דו"ח סיכום
למיני פרויקט

תוכן עניינים

תיאור הפרויקט

<u>רקע:</u>

הפרוייקט נכתב בשיטת Pair programing– מתכנת אחד כותב ומתכנת אחד בודק שאין טעויות.

פיתחנו כלי שנוכל בעזרתו לרנדר תמונה, ע"י הגדרת סצנה המכילה צורות גיאומטריות ומקורות אור.

הפרויקט נכתב ב- JAVA בInteliJ עם שימוש בJunit.

<u>מבנה</u>

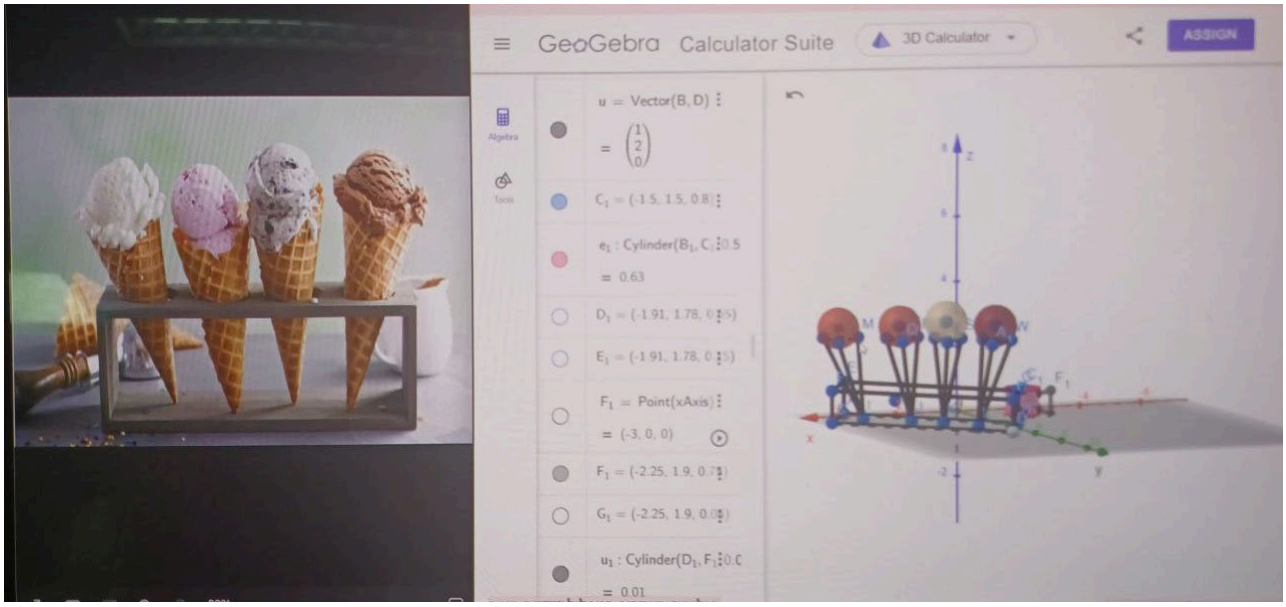
חילקנו את המחלקות לחבילות כך שכל מחלקה בחבילה שקשורה אליה.

