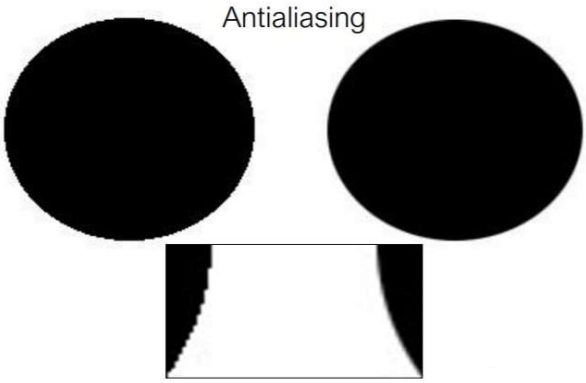
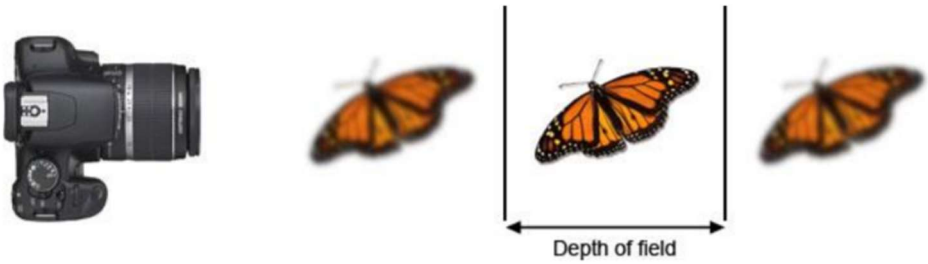
בס"ד

מגישים:
נועם שוובר (214120032)
ישעיה צובל (325889160)

דו"ח – מיני פרוייקט 1

- הוספנו מחלקה Pyramid (פירמידה), על מנת לשפר את הטסטים שלנו.
- הוספת הערות Javadoc והערות כלליות בפונקציות והמחלקות החדשות.
- שיפרנו את הטסט שלנו מהשלב קודם כך שנוכל להבחין בשיפורים בקלות יותר.
- ניסינו 2 אופציות בהזזה על פי מונטה-קרלו – שינוי נקודת שליחת הקרן ושינוי כיוון הקרן. שינוי נקודת הקרן גורמת שהקרן לא תצא מתת-הפיקסל, ושינוי הקרן כדי שתהיה השפעה של יותר מפיקסל אחד.
- שיפורי התמונה שלנו:
  - Anti-Aliasing – החלקת הקצוות.
  - Depth Of Field – עומק שדה.

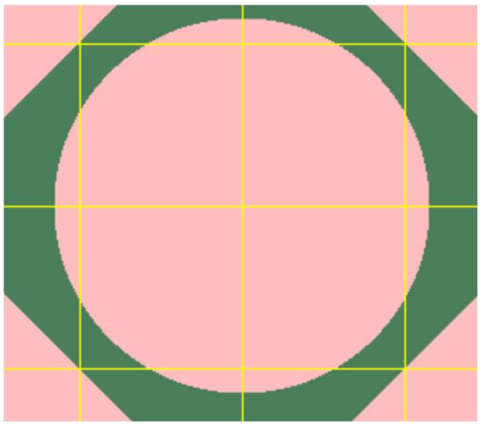




הסבר על השיפורים

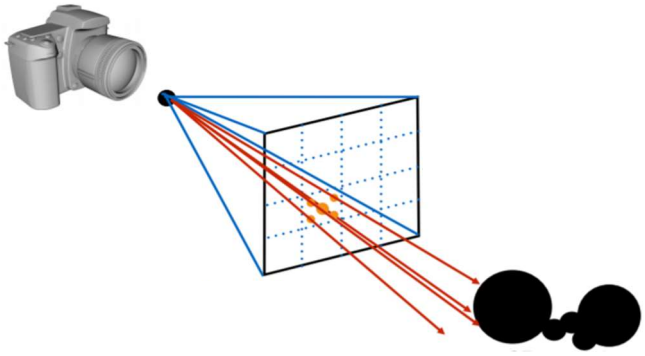
Anti-Aliasing

הבעיה בפרויקט שלנו עד כה היא שהקצוות של הפרויקט מאוד חדים, כיוון שפיקסלים הם מרובעים, ואנחנו לא יכול ליצור אינסוף על אינסוף פיקסלים, יוצא שהעיגולים שלנו נראים כך, מה שנראה מאוד לא מציאותי:



בס"ד

מגישים:
נועם שווב (214120032)
ישעיה צובל (325889160)



על מנת לתקן זאת, אנו שולחים מספר קרניים על כל פיקסל ויוצרים מין רשת של "תת-פיקסל", ומחשבים את הממוצע של הצבע שיוצא מעקיבת הקרניים (trace ray), כך שבמקרה של קצה צורה, הממוצע יתחשב גם בצבע של הקרניים האחרות:

לא שלחנו קרניים אחידות (כלומר כל קרן למרכז התת-פיקסל) אלא לכיוון אקראי, על פי שיטת מונטה-קרלו.

