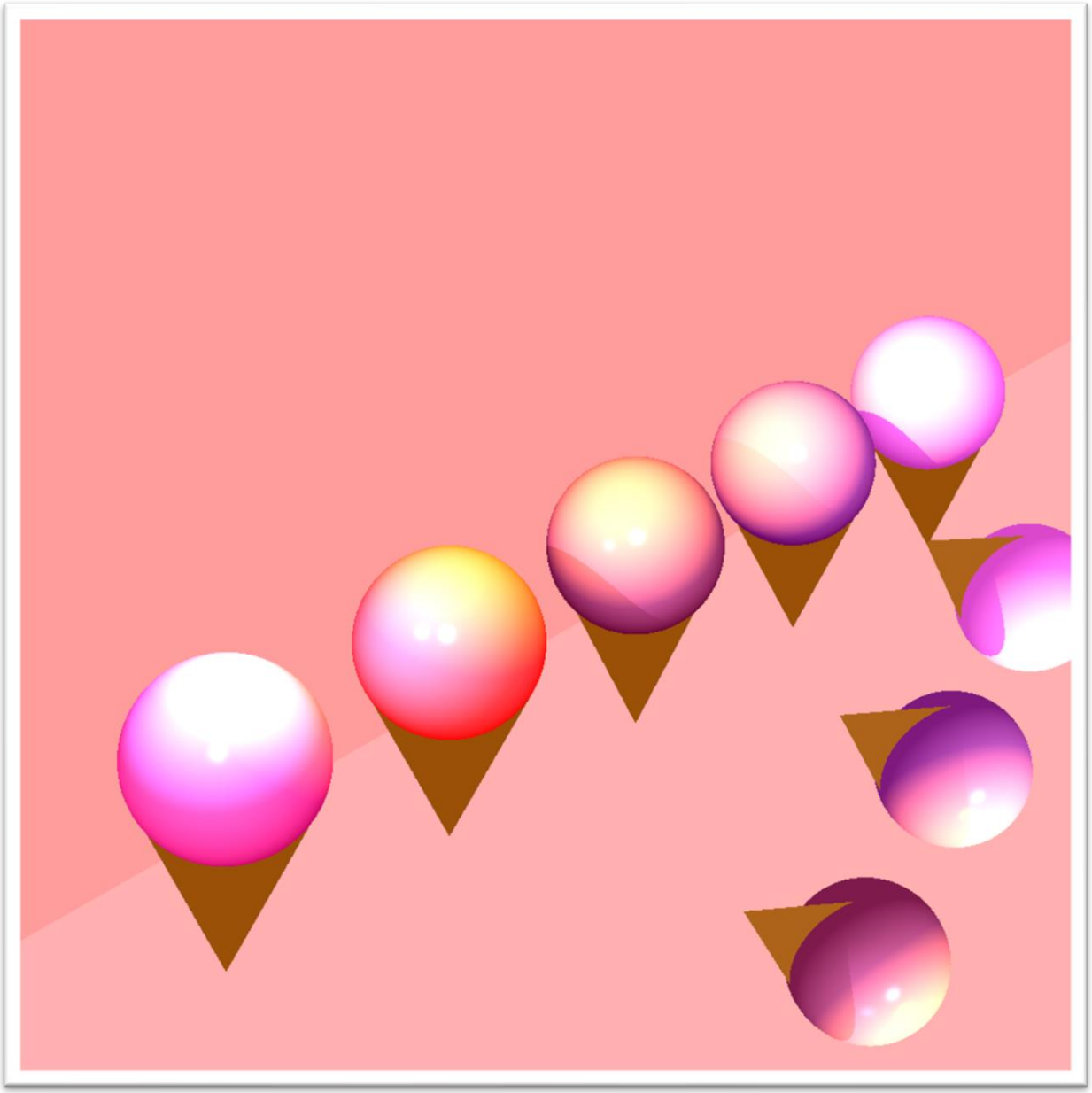
התמונה לפני השיפור:



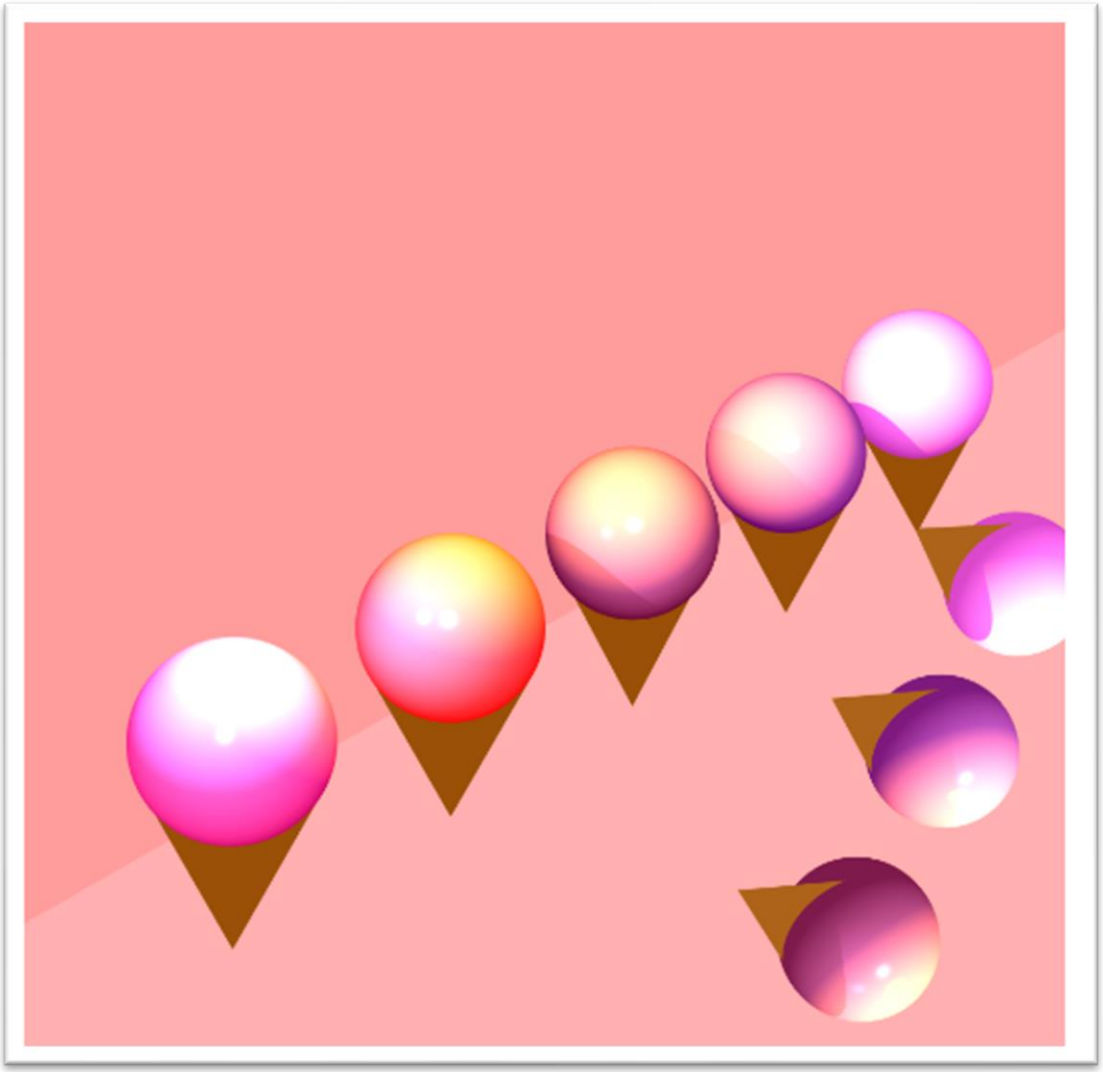
התמונה אחרי השיפור:



תמונה סופית של שלב זה ללא השיפור:



תמונה סופית של שלב זה כולל השיפור:



שיפור עומק השדה depth of field

פתרון עבור סצנה שבה הכל נראה בפוקוס, דבר שאינו נראה אמיתי לעין מפני שבכל שהאובייקט מרוחק
יותר מאיתנו כך הוא יהיה יותר מטושטש.