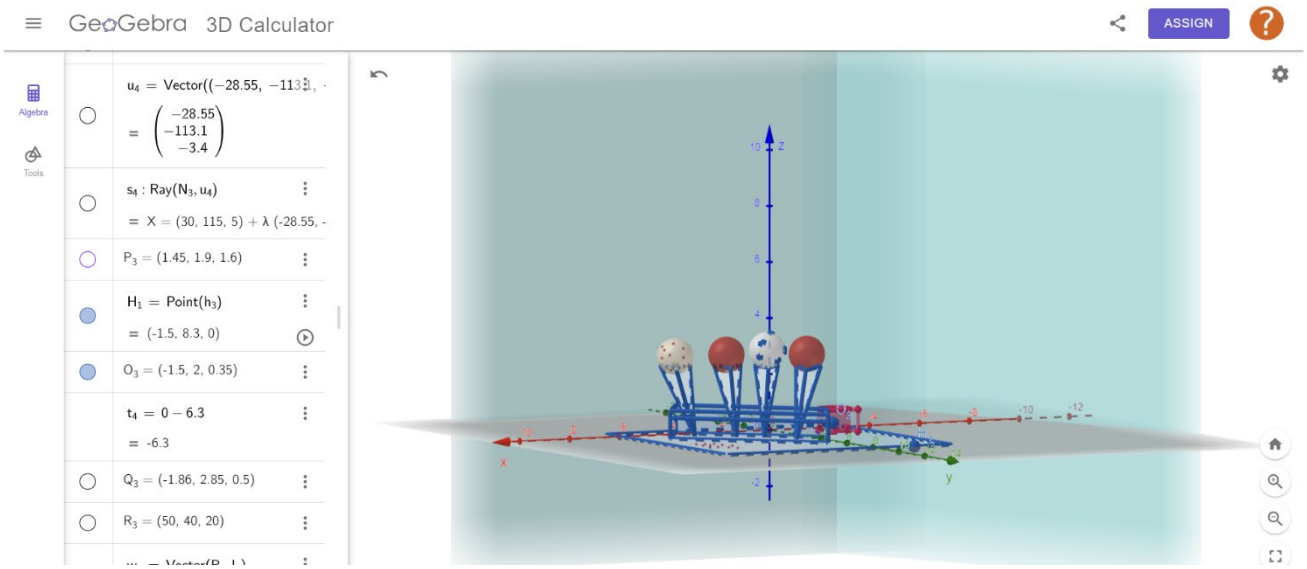
- החבילה Primitives מכילה את כל הצורות הפרימיטיביות, ביניהם נקודה, וקטור, צבע, סוג חומר.
- החבילה Geometries מכילה ממשק לצורות גאומטריות המחייב כל צורה גיאומטרית לממש פונקציות חיתוכים ופונקציות בסיסיות בזמן יצירת תמונה. היא מכילה גאומטריות כגון ספרה, מישור, משולש ועוד.
- החבילה Lighting מכילה ממשק המחייב כל מקור אור להחזיר את הכיוון והעוצמה שלו וכן מקורות אור כמו אור כיווני, אור ממוקד ואור סביבתי.
- החבילה Scene מכילה את המחלקה Scene שמגדירה Scene עם רשימת גאומטריות ומקורות אור שאפשר לרנדר.
- החבילה Renderer שאחראית על הרנדור של התמונה, מכילה את המחלקות RayTracerBase, Camera , פונקציות אלא אחראיות על שליחת קרניים וחישוב הצבע של הפיקסלים בתמונה.
בנוסף יש שם את המחלקה ImageWriter שכותבת אל התמונה את מה שיצרנו ויוצרת קובץ jpg.

שיטת עבודה

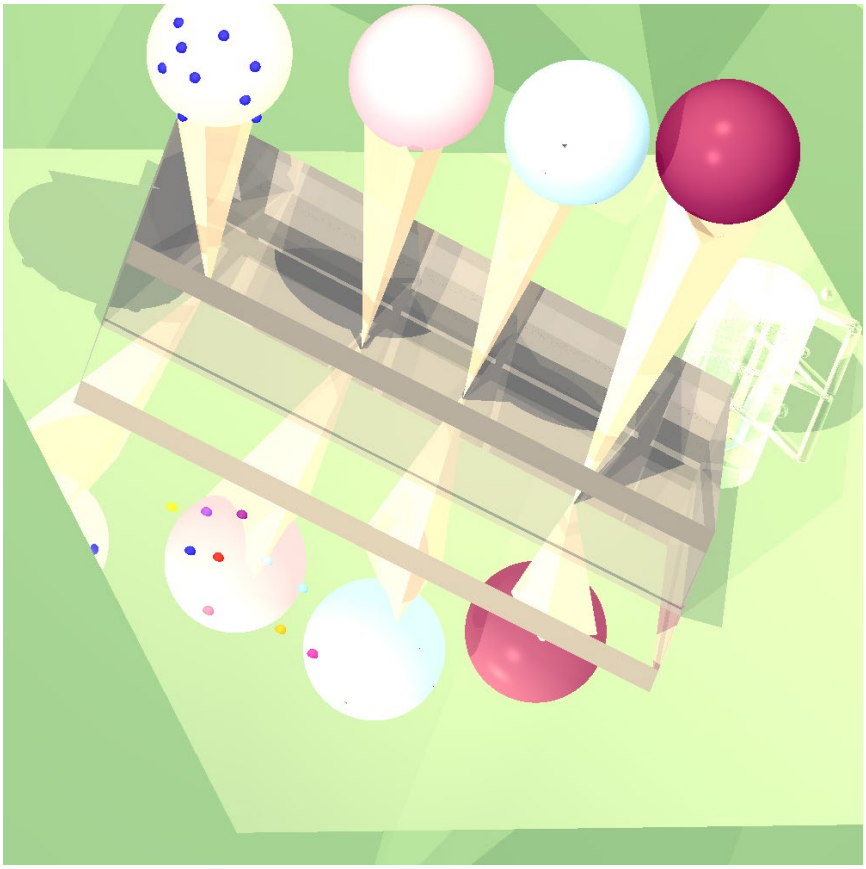
לאורך כל הפרוייקט הקפדנו על עקרון ה-TDD, עבור כל מתודה ומחלקה כתבנו טסט שבודק את תקינות המימוש שלהם ובנוסף מימשנו טסטים הבודקים את האינטגרציה בין המחלקות.