הקוד שלנו

במחלקה Render, בפונצקציה renderImage:

```
for (int i = 0; i < imageWriter.getNx(); i++) {
    for (int j = 0; j < imageWriter.getNy(); j++) {
        if (ANTI_ALIASING) { // If anti-aliasing is enabled - a
boolean constant.
            Color color = Color.BLACK;

            // A function to create list of rays to calculate
            // the average color (RAYS is a constant, how many rays
in each
            // column and row).
            List<Ray> rays =
camera.constructRaysThroughPixelAA(imageWriter.getNx(),
                    imageWriter.getNy(), i, j, RAYS);

            for (Ray ray: rays) { // A loop to sum all colors
                color = color.add(rayTracer.traceRay(ray));
            }

            // Writing the average color to the pixel.
            imageWriter.writePixel(i, j, color.reduce(rays.size()));
        }
        else // If not enabled, shoot only one ray.
                …
    }
}
```

בס"ד

מגישים:
נועם שוובר (214120032)
ישעיה צובל (325889160)

במחלקה Camera, יצרנו פונקציה חדשה בשם ConstructRaysThroughtPixelAA, שמחזירה מספר קרניים לפיקסל:

```
/**
 * Creates a list of rays that goes through a given pixel (as a grid
of sub-pixels)
 * in random directions.
 * @param nX number of pixels on X axis in the view plane
 * @param nY number of pixels on Y axis in the view plane
 * @param i Y coordinate of the pixel
 * @param j X coordinate of the pixel
 * @param rays The amount of rays in each column and row.
 * @return A list of rays around that pixel.
 */
public List<Ray> constructRaysThroughPixelAA(int nX, int nY, int i,
int j, int rays) {
    List<Ray> lst = new ArrayList<>();

    // Choosing the biggest scalar to scale the vectors.
    double rY = height / (2 * nY * rays * 0.05 * distance),
           rX = width / (2 * nX * rays * 0.05 * distance);
    Random random = new Random();

    //Constructing (rays * rays) rays in random directions.
    for (int k = 0; k < rays; k++) {
        for (int l = 0; l < rays; l++) {
            //Constructing a ray to the middle of the current
subpixel.
            Ray ray = constructRayThroughPixel(nX * rays, nY * rays,
rays * i + k, rays * j + l);

            // Creating a random direction vector.
            Vector rnd = vUp.scale(randomDouble(-rY, rY))
                    .add(vRight.scale(randomDouble(-rX, rX)));

            // Adding the random vector to the ray.
            lst.add(new Ray(ray.getP0(), ray.getDir().add(rnd)));

        }
    }

    return lst;
}
```

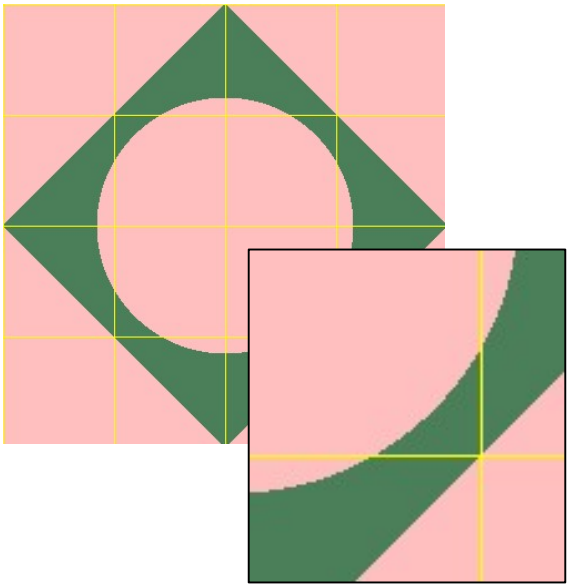
בס"ד

מגישים:
נועם שוובר (214120032)
ישעיה צובל (325889160)