פתרון לבעיה:

כאשר אנו רוצים לראות אובייקטים שנמצאים במרחק בצורה יותר ברורה אנחנו מצמצמים את האישונים.
כאשר האישונים אינם מצומצמים, אובייקטים מרוחקים יותר, יהיו מטושטשים יותר. על מנת להשיג עומק
שדה, נממש את פעולת הצמצום וההרחבה של האישונים על המצלמה.
נגדיר משטח חדש בשם משטח מיקוד plane focal שמוצב מול משטח הצפייה. הנקודה על משטח
המיקוד המתקבלת על ידי הקרן שנשלחת מהנקודה על משטח הצפייה נקראת נקודת מיקוד.
נגדיר את צמצם המצלמה מדובר בריבוע על משטח הצפייה שמתאר את
"כמות" החלון שדרכו עובר האור. עבור כל נקודה על קצוות הצמצם נשלח קרן שתעבור דרך נקודתהמיקוד.
אם כל הקרניים ששלחנו מהצמצם דרך נקודת המיקוד מחזירות אותו צבע, סימן שהאובייקט נמצא
בפוקוס ואת הצבע שחזר נשים בפיקסל. אחרת, נערבב את הצבעים שחזרו מה שיוצר את האשליה
שהתמונה לא בפוקוס ואת צבע זה נשים בפיקסל.

המימוש בתוכנית:

תחילה הגדרנו שדה במצלמה isDepthOfField שיגיד לנו האם אנחנו מעוניינים בשינוי או לא.
בפונקציה renderImageThreaded נשאל האם מעוניינים בשיפור או לא:

```
//depthOfField
else if (camera.isDepthOfField()&&!camera.isAntialiacing()) {
    List<Ray> rays = camera.constructBeamRay(nX, nY, pixel.col, pixel.row);
    Color pixelcolor = Color.BLACK;
    for (var r : rays) {
        Color rcolor = tracer.traceRay(r);
        pixelcolor = pixelcolor.add(rcolor);
    }
    pixelcolor = pixelcolor.scale(1d / rays.size());
    imageWriter.writePixel(pixel.col, pixel.row, pixelcolor);
}
```

במקרה שמעוניינים הפונקציה הנ"ל תשלח לפונקציה constructBeamRay שמחזירה רשימה של קרניים העוברת דרך הfocal point בקוד הנ"ל יעשה חישוב מומצע הצבע של כל הנקודות בסוף.

```
*/
public List<Ray> constructBeamRay(int nX, int nY, int j, int i) {
    Ray ray = constructRayThroughPixel(nX, nY, j, i);
    List<Ray> raysFocal = constructBeamRayThroughFocalPoint(ray, nX, nY);
    //List<Ray> raysAliesing = constructBeamRayForAntiAliesing(ray, nX, nY);
    List<Ray> rays = new LinkedList<>();
      if (raysAliesing != null)
          rays.addAll(raysAliesing);
    if (raysFocal != null)
        rays.addAll(raysFocal);
    rays.add(ray);
    return rays;
}
```

הפונקציה constructBeamRay שתפקידה ליצור אלומת קרניים תשלח לפונקציה constructBeamRayThroughFocalPoint קרן העוברת דרך מרכז הפיקסל שתעזור לפונקציה לחשב את ה focal point הנוכחי וליצור קרניים בהמשך..

```
*/
public List<Ray> constructBeamRayThroughFocalPoint(Ray ray, int nX, int nY) {
    if (numOfRayFormApertureWindowToFocalPoint == 0)
        return null;
    List<Ray> list = new LinkedList<>();
    double t = distanceToFocalPlane / (_vTO.dotProduct(ray.getDir()));
    list = ray.raySplitter(numOfRayFormApertureWindowToFocalPoint, sizeForApertureWindow, distanceToFocalPlane,
            ray.getPoint(t));
    return list;
}
```

הפונקציה constructBeamRayThroughFocalPoint (במקרה שמספר הקרניים הרצוי לא שווה 0) מחשבת את focal point ושולחת לraySplitter שתפורט בהמשך ומקבלת ממנה רשימת קרניים העוברות דרך הfocal point.

```
*/
public List<Ray> raySplitter(int NumOfRays, double size, double distance, Point3D focalPoint) {
    List<Ray> splittedRays = new LinkedList<>();
    for (int i = 0; i < NumOfRays; i++) {
        Point3D point3d = _p0.randomPointOnSquare(_dir, size, size);
        Vector v = focalPoint.subtract(point3d);
        splittedRays.add(new Ray(point3d, v));
    }
    splittedRays.add(this);
    return splittedRays;
}
```

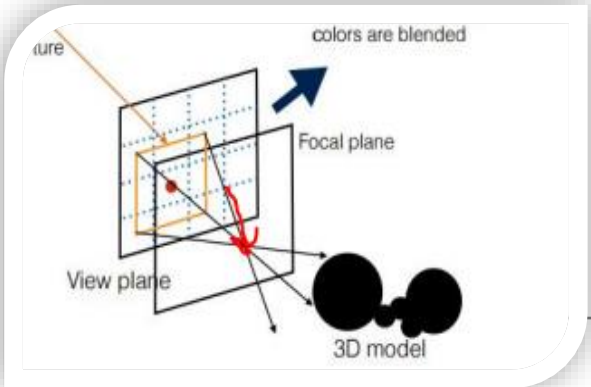
הפונקציה raySplitter שנמצאת במחלקת ray משתמשת בפונקציה randomPointOnSquare שנמצאת במחלקת point שתפקידה להגריל נקודות מריבוע שהוא הצמצם לנקודת הfocal point

הפונקציה פועלת בלולאה ותפקידה לייצר רשימת קרניים שגודלה כמספר הקרניים האמורות להשלח דרך הfocal point

הלולאה:לאחר שחוזרת נקודה מפונקציית randomPointOnSquare יוצרים וקטור שכיוונו כמו בתמונה הממחישה מסומן באדום.

ואז מוסיפים את הקרן לרשימת הקרניים הקרן מורכבת מהנקודה שהוגרלה מהצמצם עם הכיוון של הוקטור שחושב למעלה. ומגרילים עוד נקודה וכן הלאה...

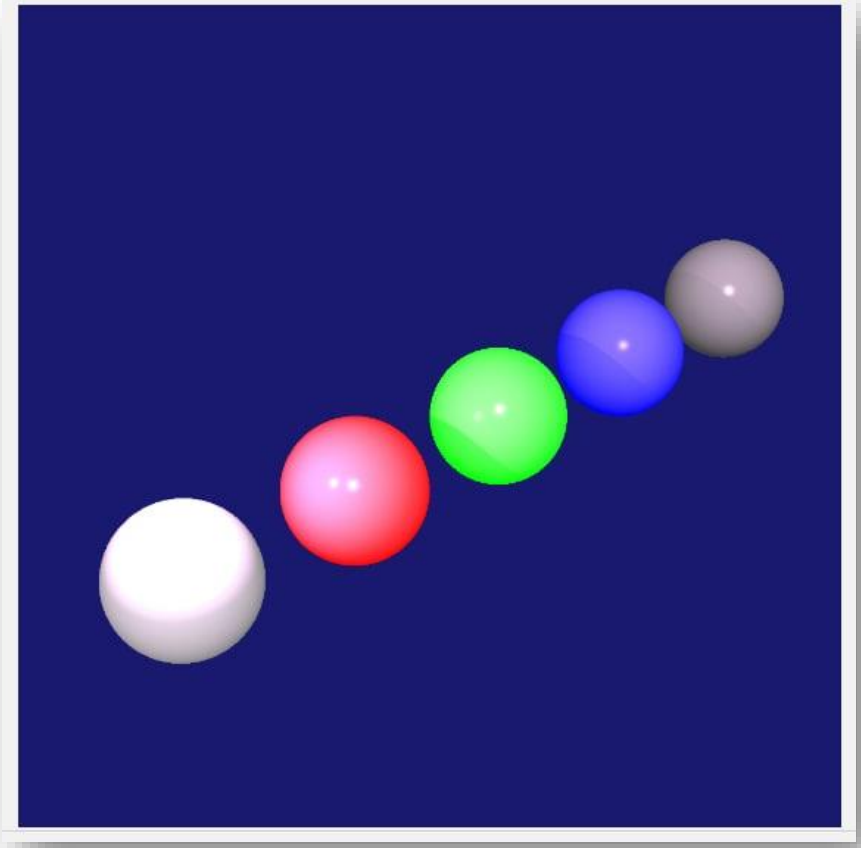
ואז מחזירים את רשימת הקרניים.

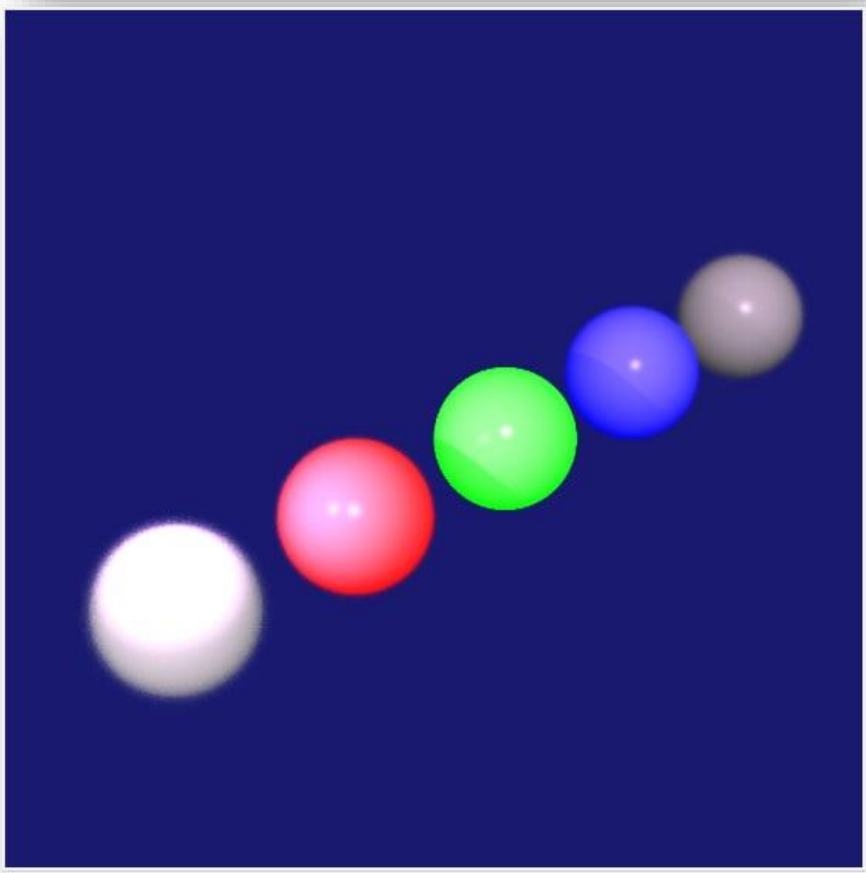


הפונקציה randomPointOnSquare מגרילה נקודות על הצמצם :

```
    */
    public Point3D randomPointOnSquare(Vector dir, double width, double height) {
        Vector firstNormal = dir.createOrthogonalVector();
        Vector secondNormal = firstNormal.crossProduct(dir).normalize();
        Point3D randomCirclePoint = this;
        double r;
        double wHalf = width / 2;
        r = random(0, wHalf);
        double x = random(-r, r);
        double hHalf = height / 2;
        r = random(0, hHalf);
        double y = random(-r, r);
        if (x != 0)
            randomCirclePoint = randomCirclePoint.add(firstNormal.scale(x));
        if (y != 0)
            randomCirclePoint = randomCirclePoint.add(secondNormal.scale(y));
        return randomCirclePoint;
    }
}
```

תמונות לפני ואחרי השיפור:



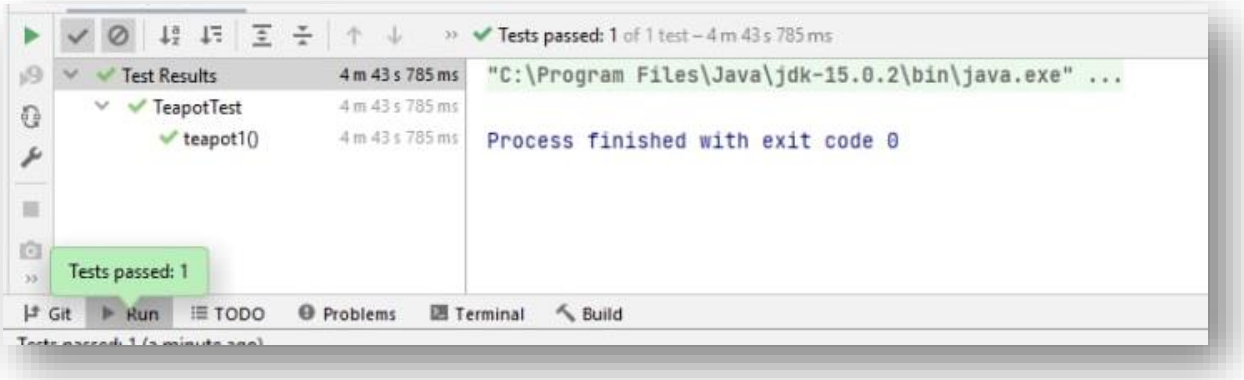


מיני פרויקט 2-שיפור ביצועים

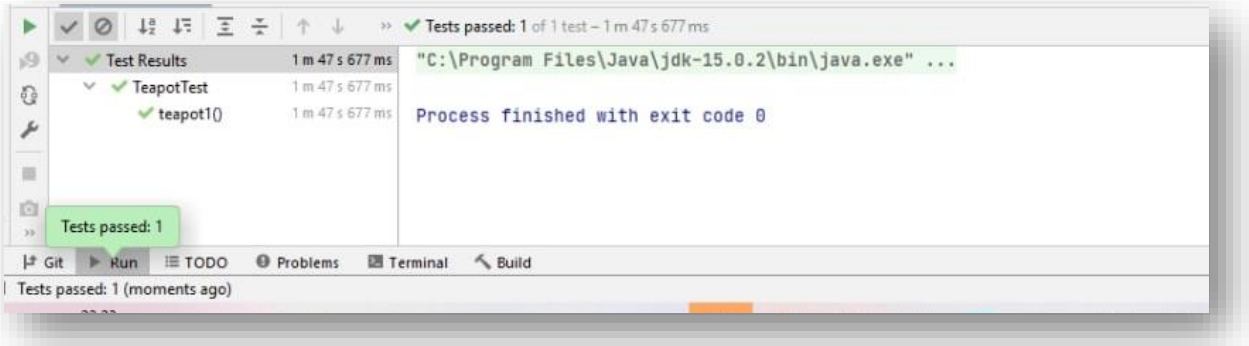
נרצה שתהליך של יצירת תמונה יתבצע יותר מהר לכן נוסיף לו:

א. תהליכונים

מבחן הקומקום לפני השיפור :



מבחן הקומקום לאחר השיפור:



283 שניות לעומת 107 שניות!!!

ב. על מנת להאיץ את התוכנית עוד יותר נשתמש בשיטת ה Boundry box
המטרה: להפחית את כמות הקריאות למציאת נקודות חיתוך בין קרן לאובייקט

נעטוף את האובייקטים שבסצנה בקופסא נבדוק אם הקרן ששלחנו בכלל פוגעת בקופסא שעוטפת את האובייקטים. אם כן אזי נבדוק נקודות חיתוך עם האובייקטים בקופסא. אחרת לא נטרח בכלל לבדוק נקודות חיתוך עם האובייקטים.
* Hierarchy Volume Bounding – תיחום היררכי תלוי נפח.
נתחום את כל הסצנה בקופסא אחת. כל אובייקט נתחום בקופסא משלו לאחר מכן נתחום אובייקטים קרובים בתוך קופסא משלהם ההליך יתבצע ברקורסיה.
כאשר נחפש נקודות חיתוך נבדוק האם הקרן פוגעת בקופסא החיצונית אם כן נתקדם לקופסאות פנימיות יותר, עד שנגיע לקופסא הפנימית ביותר. אם הקרן פוגעת בקופסא הפנימית ביותר אזי עם הפרימיטיביות בתוך הקופסא הזו נמצא נקודות חיתוך.
החלוקה לקופסא בתוך קופסא מאפשרת לנו ליצור היררכיה שתבוטא כעץ. כל קופסא פנימית היא אחד הבנים של הקופסא המכילה אותה כך שהקופסא החיצונית ביותר היא השורש.
זה מאפשר לנו לעבור על הקופסאות בצורה קלה מאוד כי עץ הוא מבנה נתונים לכל דבר שאנחנו מסוגלים להתמודד אתו.

המימוש:

בתחילה הפכנו את Intersectable ל abstract Class
והוספנו לה שדות:

- שדות שמבטאים את גבולות הקופסא מאחר והקופסא מקבילה לצירים מספיק 6 שדות במקום 24.
  2 עבור X, 2 עבור Y ו2 עבור Z . (כל אחד יבטא את ההתחלה והסוף של הקופסא בצירים):

```
protected double minX, maxX, minY, maxY, minZ, maxZ;
```

- שדה שיחושב בהמשך ויבטא את מרכז כל קופסא:

```
protected Point3D middleBoxPoint;
```

השדה יחושב בפונקציה הבאה שגם היא נמצאת במחלקת Intersectable:

```
public Point3D getMiddlePoint() {
    return new Point3D( x: minX + ((maxX - minX) / 2),
            y: minY + ((maxY - minY) / 2),
            z: minZ + ((maxZ - minZ) / 2));
}
```

- שדה נוסף בוליאני שיגיד אם הצורה סופית או לא

```
protected boolean finityShape = false;
```

השדה יאותחל בכל מחלקה כאשר ניצור את הboundry box לדוגמא בספרה נאתחל אותו ב true מאחר והצורה סופית:

```
finityShape = true;
```

- שדה שיגיד האם השיפור ביצועים פעיל או לא

```
protected boolean BVHactivated = false;
```

עד כאן שדות.

כאשר נפעיל את השיפור בטסט :

```
scene.geometries.createBox();
```

תקרא הפונקציה שתאתחל את השדה הבוליאני הנ"ל שאומר האם יש שיפור או לא ל true :

```
public void createBox() {
    BVHactivated = true;
    CreateBoundingBox();
}
```

ואז הפונקציה תקרא לפונקציה CreateBoundingBox שיוצרת קופסאות לגופים:

```
protected void CreateBoundingBox() {
    minX = Double.MAX_VALUE;
    minY = Double.MAX_VALUE;
    minZ = Double.MAX_VALUE;
    maxX = Double.MIN_VALUE;
    maxY = Double.MIN_VALUE;
    maxZ = Double.MIN_VALUE;
    for (Intersectable geo : _intersectables) {
        geo.createBox();
        if (geo.minX < minX)
            minX = geo.minX;
        if (geo.maxX > maxX)
            maxX = geo.maxX;
        if (geo.minY < minY)
            minY = geo.minY;
        if (geo.maxY > maxY)
            maxY = geo.maxY;
        if (geo.minZ < minZ)
            minZ = geo.minZ;
        if (geo.maxZ > maxZ)
            maxZ = geo.maxZ;

    }
    middleBoxPoint = getMiddlePoint();
}
```

הפונקציה הזאת עוברת צורה צורה ויוצרת קופסאות כאשר היא קוראת עבור כל צורה את היצירת קופסא משלה:

ולכן הגדרנו בכל מחלקה של צורה פונקציה שתחשב עבורה את הקופסא לדוגמא עבור sphere

```
@Override
protected void CreateBoundingBox() {
    minX = _center.getX() - radius;
    maxX = _center.getX() + radius;
    minY = _center.getY() - radius;
    maxY = _center.getY() + radius;
    minZ = _center.getZ() - radius;
    maxZ = _center.getZ() + radius;
    middleBoxPoint = _center;
    finityShape = true;
}
```

עכשיו נעבור לבניית העץ:

```
scene.geometries.createGeometriesTree();
```

פונקציה שבהתחלה מוציאה מרשימת הצורות את כל הצורות האינסופיות ואז שולחת לפונקציה רקורסיבית שבונה את העץ הפונקציה הזאת גם תוסיף את כל הגופים האינסופיים לראש העץ:

```
public void createGeometriesTree() {
    LinkedList<Intersectable> shapesWhitOutBox = null;
    for (int i = 0; i < _intersectables.size(); ++i) {
        if (!_intersectables.get(i).finityShape) {
            if (shapesWhitOutBox == null)
                shapesWhitOutBox = new LinkedList<Intersectable>();
            shapesWhitOutBox.add(_intersectables.remove(i));
            i--;
        }
    }
    if (!_intersectables.isEmpty())
        _intersectables = createGeometriesTreeRecursion(_intersectables);
    if (shapesWhitOutBox != null)
        _intersectables.addAll( index: 0, shapesWhitOutBox);
}
```

פונקציה שיוצרת את העץ לוקחת כל פעם איבר (את הראשון) ומחפשת את הצורה הכי קרובה לו היא בודקת על פי מרכזי הקופסאות ואז לאחר שהיא מצאה צורה קרובה היא מוציאה אותו מרשימת הצורות היא מאחדת את 2הצורות האלה לתוך קופסא אחת ואז היא הולכת לפונקציה updateBoxSize שמעדכנת את גבולות הקופסא החדשה שמכילה את 2 הגופים הללו (כמובן שיה גם נקודת מרכז חדשה עבשיו לקופסא הזאת) ואז היא קוראת שוב לפונקציה וכך לאט לאט נוצר העץ מלמטה ללמעלה:

```
List<Intersectable> createGeometriesTreeRecursion(List<Intersectable> finiteShapes) {
    if (finiteShapes.size() == 1)
        return finiteShapes;
    LinkedList<Intersectable> _newShapes = null;
    while (!finiteShapes.isEmpty()) {
        Intersectable first = finiteShapes.remove( index: 0), nextTo = finiteShapes.get(0);
        double minD = first.middleBoxPoint.distance(nextTo.middleBoxPoint);
        int min = 0;
        for (int i = 1; i < finiteShapes.size(); ++i) {
            if (minD == 0)
                break;
            double temp = first.middleBoxPoint.distance(finiteShapes.get(i).middleBoxPoint);
            if (temp < minD) {
                minD = temp;
                nextTo = finiteShapes.get(i);
                min = i;
            }
        }
        if (_newShapes == null)
            _newShapes = new LinkedList<Intersectable>();
        finiteShapes.remove(min);
        Geometries newGeo = new Geometries(first, nextTo);
        newGeo.updateBoxSize(first, nextTo);
        _newShapes.add(newGeo);
        if (finiteShapes.size() == 1)
            _newShapes.add(finiteShapes.remove( index: 0));
    }
    return createGeometriesTreeRecursion(_newShapes);
}
```

הפונקציה שמעדכנת גבולות קופסא:

```
protected void updateBoxSize(Intersectable a, Intersectable b) {
    finityShape = true;
    minX = Double.MAX_VALUE;
    minY = Double.MAX_VALUE;
    minZ = Double.MAX_VALUE;
    maxX = Double.MIN_VALUE;
    maxY = Double.MIN_VALUE;
    maxZ = Double.MIN_VALUE;
    minX = Math.min(a.minX, b.minX);
    minY = Math.min(a.minY, b.minY);
    minZ = Math.min(a.minZ, b.minZ);
    maxX = Math.max(a.maxX, b.maxX);
    maxY = Math.max(a.maxY, b.maxY);
    maxZ = Math.max(a.maxZ, b.maxZ);
    middleBoxPoint = getMiddlePoint();
}
```

הפונקציה findGeoIntersections שולחת לfindIntersactionBound שבודקת האם יש שיפור(האם מעוניינים בו) והאם יש חיתוך עם הקופסא אם יש חיתוך עם הקופסא הוא יחפש את החיתוך עם הצורה במקרה שאין חיתוך עם הקופסא אין צורך לחפש חיתוך עם הגופים והוא יחזיר null .

```
public List<GeoPoint> findGeoIntersections (Ray ray){
    return findIntersactionBound(ray);
}

abstract List<GeoPoint> findGeoIntersections(Ray ray, double maxDistance);

public List<GeoPoint> findIntersactionBound(Ray ray){
    //if there was an intersaction with the box or not
    return BVHactivated&&!isIntersectWithTheBox(ray)?null:findGeoIntersections(ray,Double.POSITIVE_INFINITY);
}
```

הפונקציה שבודקת אם יש חיתוך עם הקופסא או לא

```
public boolean isIntersectWithTheBox(Ray ray) {
    Point3D head = ray.getDir().getHead();
    Point3D p = ray.getP0();
    double temp;

    double dirX = head.getX(), dirY = head.getY(), dirZ = head.getZ();
    double origX = p.getX(), origY = p.getY(), origZ = p.getZ();

    // Min/Max starts with X
    double tMin = (minX - origX) / dirX, tMax = (maxX - origX) / dirX;
    if (tMin > tMax) {
        temp = tMin;
        tMin = tMax;
        tMax = temp;
    } // swap

    double tYMin = (minY - origY) / dirY, tYMax = (maxY - origY) / dirY;
    if (tYMin > tYMax) {
        temp = tYMin;
        tYMin = tYMax;
        tYMax = temp;
    } // swap
    if ((tMin > tYMax) || (tYMin > tMax))
        return false;
    if (tYMin > tMin)
        tMin = tYMin;
    if (tYMax < tMax)
        tMax = tYMax;

    double tZMin = (minZ - origZ) / dirZ, tZMax = (maxZ - origZ) / dirZ;
    if (tZMin > tZMax) {
        temp = tZMin;
        tZMin = tZMax;
        tZMax = temp;
    } // swap
    return tMin <= tZMax && tZMin <= tMax;
}
```

התמונה של הסיום: עם ובלי שיפור קצוות חדים