שלבי יצירה ב- GeoGebra





תוצאה בIntel:



MINIP 1

שיפור תמונה

Anti-Aliasing:

הבעיה- בשיטה הנוכחית, כל פיקסל מקבל את הצבע שיש במרכז שלו, ללא התחשבות במה שקורה בשאר הפיקסל. לכן בתמונה שנוצרת יש מעברים חדים בין צבע של פיקסל לצבע של הפיקסל לידו, ונקבל שצורות שאינן ישרות יראו מחוספסות והחלוקה לפיקסלים תבלוט ונקבל מראה פחות טוב.

הפתרון- יצירת קרניים לנקודות נוספות בפיקסל מלבד המרכז, וחישוב ממוצע הצבעים מכל הצבעים של הקרניים שהטלנו לאותו פיקסל.
ככה, בקצוות הצורה יתקבל צבע משולב והגבולות יהיו "רכים" יותר.

נשתמש ב- super sampling על פי האלגוריתם הרנדומלי – נשלח קרניים שמממוקמות באופן רנדומלי בתוך הפיקסל בנוסף לקרן המקורית.

השינויים שביצענו:

הוספנו במחלקת Camera שדה antiAliasingLevel מאותחל ל1, ופונקציית set כדי שנוכל להגדיר בבניית המצלמה את כמות קרניים שנטיל דרך כל פיקסל על מנת להתגבר על הבעיה.

```
private int antialiasingLevel = 1;  3 usages
```

```
public Builder setAntialiasingLevel(int antialiasingLevel) {
    camera.antialiasingLevel = antialiasingLevel;
    return this;
}
```

בנוסף, בפונקציה renderImage (שאחראית להטיל קרן דרך מרכז כל פיקסל) נבצע את השינוי הבא:
כשנקרא לפונקציה castRay נשלח לה את הפרמטר החדש שהוספנו, בין אם המשתמש רצה בשיפור ובין אם לא. (שכן אם לא דאגנו לאתחל את הפרמטר לאחד)

```
public Camera renderImage() {  25 usages  Brachi20
    int nX = imageWriter.getNx();
    int nY = imageWriter.getNy();
    // Verify that nX and nY are not zero to avoid division by zero
    if (nY == 0 || nX == 0)
        throw new IllegalArgumentException("It is impossible to divide by 0");
    // Initialize the pixel manager
    pixelManager = new PixelManager(nY, nX,  interval: 0.1);
    // Check if the number of threads is 0
    if (threadsCount == 0) {
        for (int i = 0; i < nY; ++i)
            for (int j = 0; j < nX; ++j)
                castRay(nX, nY, j, i, antialiasingLevel);
    }
    else { // see further... option 2
        var threads = new LinkedList<Thread>(); // list of threads
        while (threadsCount-- > 0) // add appropriate number of threads
            threads.add(new Thread(() -> { // add a thread with its code
                PixelManager.Pixel pixel; // current pixel(row,col)
                // allocate pixel(row,col) in loop until there are no more pixels
                while ((pixel = pixelManager.nextPixel()) != null)
                    // cast ray through pixel (and color it - inside castRay)
                    castRay(nX, nY, pixel.col(), pixel.row(),antialiasingLevel);
            }));
        // start all the threads
        for (var thread : threads) thread.start();
        // wait until all the threads have finished
        try {
            for (var thread : threads) thread.join();
        } catch (InterruptedException ignore) {
        }
    }
    return this;
}
```

בפונקציה castRay נבדוק האם המשתמש רצה בשיפור שהוספנו, ע"י בדיקה ש antiAliasingLevel שונה מ1.
אם כן, נקרא לפונקציה חדשה שנוסיף בשם costructRays שתקבל פרמטר numRays עבור מספר הקרניים שרוצים להטיל ולא קרן בודדת כמו שהיה עד עכשיו, היא תיצור כמות של קרניים דרך כל פיקסל לטווח מטרה של מלבן מסביב למרכז הפיקסל, ותחזיר רשימה של כל הקרניים.
לאחר מכן נקרא לפונקציה חדשה שהוספנו בשם avarageColor שתחשב לנו את הצבע הממוצע של כל הצבעים שכל קרן החזירה.
להלן הפונקציה castRay:

```
private void castRay(int nX, int nY, int column, int row, int numRays) {  2 usages  Brachi20 +1
    Color color = Color.BLACK;
    if (numRays == 1) {
        // Trace a single ray
        Ray ray = constructRay(nX, nY, column, row);
        color = rayTracer.traceRay(ray);
    } else {
        boolean colorsDifferernt = false;
        if(adaptive){
        // Trace multiple rays
        List<Ray> rays = constructRays(nX, nY, column, row,  numRays: 5);
        if (!rays.isEmpty()) {
            // Handle empty rays list, if applicable
            Color firstColor = rayTracer.traceRay(rays.get(0));
            // Check if all rays produce the same color
            for (Ray ray : rays) {
                Color currentColor = rayTracer.traceRay(ray);
                if (!currentColor.equals(firstColor)) {
                    colorsDifferernt = true;
                    break;
                }
            }
            if (colorsDifferernt) {
                // If the colors are different, construct more rays
                 rays = constructRays(nX, nY, column, row, numRays);
                 color = AvrageColor(rays, color);
            } else {
                color = firstColor;
            }
        }
        } else{
            List<Ray> rays = constructRays(nX, nY, column, row, numRays);
            color= AvrageColor(rays, color);

    }}
```

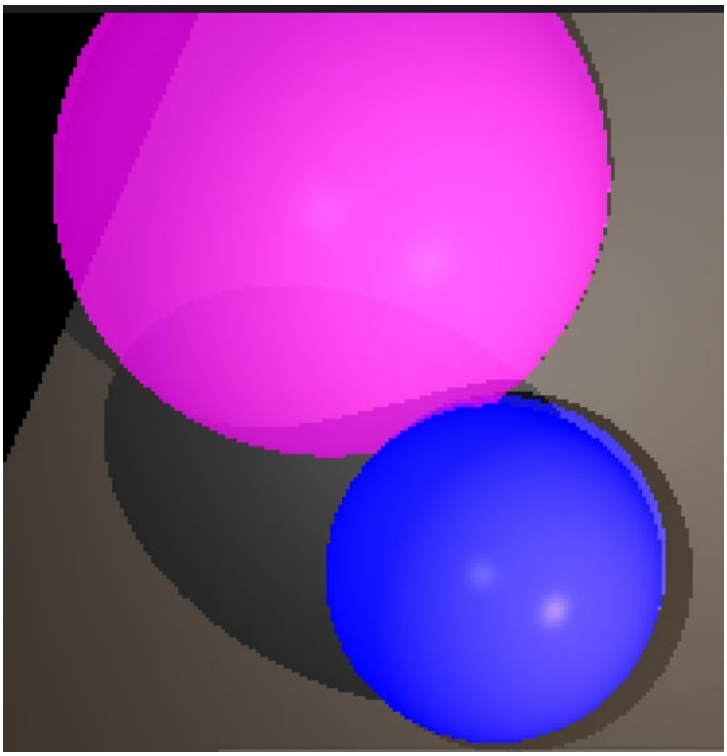
להלן הפונקציה החדשה constructRays

```
 */
public List<Ray> constructRays(int nX, int nY, int j, int i, int numRays) {  3 usages  Brachi20
    double rX = width / nX;
    double rY = height / nY;

    double Xj = (j - (nX - 1) / 2d) * rX;
    double Yi = -(i - (nY - 1) / 2d) * rY;

    Point pCenter = p0.add(vTo.scale(distance));

    Point Pij = pCenter;
    if (!isZero(Xj))
        Pij = Pij.add(vRight.scale(Xj));
    if (!isZero(Yi))
        Pij = Pij.add(vUp.scale(Yi));

    List<Ray> rays = new ArrayList<>(numRays);
    Random rand = new Random();

    for (int k = 0; k < numRays; k++) {
        // Generate random point within the pixel area
        double offsetX = (rand.nextDouble() - 0.5) * rX; // random value between -rX/2 and rX/2
        double offsetY = (rand.nextDouble() - 0.5) * rY; // random value between -rY/2 and rY/2

        Point randomPoint = Pij.add(vRight.scale(offsetX)).add(vUp.scale(offsetY));
        Vector rayDirection = randomPoint.subtract(p0);
        rays.add(new Ray(p0, rayDirection));
    }

    return rays;
}
```

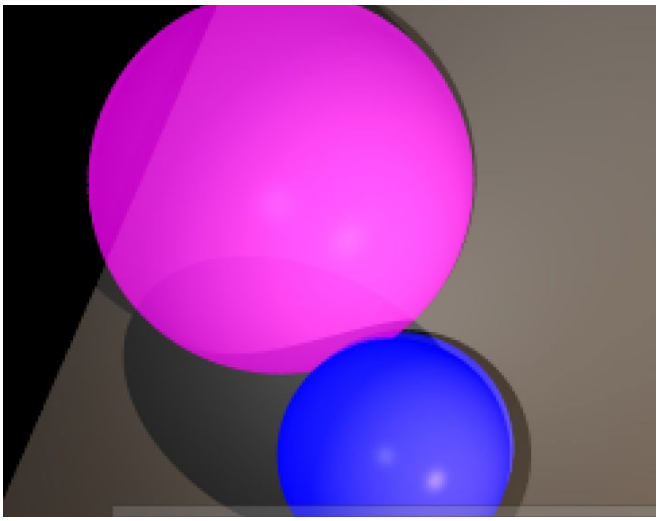
להלן הפונקציה הנוספת שהזכרנו, avarageColor:

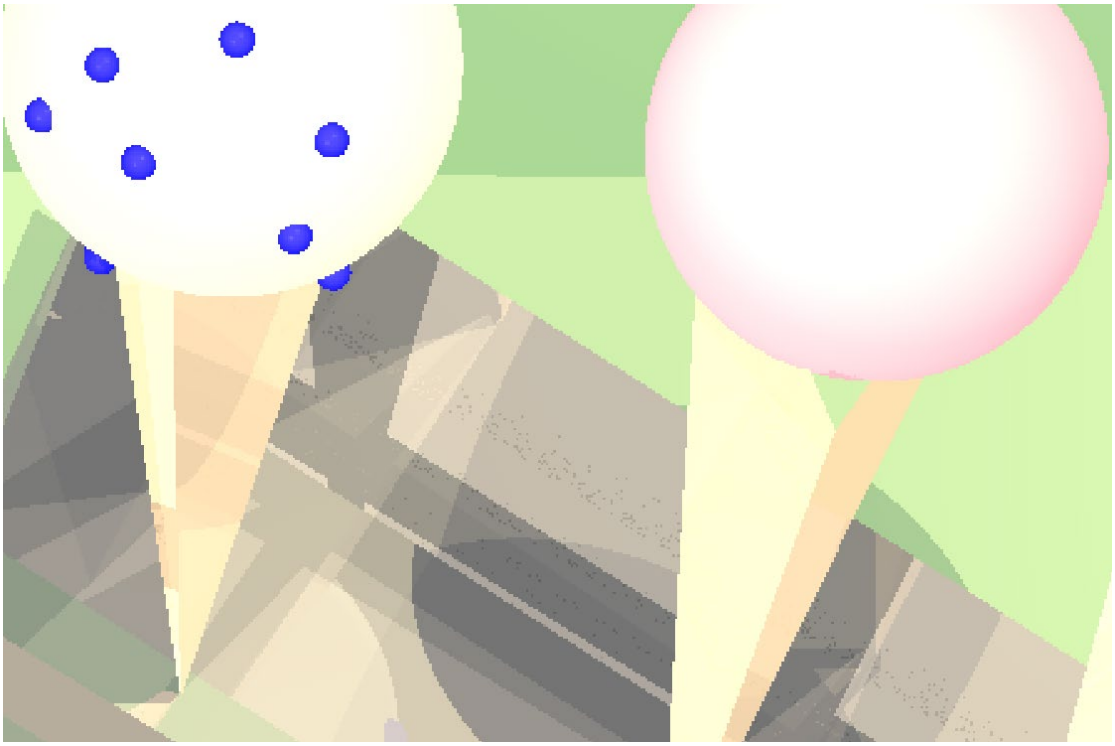
```
    /**
 * Calculates the average color from a list of rays.
 * @param rays The list of rays to calculate the average color from.
 * @param color The initial color to add to.
 * @return The average color from the list of rays.
 */
private Color AvrageColor(List<Ray> rays, Color color) {  2 usages  Brachi20
    if(rays.isEmpty())
        return Color.BLACK;
    for (Ray ray : rays) {
        color = color.add(rayTracer.traceRay(ray));
    }
    color = color.reduce(rays.size());
    return color;
}
```

לפני השיפור

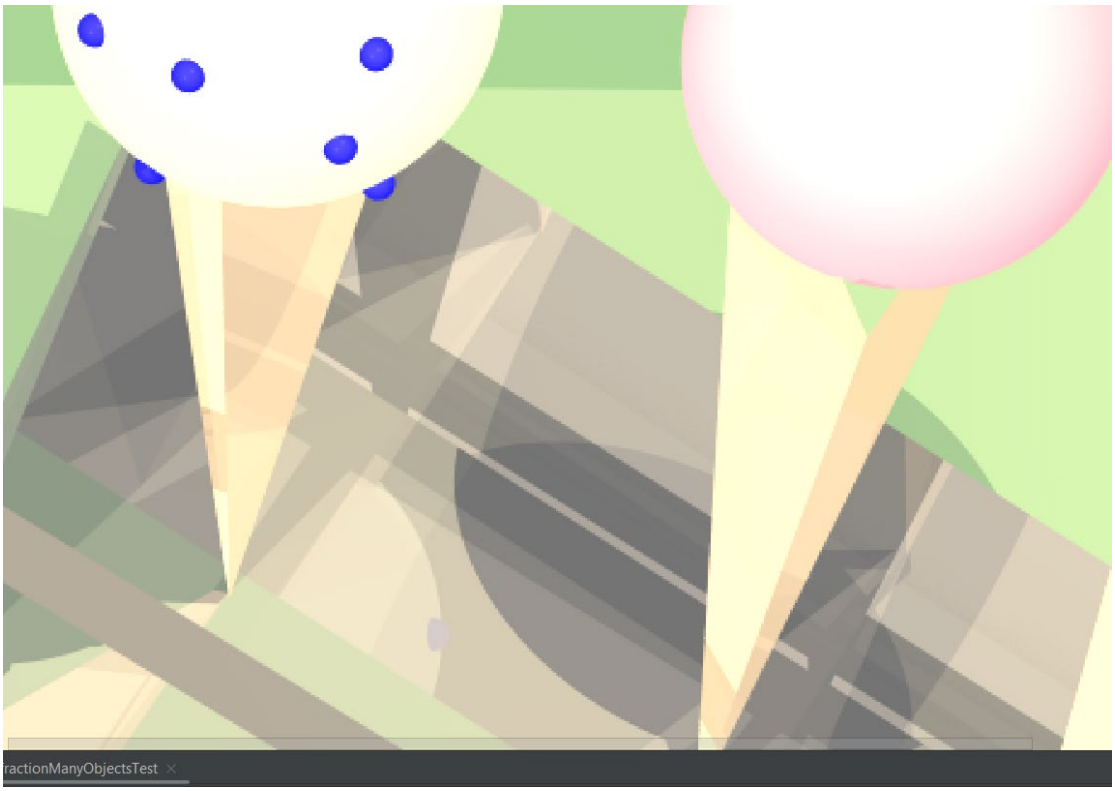


אחרי השיפור





אחרי השיפור



MINIP 2

שיפור ביצועים

Adaptive Super Sampling:

כדי לשפר את התמונה הוספנו את השיפור anti-aliasing שעבור כל פיקסל, במקום לירות קרן אחת יורה כמה קרניים ואז מחשב את הצבע הממוצע של כל הנקודות שנפגעו.
זה משפר את התמונה כאשר יש מעבר צבע בתוך הפיקסל ואז המעבר הזה יהיה פחות חד לעין.
כמובן שבמקרה שכל הפיקסל שאותו אנו רוצים לצבוע הוא באותו צבע, אז זה מיותר לירות עשרות קרניים ולחשב את הצבע של כל אחד.
הפתרון של adaptive בא לטפל במקרה הזה. לפיו , לפני שנחשב את הצבע של כל הנקודות דרך הטלת כל קרן, נבדוק קודם עבור מספר מצומצם של נקודות שנבחר , ואם נקבל שם את אותו צבע אז אין סיבה להטיל עוד קרניים ולבדוק את השאר.
בחרנו לבדוק את הנקודות במרכז ובפינות, שהרי אם הם באותו צבע אז סביר להניח שכל הפיקסל באותו צבע ואז פשוט נצבע בצבע הזה.
אחרת, נבצע את החישוב רגיל לפי השיטה שתארנו בשיפור anti- aliasing.

השינויים שביצענו-

הוספנו ב Camera שדה בוליאני חדש adaptive שיאותחל לfalse ופונקציית set כך שנוכל לסמן אם רוצים להפעיל את השיפור adaptive או לא.

```
private boolean adaptive = false;  2 usages
```

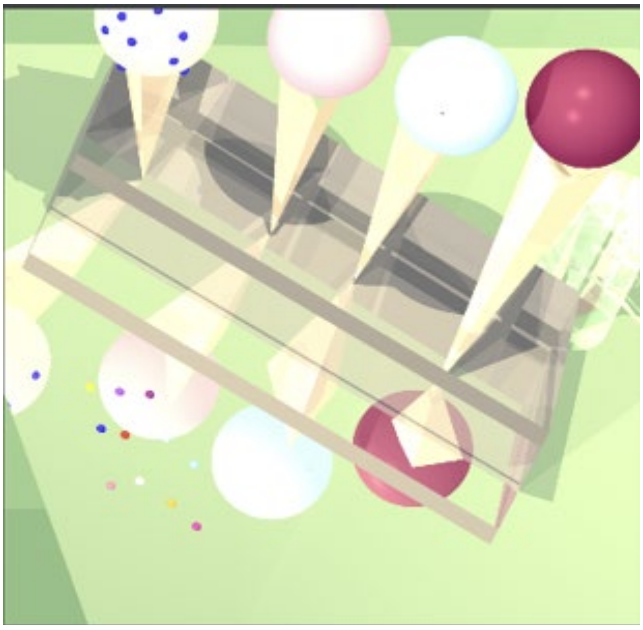
```
public Builder setAdaptive(boolean adaptive) {  6 usages  Brachi20
    camera.adaptive = adaptive;
    return this;
}
```

שינינו את הפונקציה castRay כך:
היא בודקת האם הדגל של השיפור דלוק, אם כן אז נשלח בכל פעם 5 קרניים דרך כל פיקסל ונבדוק אם בכולם חזר אותו צבע.
אם כן, אז זה יהיה הצבע של הפיקסל אבל אם חזר צבעים שונים אז צבע הפיקסל יהיה הצבע הממוצע שיחושב על ידי הפונקציה החדשה averageColor .
להלן הפונקציה castRa עם השינויים:

```
private void castRay(int nX, int nY, int column, int row, int numRays) {  2 usages  Brachi20 +1
    Color color = Color.BLACK;
    if (numRays == 1) {
        // Trace a single ray
        Ray ray = constructRay(nX, nY, column, row);
        color = rayTracer.traceRay(ray);
    } else {
        boolean colorsDifferernt = false;
        if(adaptive){
        // Trace multiple rays
        List<Ray> rays = constructRays(nX, nY, column, row,  numRays: 5);
        if (!rays.isEmpty()) {
            // Handle empty rays list, if applicable
            Color firstColor = rayTracer.traceRay(rays.get(0));
            // Check if all rays produce the same color
            for (Ray ray : rays) {
                Color currentColor = rayTracer.traceRay(ray);
                if (!currentColor.equals(firstColor)) {
                    colorsDifferernt = true;
                    break;
                }
            }
            if (colorsDifferernt) {
                // If the colors are different, construct more rays
                 rays = constructRays(nX, nY, column, row, numRays);
                 color = AvrageColor(rays, color);
            } else {
                color = firstColor;
            }
        }
        } else{
            List<Ray> rays = constructRays(nX, nY, column, row, numRays);
            color= AvrageColor(rays, color);

    }}

    // Write the computed color to the image and mark the pixel as done
    imageWriter.writePixel(column, row, color);
    pixelManager.pixelDone();
}
```

זמן הריצה לפני השיפור :

שעה ו54 דקות כאשר מטילים 100 קרניים דרך כל פיקסל



```
Tests passed: 1 of 1 test – 1 hr 54 min
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
100.0%
Process finished with exit code 0
```

זמן הריצה אחרי השיפור:

17 דקות ו24 שניות כאשר מטילים 100 קרניים דרך כל פיקסל

```
ReflectionRefractionTests.reflectionRefractionManyObjectsTest
Tests passed: 1 of 1 test – 7 min 48 sec
ReflectionRefractionTests (renderer 7 min 48 sec
reflectionRefractionManyObjec 7 min 48 sec
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
100.0%
Process finished with exit code 0
```

ניתן לראות שאחרי השיפור Adaptive קיבלנו זמן ריצה קטן פי 8!!

Multi- Threading:
נריץ את התוכנית על מספר תהליכונים במקביל כדי לנצל זמן. ככה נוכל לצבוע יותר פיקסלים בו זמנית והתהליך של יצירת התמונה יהיה הרבה יותר מהיר!

השינויים שביצענו –

הוספנו בCamera הוספנו שדה PixelManager מטיפוס המחלקה PixelManager שקיבלנו מהמרצה.
השדה עוזר לנהל את מספר הפקיסלים שכל תהליך אחראי עליהם.
בנוסף, הוספנו שדה threadsCount אשר יקבע כמה תהליכונים ירוצו במקביל.
משתנה זה נקבל כקלט מהמשתמש.

```
private PixelManager pixelManager;   3 usages
private int threadsCount;   3 usages
```

וכמובן פונקציות set כדי שנוכל להגדיר בבניית המצלמה בכמה תהליכים להשתמש.

```
/**
 * Sets the camera's number of threads.
 *
 * @param threadsCount The number of threads
 * @return The builder
 */
public Builder setThreadsCount(int threadsCount) {
    camera.threadsCount = threadsCount;
    return this;
}
```

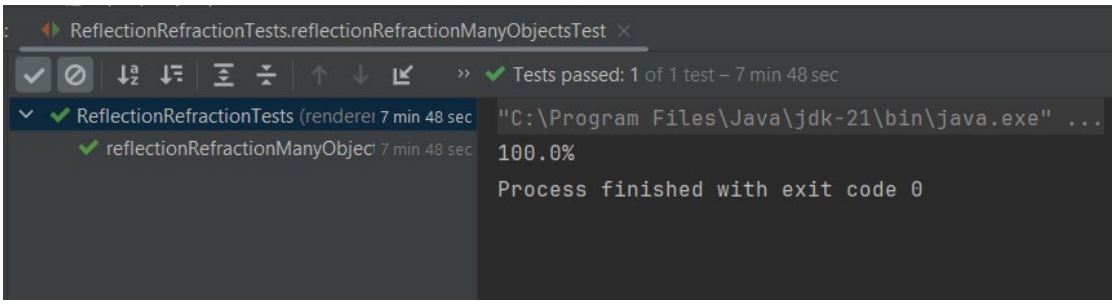
בפונקציה renderImage השתמשנו בשני הפרמטרים הללו על מנת ליצור את התהליכונים.
להלן הפונקציה renderImage בה התבצע השינויים:

```java
public Camera renderImage() {
    int nX = imageWriter.getNx();
    int nY = imageWriter.getNy();
    // Verify that nX and nY are not zero to avoid division by zero
    if (nY == 0 || nX == 0)
        throw new IllegalArgumentException("It is impossible to divide by 0");
    // Initialize the pixel manager
    pixelManager = new PixelManager(nY, nX, 0.1);
    // Check if the number of threads is 0
    if (threadsCount == 0) {
        for (int i = 0; i < nY; ++i)
            for (int j = 0; j < nX; ++j)
                castRay(nX, nY, j, i, antialiasingLevel);
    }
    else { // see further... option 2
        var threads = new LinkedList<Thread>(); // list of threads
        while (threadsCount-- > 0) // add appropriate number of threads
            threads.add(new Thread(() -> { // add a thread with its code
                PixelManager.Pixel pixel; // current pixel(row,col)
                // allocate pixel(row,col) in loop until there are no more pixels
                while ((pixel = pixelManager.nextPixel()) != null)
                    // cast ray through pixel (and color it - inside castRay)
                    castRay(nX, nY, pixel.col(), pixel.row(),antialiasingLevel);
            }));
        // start all the threads
        for (var thread : threads) thread.start();
        // wait until all the threads have finished
        try {
            for (var thread : threads) thread.join();
        } catch (InterruptedException ignore) {
        }
    }
    return this;
}
```

הפונקציה בשילוב הthreads -

במקום לעבד כל פיקסל בתמונה בצורה סינכרונית (מה שעלול להיות איטי), הפונקציה יוצרת מספר תהליכונים שיכולים לפעול במקביל. כל תהליכון מטפל בחלק מסוים מהתמונה ומבצע את פעולת הקריאה (castRay) על הפיקסלים שבתחום אחריותו. בכך היא מקטינה את זמן הרינדור הכולל על ידי פיזור עומס העבודה על פני מספר תהליכונים שפועלים בו-זמנית.

זמן הריצה אחרי שיפור זמן adaptive ואחרי שיפור זמן על ידי threads



בונוסים

- מימוש get Normal עבור גליל סופי
- מימוש חיתוכי קרן עם Polygon
- מימוש חיתוכי קרן עם Cylinder
- טרנספורמציית הזזה וסיבוב
- תאורת ספוט עם אלומה צרה יותר
- בניית תמונה עם המון גופים
- פתירת בעיית המרחק עם הצללה בדרך השנייה (מימוש FindIntersection עם פרמטר maxDistance)
- שיפור זמן ריצה על ידי threads

ביבליוגרפיה

- חומרי עזר מהמודל-
  מצגות הקורס התאורטי, מצגות המעבדה.

- הסבר על המצגות של הקורס ע"י הסטודנטית לאה חיים
  .055-971-8363

- תודה רבה לאליעזר על העזרה המרובה.