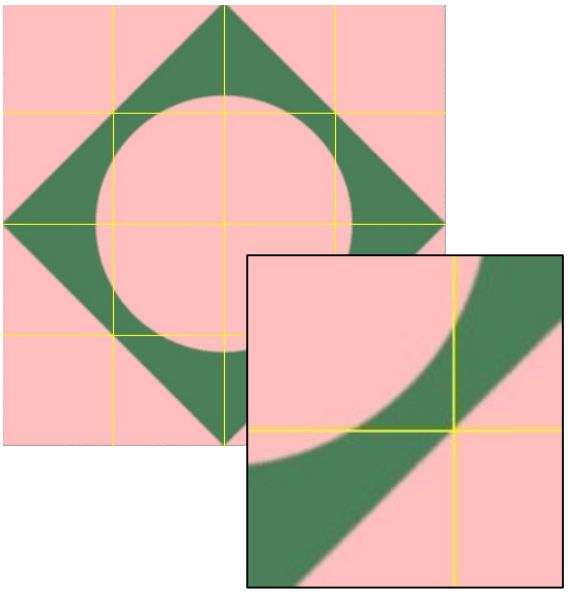
דוגמה לשיפור:

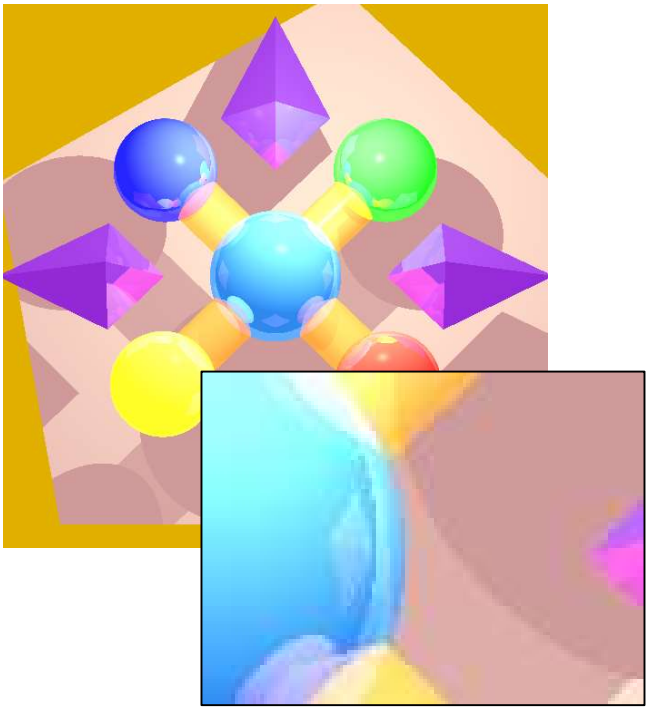
ירינו 100 קרניים על כל פיקסל (דהיינו רשת של 10x10 תתי-פיקסלים), וכך זה נראה:

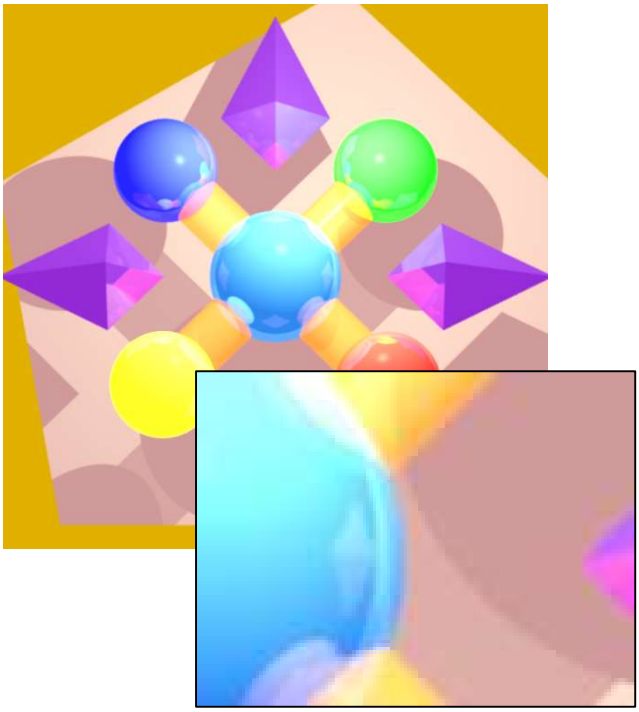
לפני השיפור:



אחרי השיפור:





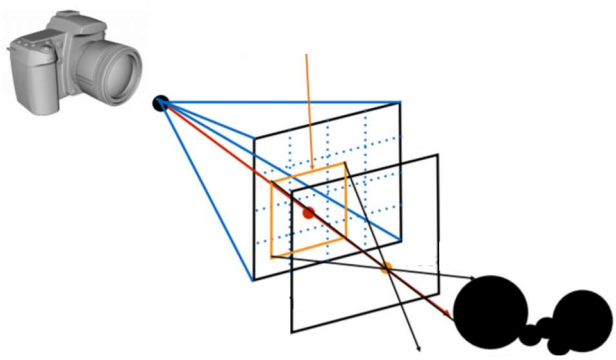


בס"ד

מגישים:
נועם שווב (214120032)
ישעיה צובל (325889160)

Depth Of Field

בעיה נוספת בפרויקט שלנו עד כה היא שכל הצורות בסצנה בפוקוס, מה שנראה מזויף וגם מונע ממנו מספר יכולות כמו לתת פוקוס על הדבר המרכזי בסצנה.



על מנת לתקן זאת, אנו שולחים מספר קרניים על כל פיקסל מצדדי הצמצם, בדומה למה שקורה במצלמה אמיתית, ומחשבים את הממוצע של הצבע שיוצא מעקיבת הקרניים (trace ray), כך שזה יתחשב בפוקוס ובגודל העדשה, וכך ממוצע הצבע בקרניים גורם לאפקט הטשטוש:

בס"ד

מגישים:
נועם שוובר (214120032)
ישעיה צובל (325889160)

הקוד שלנו

במחלקה Camera, יצרנו פונקציה חדשה בשם ConstructRaysThroughtPixelDoF, שמחזירה מספר קרניים בהתאם לפיקסל (i, j) הנוכחי:

```
public List<Ray> constructRaysThroughPixelDoF(int nX, int nY, int i,
int j, int rays) {
    Point3D PcV = p0.add(vTo.scale(distance)); //Pc view plane
    Point3D PcF = p0.add(vTo.scale(distance + focalDistance)); //Pc
focal plane

    Plane viewPlane = new Plane(PcV, vTo); // Creates the view plane.
    Plane focalPlane = new Plane(PcF, vTo); // Creates the focal
plane.

    Ray ray = constructRayThroughPixel(nX, nY, i, j); // Constructs a
ray to the middle of the current pixel

    GeoPoint viewIntersection =
viewPlane.findGeoIntersections(ray).get(0); // Only one intersection
point in plane.
    GeoPoint focalPoint =
focalPlane.findGeoIntersections(ray).get(0);

    List<Ray> lst = new ArrayList<>();

    double rX = apertureWidth / (2 * rays), // The width of a half
sub-pixel in the aperture.
            rY = apertureHeight / (2 * rays); // The height of a half
sub-pixel in the aperture.

    // Constructing (rays * rays) rays, while moving the point, but
all the rays
    // will go throughout the focal point.
    for (int k = -rays / 2; k < Math.ceil(rays / 2d); k++) {
        for (int l = -rays / 2; l < Math.ceil(rays / 2d); l++) {
            Point3D point = viewIntersection.point;
            if (k != 0)
                point = point.add(vRight.scale(k * rX));
            if (l != 0)
                point = point.add(vUp.scale(l * rY));

            lst.add(new Ray(point,
focalPoint.point.subtract(point)));
        }
    }

    return lst;
}
```

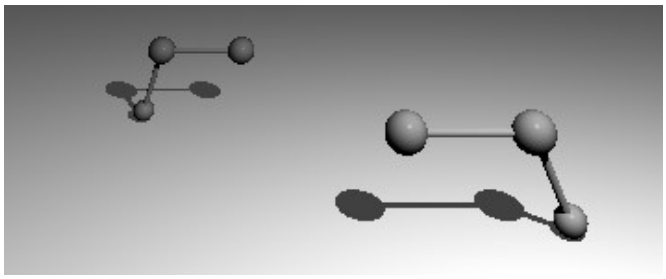
בנוסף, במחלקה Renderer עשינו כמו בAA – לולאה שעושה מעקב אחרי הקרניים (ray trace) ומחשבת את הממוצע של הצבעים.

בס"ד

מגישים:
נועם שווב (214120032)
ישעיה צובל (325889160)

דוגמאות לשיפור

לפני השיפור:



אחרי השיפור: