

# מיפוי פרויקט במبدأ להבנת תוכנה:

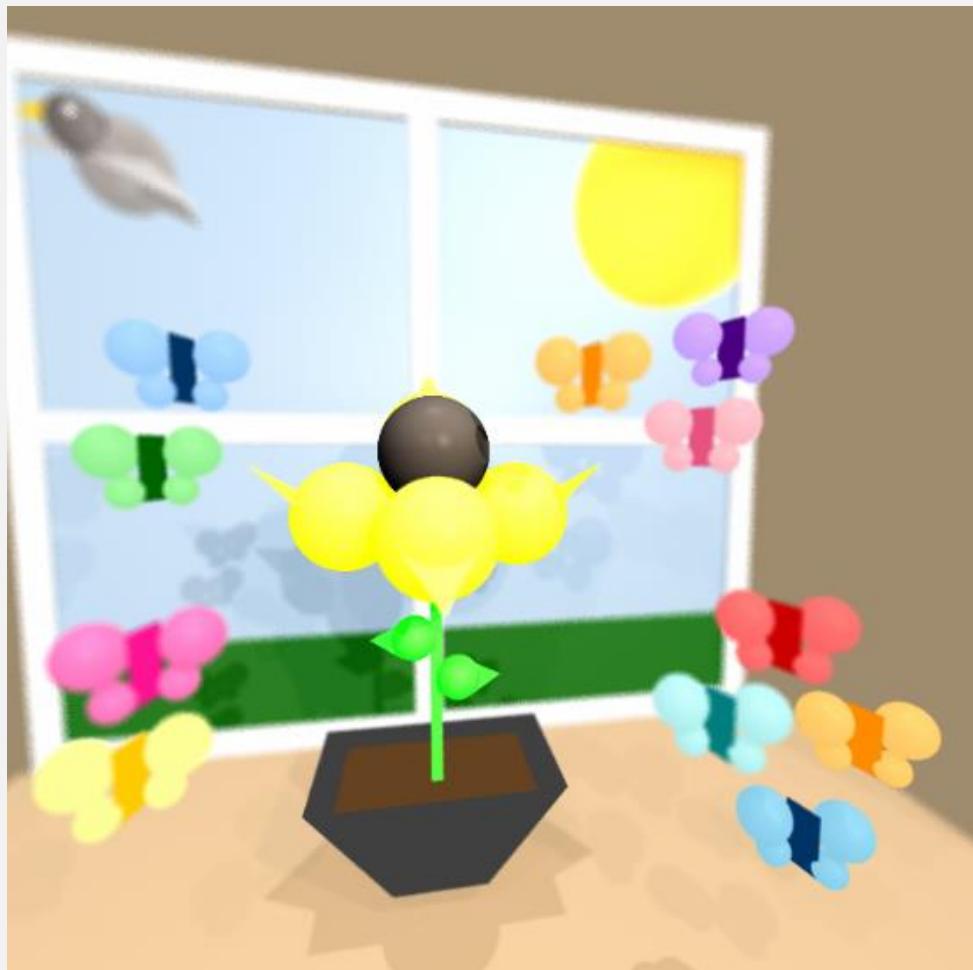
מגישות: טוביה טרייטיאק 326136041 ותליה שרגא 325439305.

סטודנטיות במכון טל (זוג מס' 222).

מרצה: אליעזר גנסבורגר.

שם הפרויקט: ISE5785\_6041\_9305 .

קישור לפרויקט בGITהאב: [https://github.com/tova-888111/ISE5785\\_6041\\_9305](https://github.com/tova-888111/ISE5785_6041_9305)



**תוכן עניינים:**

3.....	קדמה
4.....	הסצנה הסופית ותיאורה
5.....	עומק השדה
9.....	BVH
20.....	תהליכיונים
23.....	זמן ריצה
27.....	בונוסים
35.....	נימה אישית

## הקדמה:

בפרויקט זה אנו עוסקים בסביבת הפיתוח Idea IntelliJ. כתבו תוכנית בשפה **java** שיצרת תמונה תלת-ממדית באמצעות יריית קרניים (ray tracing) ל عبر כל פיקסל במשטח התמונה. עבור כל פיקסל מחושבת נקודת החיתוך בין הקרן לגופים בסצנה, ונקבעים צבעי הפיקסלים בהתאם למאפייני חיתוך הגוף.

הפרויקט מחולק למספר חビルות:

geometries – מכילה מחלקות עבור גופים גאומטריים תלת-ממדיים כולל חישוב חיתוכי קרניים והאצת ביצועים (BVH).

החבורה מכילה מחלקות עבור גופים: Plane, Polygon, Cylinder, Tube, Triangle, Sphere וכן מחלקת Geometries המכילה רשימת גופים גאומטריים.

ה גופים כוללים ממשים את המחלוקת המופשטת `intesectable` ואת המתודה המופשטת :

```
protected abstract List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance);
```

Intersection היא מחלוקת פנימית של Intersectable המכילה גאומטריה, נקודת וחומר הגאומטריה לשם החזרת הנתונים המדוייקים עבור נקודת החיתוך.

lighting – מכילה מחלקות המיצגות מקורות אור שונים בסצנה: אור נקודתי, אור ציוני, אור סיבתי ואור ממוקד.

primitives – מכילה מחלקות בסיס למתמטיקה גרפית כמו וקטור, נקודת, קרן, צבע וחומר.

renderer – מכילה מחלקות עבור רכיבי מנוע הרינדור כגון מצלמה, מחולל קרניים, כתיבת תמונה, ניהול פיקסלים. SimpleRayTracer, Camera, PixelManager, ImageWriter .

המחלקה Camera מכילה מחלוקת סטית פנימית בשם Builder בעלת שרשור מתודות set לשם יצירת מצלמה.

scene – מכילה את המחלוקת Scene המיצגת את הסצנה הgrafית הכוללת את שמה, רשימת הגופים בסצנה, רשימת גופי תאורה בסצנה, תאורה סיבתית וצבע רקע.

unitests – חבילת מחלוקות עבור בדיקות ייחודית לכל רכיב המערכת.

בפרויקט זה מימושנו את שיפור התמונה עומק השדה (Depth Of Field- DOF) וכן את שיפור ההאצה BVH עליהם נפרט בהמשך.

## **הסצנה הסופית ותיאורה:**

להלן מוצגת הסצנה הסופית עליה עבדנו בפרויקט:



מה בתמונה?

בתמונה מוצגת סצנה צבעונית בחדר מואר, שבמרכזה פרח תלת-ממדי בתוק אדנית, כשברקע נראה חלון גדול.

מחוץ לחלון זורחת שמש מוארת.

התמונה עצמה מלאה בהרבה מקורות אור, שיוצרות תאורה רכה וצל עדין על רצפת החדר וקירותיו. סבב הפרח מרחפים פרפרים צבעוניים, חלקם חדים וממקדים ואחרים מטושטשים – בהתאם לעומק השדה שייצר תחושת עומק ריאליסטית.

החלון עשוי זכוכית שקופה, דרכו ניתן לראות את השמש, הדשא וציפור עפה. השקיפות מאפשרת גם השתקפות עצינה של הפרפרים והחדר עצמו, המדגישה שימוש במופיעני חומר פיזיקליים כמו זכוכית ופלסטיק מבריק.

## עומק שדה (Depth of Field)

בפרויקט ישנו את אפקט עומק שדה, המדמה את הדרכ שבה מצלמה אמיתית מתמקדת באובייקטים הנמצאים למרחק מסוים בעוד שאובייקטים אחרים הקרובים יותר נראים מוטשטשים.

**המימוש:**

הגדנו 3 פרמטרים במחלקה המצלמה:  
 – קובע את גודל הצמצם – apertureRadius  
 – מגדר את המרחק אליו המצלמה מתפרקת – focalDistance  
 – מספר הקרןים הנשלחות לכל פיקסל שימושיפעות על איקות האפקט – dofRays

```
/** Aperture radius for depth of field effect (0 disables the effect) */
private double apertureRadius = 0.0; 5 usages
/** Distance from camera to the focus plane */
private double focalDistance = 0.0; 4 usages
/** Number of rays per pixel for depth of field */
private int dofRays = 1; 5 usages
```

כמו כן הוספנו מתודות חדשות במחלקה המצלמה:

:constructDofRays

המתודה מחזירה רשימה של קרןיהם עבור פיקסל בודד בתמונה לצורך ייצור יצרת אפקט עומק השדה. במהלך הפעולה:

בנייה קרטזית העוברת דרך הפיקסל הנבחר.

מחושבת נקודת הפקוס לפי הקרן המרכזית הנמצאת במרחב קבוע מהמכצלמה.

נשלחות מספר קרןיהם (dofRays) שמתחלקות מנוקודות שונות על הצמצם ומכוונות כלן לנקודת הפקוס. קרןיהם שפוגעות באובייקטים שנמצאים לפני או אחרי מישור הפקוס יגרמו לאזורים מוטשטשים – כך נוצר אפקט עומק השדה.

```
public List<Ray> constructDofRays(int nX, int nY, int j, int i) { 1 usage ± Tovat
    List<Ray> rays = new LinkedList<>();

    // Construct the center ray passing through the pixel (j, i)
    Ray centerRay = constructRay(nX, nY, j, i);
    // Calculate the focal point at the focal distance along the center ray
    Point focusPoint = centerRay.getPoint(focalDistance);

    for (int k = 0; k < dofRays; k++) {
        // Sample a random radius using sqrt for uniform distribution over the circle area
        double r = Math.sqrt(Math.random()) * apertureRadius;
        // Sample a random angle between 0 and 2*PI radians
        double theta = Math.random() * 2 * Math.PI;

        // Convert polar coordinates (r, theta) to Cartesian (x, y) on the aperture plane
        double x = r * Math.cos(theta);
        double y = r * Math.sin(theta);

        // Calculate the random point on the circular aperture around the camera position p0
        Point aperturePoint = p0
            .add(vRight.scale(x)) // move right by x units
            .add(vUp.scale(y));   // move up by y units

        // Calculate the direction from the aperture point to the focal point
        Vector direction = focusPoint.subtract(aperturePoint).normalize();

        // Create a new ray from the aperture point towards the focal point and add it to the list
        rays.add(new Ray(aperturePoint, direction));
    }

    return rays;
}
```

המתודה `castRay` אחראית על שליחת קרניים דרך פיקסלים בודדים במהלך תהליך הרינדור, והוא תומכת הן ב- `Ray Tracing` גליל וחן באופן עומק שדה (Depth of Field) כאשר `apertureRadius` שווה לאפס או שמספר הקרניים (`dofRays`) הוא 1, המתודה שולחת קרן אחת בלבד דרך הפיקסל – זהו מצב של צילום רגיל, ללא עומק שדה. כאשר `apertureRadius` גדול מאפס, המתודה יוצרת מספר קרניים עבור הפיקסל באמצעות (`constructDofRays`) כל קרן נשלחת מנוקודה שונה על הצמצם, ומכוונת לעבר נקודת פוקוס קבועה. הוצאה היא אפקט הצבע של קרן מחושב בנפרד (traceRay) ובבסיסו מושע של כל הצלבים. התוצאה היא אפקט של טשטוש באזוריים שאינם מצויים במישור הפוקוס – בדומה לאפקט שמתקיים בעדשות אופטיות בצלמות אמיתיות: עצמים למרחוק הפוקוס נראים חדים, בעוד עצמים קרובים או רחוקים מהם נראים מוטשטשים בהתאם לערך `apertureRadius`.

```
private void castRay( int column, int row){ 3 usages ▾ Tovat +1
    Color color;
    if (apertureRadius == 0 || dofRays == 1) {
        // Construct a ray through the pixel (column, row)
        Ray ray = constructRay(nX, nY, column, row);
        // Cast the ray and get the color at the intersection point
        color = rayTracer.traceRay(ray);
    } else {
        // Construct multiple rays for depth of field effect
        List<Ray> rays = constructDofRays(nX, nY, column, row);
        color = Color.BLACK;
        // Trace each ray and accumulate the color
        for (Ray ray : rays) {
            color = color.add(rayTracer.traceRay(ray));
        }
        // Average the color over the number of rays
        color = color.reduce(rays.size());
    }
    // Write the color to the image
    imageWriter.writePixel(column, row, color);
    pixelManager.pixelDone();
}
```

כיצד נפעיל את השיפור הזה ביצירת תמונה?  
באמצעות 3 מתודות `set` במחלקה הפנימית של Camera - Builder. המתחילה את 3 השדות השייכות לשיפור זה.  
זה: והן:

```
/**  
 * Sets the aperture radius for depth of field effect.  
 * @param apertureRadius the aperture radius (0 disables the effect)  
 * @return the Builder instance  
 */  
public Builder setAperture(double apertureRadius) { 6 usages ± Tovat  
    if (alignZero(apertureRadius) < 0) {  
        throw new IllegalArgumentException("Aperture radius must be non-negative");  
    }  
    camera.apertureRadius = apertureRadius;  
    return this;  
}
```

```
/**  
 * Sets the focal distance for the depth of field effect.  
 * @param focalDistance the distance to the focal plane  
 * @return the Builder instance  
 */  
public Builder setFocalDistance(double focalDistance) { 6 usages ± Tovat  
    if (alignZero(focalDistance) <= 0) {  
        throw new IllegalArgumentException("Focal distance must be positive");  
    }  
    camera.focalDistance = focalDistance;  
    return this;  
}
```

```
/**  
 * Sets the number of rays per pixel for depth of field effect.  
 * @param numRays the number of rays per pixel  
 * @return the Builder instance  
 */  
public Builder setDofRays(int numRays) { 6 usages ± Tovat  
    if (alignZero(numRays) <= 0) {  
        throw new IllegalArgumentException("Number of rays must be positive");  
    }  
    camera.dofRays = numRays;  
    return this;  
}
```

ביצירת התמונה הסופית שלנו השתמשנו בשיפור זה.  
התמונה לפני השיפור ואחריו:

אחרי:

לפני:



ניתן לראות שבתמונה לפני השיפור הציפור נראית חדה ומדויקת יותר וכן כלל הפרפרים, השמש והחלון. לעומת זאת אחרי השיפור מטושטשת יותר שכן היא רחוקה מהחלון. הפרפרים שקרובים או רחוקים יותר מהצלמה גם הם נראים מטושטשים יותר.

בנוסף לשיפור זה אף החליק מעט את כלל העקומות בתמונה.

## :BVH

בפרויקט יישמו את שיטת האצה מסוג BVH (Bounding Volume Hierarchy). מבנה נתונים היררכי לארגון גאומטריה תלת-ממדית, שמאפשר ביצוע חישובי חיתוך קרנויים (Ray Tracing) באופן יעיל ומהיר יותר.

מבנה ה- BV בני BV בנו עץ שבו כל צומת מכיל תיבת תוחמת מהסוג – (AABB- Axis Aligned Bounding Box) שזה תיבת ישרת-זווית המושרת עם הצירים. כל תיבת צזו מקיפה קבוצת גופים גאומטריים או תיבות תוחמות אחרות.

במהלך בדיקות החיתוך, הקורן נבדקת תחילה מול התיבות תוחמות בלבד. רק אם הקורן חותכת תיבת מסוימת – מתבצע חיתוך מול העצמים שבתוכה. כך נחסכת בדיקה ישירה מול כל אובייקט בסצנה, מה שמחית משמעותית את מספר החישובים ומשפר את ביצוע הרינדור.

המשמעות:

. geometries בטור החבילה BVHNode – AABB מחלקות חדשות.

מחלקה AABB:

מחלקה זו מייצגת תיבת תוחמת מישרת לצירים ומשמשת לבדיקה גאומטרית האם קורן יכולה לפגוע באובייקט כלשהו.

```
package geometries;

import primitives.*;

/**
 * The AABB (Axis-Aligned Bounding Box) class represents a 3D bounding box
 * defined by two points: the minimum and maximum corners.
 * It provides methods to check for intersection with rays and to compute
 * the union of two AABBs.
 * This class is used in various geometric computations,
 * such as collision detection and spatial partitioning.
 *
 * @author Tehila Shraga and Tova Tretiak
 */
public class AABB { 22 usages ± Tovat
    /** The minimum corner of the bounding box */
    public final Point min;
    /** The maximum corner of the bounding box */
    public final Point max;

    /**
     * Constructs an AABB with the specified minimum and maximum points.
     * The minimum point represents one corner of the box, and the maximum point
     * represents the opposite corner.
     *
     * @param min the minimum corner of the bounding box
     * @param max the maximum corner of the bounding box
     */
    public AABB(Point min, Point max) { 5 usages ± Tovat
        this.min = min;
        this.max = max;
    }
}
```

```

    /**
     * Checks if the AABB intersects with a given ray.
     * This method uses the slab method to determine if the ray intersects
     * the bounding box defined by the minimum and maximum points.
     *
     * @param ray the ray to check for intersection with the AABB
     * @return true if the ray intersects the AABB, false otherwise
     */
    public boolean intersects(Ray ray) { 4 usages ± Tovat
        // Get the origin and direction of the ray
        Point origin = ray.getHead();
        Vector dir = ray.getDirection();

        // Initialize the tMin and tMax values for the intersection test
        double tMin = Double.NEGATIVE_INFINITY;
        double tMax = Double.POSITIVE_INFINITY;

        // Convert the points to arrays for easier manipulation
        double[] originCoords = {origin.getX(), origin.getY(), origin.getZ()};
        double[] dirCoords = {dir.getX(), dir.getY(), dir.getZ()};
        double[] minCoords = {min.getX(), min.getY(), min.getZ()};
        double[] maxCoords = {max.getX(), max.getY(), max.getZ()};

        for (int i = 0; i < 3; i++) {
            if (Math.abs(dirCoords[i]) < 1e-9) { // Check if the direction component is nearly zero
                if (originCoords[i] < minCoords[i] || originCoords[i] > maxCoords[i])
                    return false;
            } else {
                double invDir = 1.0 / dirCoords[i]; // Inverse of the direction component
                double t1 = (minCoords[i] - originCoords[i]) * invDir;
                double t2 = (maxCoords[i] - originCoords[i]) * invDir;

                // Ensure t1 is the smaller value and t2 is the larger value
                if (t1 > t2) {
                    double tmp = t1; t1 = t2; t2 = tmp;
                }

                // Update tMin and tMax based on the intersection with this slab
                tMin = Math.max(tMin, t1);
                tMax = Math.min(tMax, t2);

                // If tMin exceeds tMax, there is no intersection
                if (tMin > tMax) return false;
            }
        }
        // Check if the ray intersects the AABB in the positive direction
        return tMax >= 0;
    }

    /**
     * Computes the union of two AABBs.
     * The union is defined as the smallest AABB that contains both input AABBs.
     *
     * @param a the first AABB
     * @param b the second AABB
     * @return a new AABB that represents the union of the two input AABBs
     */
    public static AABB union(AABB a, AABB b) { 4 usages ± Tovat
        // Ensure that the minimum and maximum points are correctly calculated
        Point min = new Point(
            Math.min(a.min.getX(), b.min.getX()),

```

```

        Math.min(a.min.getY(), b.min.getY()),
        Math.min(a.min.getZ(), b.min.getZ())
    );
    // Ensure that the maximum points are correctly calculated
    Point max = new Point(
        Math.max(a.max.getX(), b.max.getX()),
        Math.max(a.max.getY(), b.max.getY()),
        Math.max(a.max.getZ(), b.max.getZ())
    );
    return new AABB(min, max);
}
}

```

## המחלקה מכילה 2 מетодות עיקריות:

מетодת `intersects`:  
פונקציה זו בודקת האם קרן חותכת את התיבה התחומת המוצגת על ידי האובייקט `AABB`, זו בדיקה ראשונית שחווסכת זמן חישוב.

שלבים עיקריים:  
\* פירוק הקרן לרכיבים – הפונקציה מוציאה את נקודת הראש והכיוון של הקרן ומפרקת אותו לרכיבי  $Y$ ,  $X$  ו- $Z$ .  
\* בדיקה לכל ציר בנפרד – עבור כל אחד משלושת הצירים ( $Z$ ,  $Y$ ,  $X$ ):  
אם הקרן כמעט מקבילה לציר מסוים (כלומר כמעט אפס), נבדק אם הראש של הקרן נמצא בתחום גבולות התיבה בציר זהה, אם לא אין חיתוך.  
ואם יש כיוון לאורך הציר, מחשבים את הזמן  $t_1$  ו- $t_2$  שבו הקרן נכנסתו ויצאתו מהຕיבת באותו ציר. שומרים את טווחי הזמן המותרים לחיתוך ( $tMin$ ,  $tMax$ ).  
בדיקה של חיפוי בין הצירים – אם הטווחים לא חופפים ( $tMin > tMax$ ) – הקרן מפספסת את התיבה.  
תוצאה סופית – אם כל התנאים מתקיים, והקרן אכן נכנסת לתיבת ( $0 \leq tMax \leq tMin$ ) – הפונקציה מחזירה `true`.

```

public boolean intersects(Ray ray) { ↵ Tovat
    // Get the origin and direction of the ray
    Point origin = ray.getHead();
    Vector dir = ray.getDirection();

    // Initialize the tMin and tMax values for the intersection test
    double tMin = Double.NEGATIVE_INFINITY;
    double tMax = Double.POSITIVE_INFINITY;

    // Convert the points to arrays for easier manipulation
    double[] originCoords = {origin.getX(), origin.getY(), origin.getZ()};
    double[] dirCoords = {dir.getX(), dir.getY(), dir.getZ()};
    double[] minCoords = {min.getX(), min.getY(), min.getZ()};
    double[] maxCoords = {max.getX(), max.getY(), max.getZ()};

    for (int i = 0; i < 3; i++) {
        if (Math.abs(dirCoords[i]) < 1e-9) { // Check if the direction component is nearly zero
            if (originCoords[i] < minCoords[i] || originCoords[i] > maxCoords[i])
                return false;
        } else {
            double invDir = 1.0 / dirCoords[i]; // Inverse of the direction component
            double t1 = (minCoords[i] - originCoords[i]) * invDir;
            double t2 = (maxCoords[i] - originCoords[i]) * invDir;
            // Ensure t1 is the smaller value and t2 is the larger value
            if (t1 > t2) {
                double tmp = t1; t1 = t2; t2 = tmp;
            }

            // Update tMin and tMax based on the intersection with this slab
            tMin = Math.max(tMin, t1);
            tMax = Math.min(tMax, t2);

            // If tMin exceeds tMax, there is no intersection
            if (tMin > tMax) return false;
        }
    }
    // Check if the ray intersects the AABB in the positive direction
    return tMax >= 0;
}

```

מתודת חסון:  
פונקציה זו מקבלת שתי תיבות תוחמות ומחזירה תיבה חדשה שמכילה את שתיהן יחד כלומר, יוצרים תיבה תוחמתה משותפת עבור שני צומתים בעץ ה-BVH.  
לצורך בניית צומת חדש המכסה את התת צומתים שלו זהו שלב מרכזי בתהליך בניית עץ ה-BVH.

- שלבים עיקריים:
- \* מחשבת את הנקודה המינימלית החדשה.
- \* מחשבת את הנקודה המקסימלית החדשה.
- \* יוצרת תיבה חדשה (AABB) עם שתי הנקודות האלה.
- \* מחזירה את התיבה המאוחדת.

```
public static AABB union(AABB a, AABB b) { 4 usages ▾ Tovat
    // Ensure that the minimum and maximum points are correctly calculated
    Point min = new Point(
        Math.min(a.min.getX(), b.min.getX()),
        Math.min(a.min.getY(), b.min.getY()),
        Math.min(a.min.getZ(), b.min.getZ())
    );
    // Ensure that the maximum points are correctly calculated
    Point max = new Point(
        Math.max(a.max.getX(), b.max.getX()),
        Math.max(a.max.getY(), b.max.getY()),
        Math.max(a.max.getZ(), b.max.getZ())
    );
    return new AABB(min, max);
}
```

הוספנו לכל גוף גאומטרי מתודה בשם `getBoundingBox` שמחזירה את תיבת ה-BVH התוחמת המתאימה לו.  
כך כל גוף יודע לספק את התיבה התוחמתה שלו, המגדירה את גבולותיו במרחב התלת-ממדי.

מחלקה זו מייצגת צומת בעץ ה-BVH.  
מחלקה `BVHNode`:

```

package geometries;

import primitives.Ray;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 * The BVHNode class represents a node in a Bounding Volume Hierarchy (BVH).
 * It is used for spatial acceleration of ray intersection tests.
 * Each node contains a bounding box and references to left and right child nodes.
 * The BVH is built from a list of intersectable geometries.
 *
 * @author Tehila Shraga and Tova Tretiak
 */
public class BVHNode extends Intersectable { 3 usages ± Tovat
    /** The bounding box of this BVH node */
    private final AABB box; 5 usages
    /** The left child node, which may be another BVHNode or an intersectable geometry */
    private final Intersectable left; 8 usages
    /** The right child node, which may be another BVHNode or an intersectable geometry */
    private final Intersectable right; 7 usages

    /**
     * Constructs a BVHNode from a list of intersectable geometries.
     * If the list contains one geometry, it becomes a leaf node.
     * If it contains two geometries, they become the left and right children.
     * For more than two geometries, the list is split into two halves recursively.
     *
     * @param geometries the list of intersectable geometries to build the BVH from
     */
}

public BVHNode(List<Intersectable> geometries) { 3 usages ± Tovat
    // Ensure the list is not empty
    if (geometries.size() == 1) {
        // If there's only one geometry, it becomes a leaf node
        left = geometries.get(0);
        right = null;
        box = left.getBoundingBox();
        // If there's only one geometry, it becomes a leaf node
    } else if (geometries.size() == 2) {
        left = geometries.get(0);
        right = geometries.get(1);
        box = AABB.union(left.getBoundingBox(), right.getBoundingBox());
        // If there are two geometries, they become the left and right children
    } else {
        // More than two geometries, we need to split them
        AABB totalBox = geometries.get(0).getBoundingBox();
        for (int i = 1; i < geometries.size(); i++) {
            totalBox = AABB.union(totalBox, geometries.get(i).getBoundingBox());
        }

        // Sort the geometries based on their bounding box extents
        double xExtent = totalBox.max.getX() - totalBox.min.getX();
        double yExtent = totalBox.max.getY() - totalBox.min.getY();
        double zExtent = totalBox.max.getZ() - totalBox.min.getZ();

        // Sort geometries based on the largest extent
        if (xExtent >= yExtent && xExtent >= zExtent) {
            geometries.sort(Comparator.comparingDouble( Intersectable g -> g.getBoundingBox().min.getX()));
        } else if (yExtent >= zExtent) {
            geometries.sort(Comparator.comparingDouble( Intersectable g -> g.getBoundingBox().min.getY()));
        }
    }
}

```

```

        geometries.sort(Comparator.comparingDouble( Interseable g -> g.getBoundingBox().min.getZ()));
    }

    // Split the list into two halves
    int mid = geometries.size() / 2;
    // Create left and right child nodes recursively
    left = new BVHNode(geometries.subList(0, mid));
    // Create right child node with the remaining geometries
    right = new BVHNode(geometries.subList(mid, geometries.size()));
    // Calculate the bounding box for this node
    box = AABB.union(left.getBoundingBox(), right.getBoundingBox());
}
}

/**
 * Finds the intersection points of a ray with the BVH node.
 * This method checks if the ray intersects the bounding box of this node.
 * If it does, it recursively checks the left and right child nodes for intersections.
 *
 * @param ray the ray to check for intersections
 * @param maxDistance the maximum distance from the ray's origin to consider for intersections
 * @return a list of Intersection objects, or {@code null} if there are no intersections
 */
@Override ̄ Tovat
protected List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance) {
    if (!box.intersects(ray)) return null;

    // Check if the ray intersects the bounding box of this node
    List<Intersection> intersections = new ArrayList<>();
    double currentMaxDistance = maxDistance;

    // Check intersections in the left child node
    if (left != null) {
        List<Intersection> leftHits = left.calculateIntersections(ray, currentMaxDistance);
        if (leftHits != null) {
            intersections.addAll(leftHits);
            // Update the current maximum distance based on the closest intersection found in the left subtree
            for (Intersection hit : leftHits) {
                currentMaxDistance = Math.min(currentMaxDistance, hit.point.distance(ray.getHead()));
            }
        }
    }

    // Check intersections in the right child node
    if (right != null) {
        List<Intersection> rightHits = right.calculateIntersections(ray, currentMaxDistance);
        if (rightHits != null) {
            intersections.addAll(rightHits);
        }
    }

    // If no intersections were found, return null
    return intersections.isEmpty() ? null : intersections;
}
}

```

```

    /**
     * Returns the axis-aligned bounding box (AABB) of this BVH node.
     * The bounding box is defined by the minimum and maximum coordinates
     * of the geometries contained in this node.
     *
     * @return the bounding box of this BVH node
     */
    @Override 12 usages ± Tovat
    public AABB getBoundingBox() {
        return box;
    }
}

```

הע' נבנה בקורס בית מתוך רשיית גומטריות:  
אם קיימת גומטריה אחת – היא הופכת לעלה בע'.

אם קיימות שתי גומטריות – הן מוקפות לשני תת-ענפים (ימין ושמאל).

אם יש יותר – הרשימה מפוצלת לפי הציר שבו הפיזור הכ' גדול (Z/Y/X) והצמתים נוצרים רקorsiיט.

```

public BVHNode(List<Intersectable> geometries) { 3 usages ± Tovat
    // Ensure the list is not empty
    if (geometries.size() == 1) {
        // If there's only one geometry, it becomes a leaf node
        left = geometries.get(0);
        right = null;
        box = left.getBoundingBox();
    } else if (geometries.size() == 2) {
        left = geometries.get(0);
        right = geometries.get(1);
        box = AABB.union(left.getBoundingBox(), right.getBoundingBox());
    } // If there are two geometries, they become the left and right children
    else {
        // More than two geometries, we need to split them
        AABB totalBox = geometries.get(0).getBoundingBox();
        for (int i = 1; i < geometries.size(); i++) {
            totalBox = AABB.union(totalBox, geometries.get(i).getBoundingBox());
        }

        // Sort the geometries based on their bounding box extents
        double xExtent = totalBox.max.getX() - totalBox.min.getX();
        double yExtent = totalBox.max.getY() - totalBox.min.getY();
        double zExtent = totalBox.max.getZ() - totalBox.min.getZ();
        // Sort geometries based on the largest extent
        if (xExtent > yExtent && xExtent > zExtent) {
            geometries.sort(Comparator.comparingDouble(Intersectable g -> g.getBoundingBox().min.getX()));
        } else if (yExtent >= zExtent) {
            geometries.sort(Comparator.comparingDouble(Intersectable g -> g.getBoundingBox().min.getY()));
        } else {
            geometries.sort(Comparator.comparingDouble(Intersectable g -> g.getBoundingBox().min.getZ()));
        }

        // Split the list into two halves
        int mid = geometries.size() / 2;
        // Create left and right child nodes recursively
        left = new BVHNode(geometries.subList(0, mid));
        // Create right child node with the remaining geometries
        right = new BVHNode(geometries.subList(mid, geometries.size()));
        // Calculate the bounding box for this node
        box = AABB.union(left.getBoundingBox(), right.getBoundingBox());
    }
}

```

:calculateIntersectionsHelper ()

פונקציה זו אחראית לבדוק האם الكرن נטונה פוגעת בעצמים הגאומטריים שנמצאים בתחום עץ ה-BVH, תוך שימוש בתיבות התוחמות לצורך האצה ובקרה. היא משתמשת את כל נקודות החיתוך של הקרן עם העצמים שנמצאים תחת הצומת הנוכחי בעץ BVH, אך רק אם הקרן פוגעת בתיבה התוחמת של הצומת ורק נמנעים מחישובי חיתוך מיוחדים עם עצמים שהקרן יכול לא יכול לפגוע בהם.

היא בודקת האם הקרן פוגעת בתיבה התוחמת של הצומת בעזרת הפונקציה, אם אין חיתוך עם התיבה – הקרן גם לא יכולה לפגוע בעצמים שבתוכה ולכן הפונקציה מחזיר `null`.

בדיקה רקורסיבית של תמי עצים:

אם יש חיתוך עם התיבה: הקרן נשלחת לבדיקה בצומת השמאלי (`left`) – אם יש פגיעות, הן נאספות לרשיימה. מחושבת המרחק הקצר ביותר מהקרן לנקודות פגעה שמאלית – כדי להשתמש בו כגבול חיתוך עבור הצומת הימני (ולמנוע פגיעות מיותרות רוחקות יותר).

הקרן נבדקת גם בצומת הימני (`right`), אך רק עד למרחק הקצר שנמצא קודם לכן. אם נמצא נקודות חיתוך – הן מוחזרות. אם לא – מוחזר `null`.

```
protected List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance) {
    if (!box.intersects(ray)) return null;

    // Check if the ray intersects the bounding box of this node
    List<Intersection> intersections = new ArrayList<>();
    double currentMaxDistance = maxDistance;

    // Check intersections in the left child node
    if (left != null) {
        List<Intersection> leftHits = left.calculateIntersections(ray, currentMaxDistance);
        if (leftHits != null) {
            intersections.addAll(leftHits);
            // Update the current maximum distance based on the closest intersection found in the left subtree
            for (Intersection hit : leftHits) {
                currentMaxDistance = Math.min(currentMaxDistance, hit.point.distance(ray.getHead()));
            }
        }
    }

    // Check intersections in the right child node
    if (right != null) {
        List<Intersection> rightHits = right.calculateIntersections(ray, currentMaxDistance);
        if (rightHits != null) {
            intersections.addAll(rightHits);
        }
    }

    // If no intersections were found, return null
    return intersections.isEmpty() ? null : intersections;
}
```

איך כל זה מתחבר ליצירת תמונה? המצלמה מכילה אובייקט `rayTracer` המכיל סצנה. באובייקט הסצנה קיימות רשימת גאומטריות, של המחלקה `Geometries`.

במחלקה זו קיימות השדות: `accelerationStructure` – מציין את שורש עץ ה-BVH שנבנה. למעשה זהו מבנה האצה עצמו. `useBVH` – קבוע האם להשתמש במנגנון האצה מסוג BVH. אם הוא – `true` הקרן יי"ז בדקו מול עץ אחר – מול כל

הגאומטריות ישירות.

```
/** The acceleration structure for ray intersection, if BVH is used */
private Intersectable accelerationStructure = null; 5 usages
/** Whether to use BVH for acceleration */
private boolean useBVH; 5 usages
```

ובנוסף היא מכילה את המתודות:  
 buildBVH() - בונה את מבנה עץ ה-BVH מתוך רשימת הגאומטריות. כל קראיה למתודה זו יוצרת עץ חדש מהגיאומטריות הקיימות על בסיס תיבות (AABB).

```
/**
 * Builds the BVH tree from current geometries.
 */
public void buildBVH() { this.accelerationStructure = new BVHNode(geometries); }
```

setUseBVH(boolean useBVH) - מאפשר להפעיל או לכבות את השימוש ב-BVH, בעת הפעלה – בונה את עץ BVH אם הוא עדין לא נבנה.

```
/**
 * Sets whether to use BVH for acceleration.
 *
 * @param useBVH true to enable BVH, false to disable
 * @return this Geometries instance for method chaining
 */
public Geometries setUseBVH(boolean useBVH) { 1 usage ▾ Tovat
    this.useBVH = useBVH;

    if (useBVH) {
        if (accelerationStructure == null) {
            buildBVH();
        }
    } else {
        this.accelerationStructure = null;
    }
    return this;
}
```

buildBVH() – מוסיף גיאומטריות לאויפר. אם useBVH פעיל – מפעילה את add(Intersectable... geometries)

כדי לעדכן את עץ ה-BVH עם הגיאומטריות החדשנות.

```
/**
 * Adds one or more intersectable geometries to the collection.
 * @param geometries the geometries to be added
 */
public void add(Intersectable... geometries) { ± Tovat
    this.geometries.addAll(List.of(geometries));
    if (useBVH) {
        // Rebuild the BVH structure whenever geometries are added
        buildBVH();
    }
}

/**
```

לגיאומטריות: אם BVH מופעל – הערך נבדקת מול מבנה העץ (accelerationStructure) אחרת – נבדקת ישירות מול כל גיאומטריה.

```
@Override ± Tovat
protected List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance) {
    // If BVH is used, delegate to the acceleration structure
    if (useBVH && accelerationStructure != null) {
        return accelerationStructure.calculateIntersections(ray, maxDistance);
    }

    // If BVH is not used, iterate through all geometries directly
    List<Intersection> intersections = null;
    double currentMaxDistance = maxDistance;

    // Iterate through all geometries and calculate intersections
    for (Intersectable geometry : geometries) {
        List<Intersection> tempIntersections = geometry.calculateIntersections(ray, currentMaxDistance);
        if (tempIntersections != null) {
            if (intersections == null)
                intersections = new LinkedList<>();
            intersections.addAll(tempIntersections);

            // Update the current maximum distance based on the closest intersection found
            for (Intersection hit : tempIntersections) {
                double distance = hit.point.distance(ray.getHead());
                currentMaxDistance = Math.min(currentMaxDistance, distance);
            }
        }
    }
    // If no intersections were found, return null
    return intersections;
}
```

איך נפעיל את BVH?

בתוך Builder במחלקה המצלמה, קיימת מתודת שרשור מתאימה:

```
/**  
 * Enables the use of BVH (Bounding Volume Hierarchy) for acceleration.  
 * If enabled, the camera will use BVH for faster ray intersection tests.  
 *  
 * @return the Builder instance  
 */  
public Builder enableBVH() { 5 usages ▾ Tovat  
    camera.useBVH = true;  
    camera.rayTracer.setScene().setUseBVH(true);  
    return this;  
}
```

מתודה זו מפעילה את המתודה במחלקה סצנה שלהן, והיא מזמנת את המתודה (`setUseBVH()`) במחלקה הגאומטריות:

```
/**  
 * Sets whether to use BVH (Bounding Volume Hierarchy) for acceleration in the scene's geometries.  
 * @param useBVH Whether to use BVH.  
 * @return The current Scene object for method chaining.  
 */  
public Scene setUseBVH(boolean useBVH) { 1 usage ▾ Tovat  
    this.geometries.setUseBVH(useBVH);  
    return this;  
}
```

## תהליכיונים:

בנוסף לצורכי זירוז זמן יצרת תמונה בפרויקט מימוש בתחום תהליכיונים.  
לצורך כך קיימת המחלקה `PixelManager` בחיבור `render` שנועדה לנהל שימוש בהליךונים, וכן מספר מתודות ושדות חדשים במחלקה המצלמה.

במצלמה הוספנו שדות:

```
/** Amount of threads to use for rendering image by the camera */
private int threadsCount = 0; 6 usages
/**
 * Amount of threads to spare for Java VM threads:<br>
 * Spare threads if trying to use all the cores
 */
private static final int SPARE_THREADS = 2; 1 usage
/**
 * Debug print interval in seconds (for progress percentage)<br>
 * if it is zero - there is no progress output
 */
private double printInterval = 0; 4 usages
/**
 * Pixel manager for supporting:
 * <ul>
 * <li>multi-threading</li>
 * <li>debug print of progress percentage in Console window/tab</li>
 * </ul>
 */
private PixelManager pixelManager; 5 usages
/** Aperture radius for depth of field effect (0 disables the effect) */
```

הוספנו מетодות `set` ב-`Builder`:

```
/*
 * Set multi-threading <br>
 * Parameter value meaning:
 * <ul>
 * <li>-2 - number of threads is number of logical processors less 2</li>
 * <li>-1 - stream processing parallelization (implicit multi-threading) is used</li>
 * <li>0 - multi-threading is not activated</li>
 * <li>1 and more - literally number of threads</li>
 * </ul>
 * @param threads number of threads
 * @return builder object itself
 */
public Builder setMultithreading(int threads) { 5 usages ± Tehila Shraga
    if (threads < -3)
        throw new IllegalArgumentException("Multithreading parameter must be -2 or higher");
    if (threads == -2) {
        int cores = Runtime.getRuntime().availableProcessors() - SPARE_THREADS;
        camera.threadsCount = cores <= 2 ? 1 : cores;
    } else
        camera.threadsCount = threads;
    return this;
}
```

```
/**
 * Set debug printing interval. If it's zero - there won't be printing at all
 * @param interval printing interval in %
 * @return builder object itself
 */
public Builder setDebugPrint(double interval) { no usages  ± Tehila Shraga
    if (interval < 0) throw new IllegalArgumentException("interval parameter must be non-negative");
    camera.printInterval = interval;
    return this;
}
```

במצלמה הוספנו מתוודות:

```
/**
 * Retrieves the image writer used for rendering the scene.
 * Iterates through each pixel in the image and casts a ray through it.
 * @return the image writer
 */
public Camera renderImage(){  ± Tehila Shraga +1
    pixelManager = new PixelManager(nY, nX, printInterval);
    return switch (threadsCount) {
        case 0 -> renderImageNoThreads();
        case -1 -> renderImageStream();
        default -> renderImageRawThreads();
    };
}
```

```
/**
 * Render image using multi-threading by parallel streaming
 * @return the camera object itself
 */
private Camera renderImageStream() { 1 usage  ± Tovat +1
    IntStream.range(0, nY).parallel()
        .forEach( int i -> {
            for (int j = 0; j < nX; j++) {
                castRay(j, i);
            }
        });
    return this;
}
```

```

    /**
     * Render image without multi-threading
     * @return the camera object itself
     */
    private Camera renderImageNoThreads() { 1 usage ± Tehila Shraga
        for (int i = 0; i < nY; ++i)
            for (int j = 0; j < nX; ++j)
                castRay(j, i);
        return this;
    }

    /**
     * Render image using multi-threading by creating and running raw threads
     * @return the camera object itself
     */
    private Camera renderImageRawThreads() { 1 usage ± Tehila Shraga
        var threads = new LinkedList<Thread>();
        while (threadsCount-- > 0)
            threads.add(new Thread(() -> {
                PixelManager.Pixel pixel;
                while ((pixel = pixelManager.nextPixel()) != null)
                    castRay(pixel.col(), pixel.row());
            }));
        for (var thread : threads) thread.start();
        try {
            for (var thread : threads) thread.join();
        } catch (InterruptedException ignored) {}
        return this;
    }
}

```

ואת הפקודה הבאה הוספנו לו:  
**pixelManager.pixelDone();**

במימוש זה, רינדור התמונה מתבצע באמצעות מספר שיטות תהליכי (Multithreading) בהתאם לערך השדה `.threadsCount`

אם הוא 0 – הרינדור מתבצע בלולאה סידרתית, אם הוא 1 – משתמשים ב- Java Streams (parallel למקביליות אוטומטית, ואם הוא גדול מ-0 – נוצרים תהליכי גולמיים (Raw Threads) ידנית, כשל תהליכי שלף פיקסל דרך `pixelManager` ומבצע עליי יריית קרן.

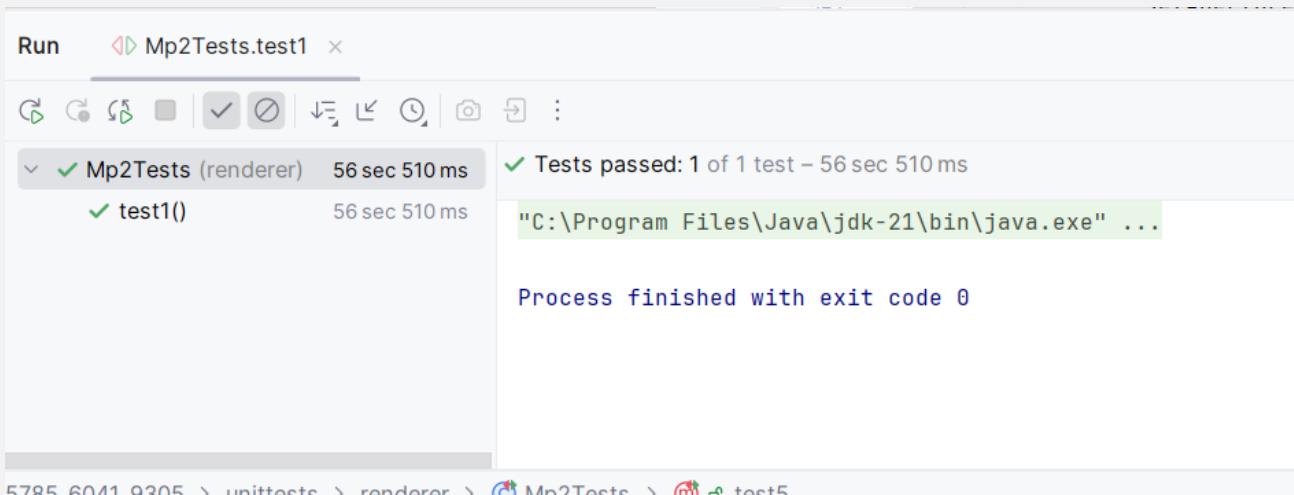
השדה `threadsCount` מגדר את כמות התהליכי, בעוד `SPARE_THREADS` קובע כמה ליבוט לשאר חופשיות עבור מערכת הפעלה.

`printInterval` מאפשר הדפסת התקדמות באחזים, ו- `pixelManager` מנהל את חלוקת הפיקסלים בין התהליכיים באופן מבוקר. המתודות `setDebugPrint` ו- `setMultithreading` משמשות להגדרת אופן הרינדור והדפסת הדיבאג בהתאם.

### זמן ריצה:

את התמונה הסופית שלנו הרצנו 4 פעמים כדי להשוות זמן ריצה.

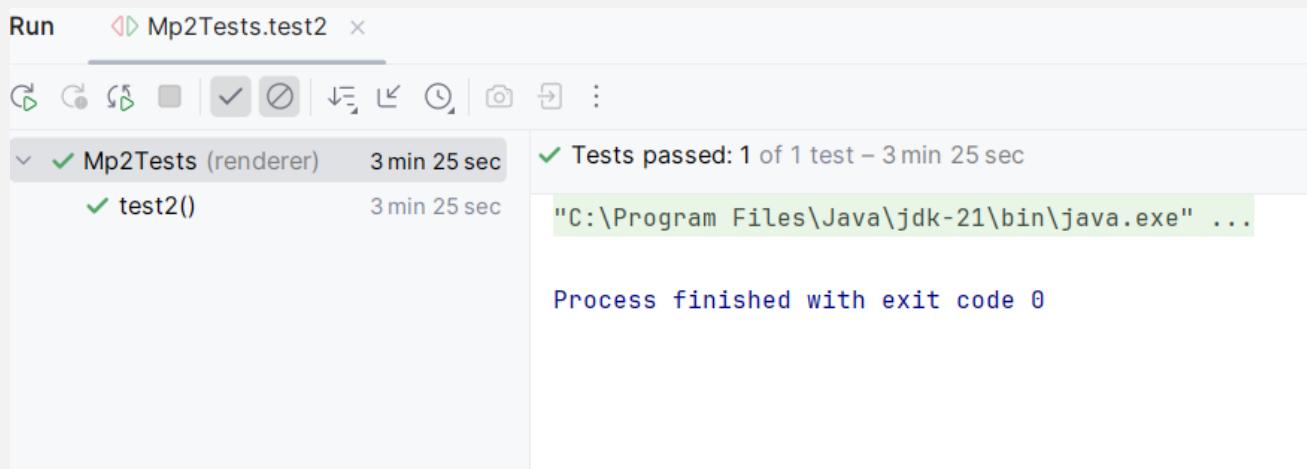
```
/*
 * Test method that renders a simple flower scene with a camera.
 * It uses multithreading to speed up the rendering process.
 * It uses BVH for optimization.
 */
@Test + Tehila Shraga +1
void test1() {//Time: 56 sec 510 ms
    // Create a camera with a specific location and direction
    Camera cam1 = Camera.getBuilder()
        .setLocation(new Point( x: -1, y: 0.5, z: 5))
        .setDirection(new Point( x: 0, y: 0, z: -1))
        .setVpSize( width: 20, height: 20)
        .setVpDistance(10)
        .setResolution( nX: 700, nY: 700)
        .setRayTracer(scene, RayTracerType.SIMPLE)
        .setDofRays(50)
        .setAperture(0.5)
        .setFocalDistance(23)
        .setMultithreading(-2) // Use all available cores for multithreading
        .enableBVH()// Enable BVH for optimization
        .build();
    cam1.renderImage().writeToImage( filePath: "FinalPicture1");
}
```



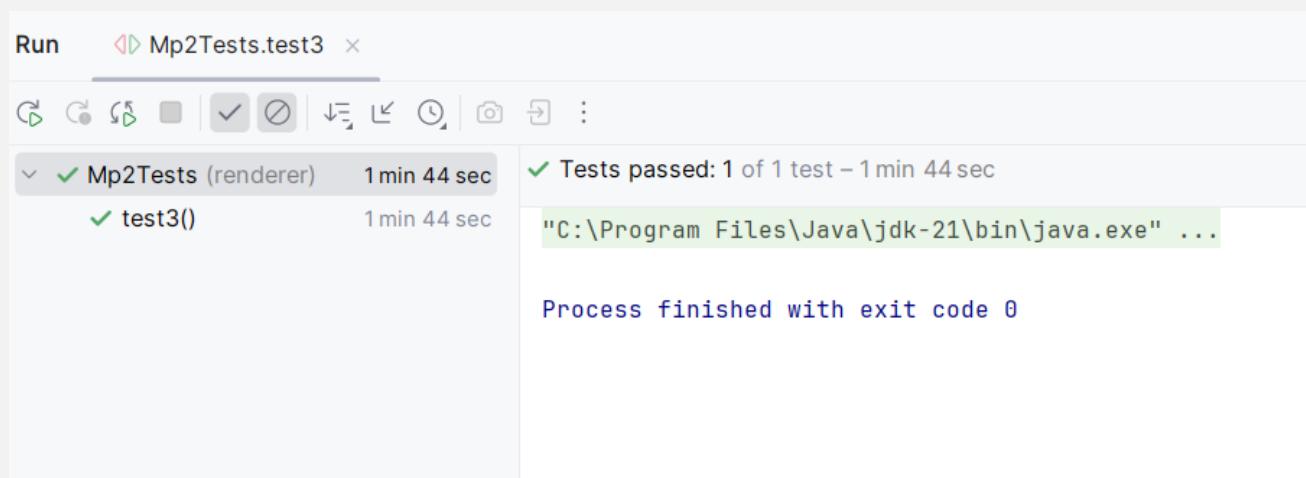
```

/**
 * Test method that renders a simple flower scene with a camera.
 * It doesn't use multithreading to speed up the rendering process.
 * It uses BVH for optimization.
 */
@Test
void test2() {//Time: 3 min 25 sec
    // Create a camera with a specific location and direction
    Camera cam1 = Camera.getBuilder()
        .setLocation(new Point( x: -1, y: 0.5, z: 5))
        .setDirection(new Point( x: 0, y: 0, z: -1))
        .setVpSize( width: 20, height: 20)
        .setVpDistance(10)
        .setResolution( nX: 700, nY: 700)
        .setRayTracer(scene, RayTracerType.SIMPLE)
        .setDofRays(50)
        .setAperture(0.5)
        .setFocalDistance(23)
        .enableBVH()// Enable BVH for optimization
        .build();
    cam1.renderImage().writeToImage( filePath: "FinalPicture2");
}

```



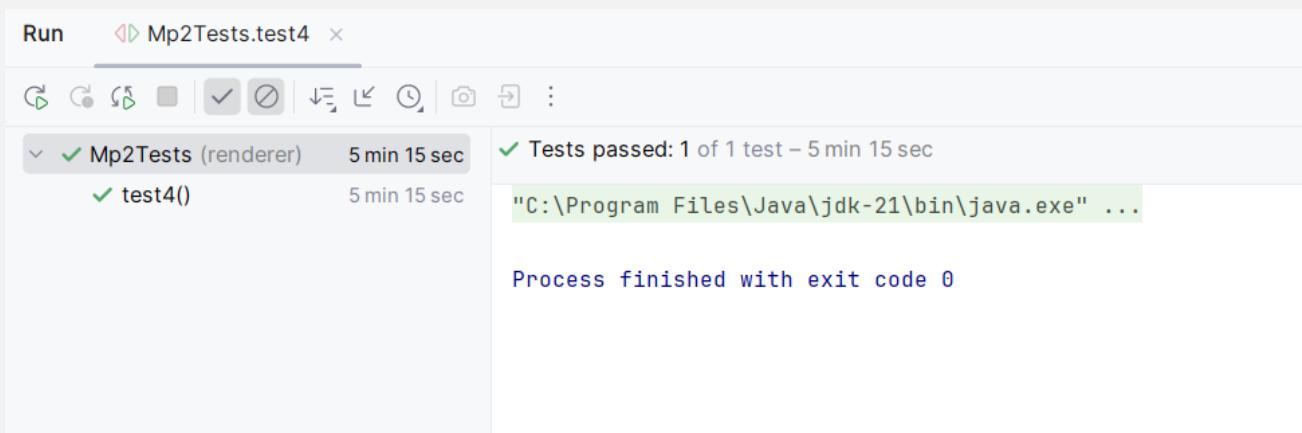
```
/*
 * Test method that renders a simple flower scene with a camera.
 * It uses multithreading to speed up the rendering process.
 * It doesn't use BVH for optimization.
 */
@Test  ± Tovat
void test3() { //Time: 1 min 44 sec
    // Create a camera with a specific location and direction
    Camera cam1 = Camera.getBuilder()
        .setLocation(new Point(x: -1, y: 0.5, z: 5))
        .setDirection(new Point(x: 0, y: 0, z: -1))
        .setVpSize(width: 20, height: 20)
        .setVpDistance(10)
        .setResolution(nX: 700, nY: 700)
        .setRayTracer(scene, RayTracerType.SIMPLE)
        .setDofRays(50)
        .setAperture(0.5)
        .setFocalDistance(23)
        .setMultithreading(-2) // Use all available cores for multithreading
        .build();
    cam1.renderImage().writeToImage(filePath: "FinalPicture3");
}
```



```

/**
 * Test method that renders a simple flower scene with a camera.
 * It doesn't use multithreading to speed up the rendering process.
 * It doesn't use BVH for optimization.
 */
@Test // Tovat
void test4() //Time: 5 min 15 sec
    // Create a camera with a specific location and direction
    Camera cam1 = Camera.getBuilder()
        .setLocation(new Point(x: -1, y: 0.5, z: 5))
        .setDirection(new Point(x: 0, y: 0, z: -1))
        .setVpSize(width: 20, height: 20)
        .setVpDistance(10)
        .setResolution(nX: 700, nY: 700)
        .setRayTracer(scene, RayTracerType.SIMPLE)
        .setDofRays(50)
        .setAperture(0.5)
        .setFocalDistance(23)
        .build();
    cam1.renderImage().writeToImage(filePath: "FinalPicture4");
}

```



## בונוסים:

אתאר מהם הבונוסים אותם בחרנו למש בפרויקט:

1- מימוש `getNormal(Point point)` עבור גליל סופי.

```

@Override  ✎ Tovat
public Vector getNormal(Point point) {
    // Base center
    Point p0 = axis.getHead();
    // Axis direction
    Vector vector = axis.getDirection();

    // Check if the point is on the bottom base center
    if (p0.equals(point)) {
        return vector.scale( scalar: -1);
    }

    // Check if the point is on the top base center
    if (axis.getPoint(height).equals(point)) {
        return vector;
    }

    // Projection of the point onto the axis
    double t = axis.getDirection().dotProduct(point.subtract(axis.getHead()));
    // Calculate the projection point on the axis
    // Point is on the bottom base center
    if (isZero(t)) {
        return vector.scale( scalar: -1);
        // Point is on the top base center
    } else if (isZero( number: t - height)) {
        return vector;
    } else {
        // Point is on the lateral surface
        return point.subtract(axis.getPoint(t)).normalize();
    }
}

```

## -2 - חישוב חיתוכים בין קרכן למצולע.

```

@Override 1 override ± Tehila Shraga +
public List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance) {
    // Step 1: Intersect the ray with the polygon's plane
    List<Intersection> planeIntersections = plane.calculateIntersections(ray, maxDistance);
    if (planeIntersections == null) {
        return null; // No intersection with the plane
    }

    Point intersectionPoint = planeIntersections.getFirst().point;
    Point p0 = ray.getHead();

    // Exclude the ray's origin point (head) if it's exactly on the polygon
    if (intersectionPoint.equals(p0)) {
        return null;
    }

    Vector v = intersectionPoint.subtract(p0);

    // Step 2: Check if the intersection point is inside the polygon
    int size = vertices.size();
    boolean isPositive = false;

    for (int i = 0; i < size; i++) {
        Point vi = vertices.get(i);
        Point vi1 = vertices.get((i + 1) % size);

        Vector edge1 = vi.subtract(p0);
        Vector edge2 = vi1.subtract(p0);

        Vector cross = edge1.crossProduct(edge2);
        double dot = alignZero(cross.dotProduct(v));
    }
}

```

```

    // On the first iteration, determine the direction (positive/negative)
    if (i == 0) {
        if (isZero(dot)) return null; // Point is on the edge
        isPositive = dot > 0;
    } else {
        // If the sign differs from the initial sign - point is outside
        if (isZero(dot) || (dot > 0) != isPositive) return null;
    }
}

// If the point is inside, return it
return List.of(new Intersection( geometry: this, intersectionPoint));
}

```

## -3 - חישוב חיתוכים בין קָרֵן לצינור אַנְסּוֹפִי .

```

@Override 1 override  ± Tovat
public List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance) {
    Point p = ray.getHead(); // Ray's origin point
    Vector v = ray.getDirection(); // Ray's direction vector
    Point p0 = axis.getHead(); // Tube's axis origin
    Vector v0 = axis.getDirection(); // Tube's axis direction

    // Compute the vector from the ray's origin to the tube's axis origin
    Vector deltaP;
    try {
        deltaP = p.subtract(p0);
    } catch (IllegalArgumentException e) {
        // Special case: ray starts at the axis head
        if (isZero(v.dotProduct(v0))) {
            Point intersection = ray.getPoint(radius);
            if (intersection.distance(p) < maxDistance) {
                return List.of(new Intersection( geometry: this, intersection));
            }
            return null;
        }
        return null;
    }

    // Check if the ray is perpendicular to the tube's axis and starts on the axis
    try {
        Vector cross = deltaP.crossProduct(v0);
        if (isZero(cross.length())) {
            if (isZero(v.dotProduct(v0))) {
                Point intersection = ray.getPoint(radius);
                if (intersection.distance(p) < maxDistance) {
                    return List.of(new Intersection( geometry: this, intersection));
                }
            }
            return null;
        }
    } catch (IllegalArgumentException e) {
        // Special case: deltaP is parallel to v0
        if (isZero(v.dotProduct(v0))) {
            Point intersection = ray.getPoint(radius);
            if (intersection.distance(p) < maxDistance) {
                return List.of(new Intersection( geometry: this, intersection));
            }
        }
        return null;
    }

    // If the ray is almost parallel to the tube's axis, no intersection
    if (Math.abs(v0.dotProduct(v)) > v0.length() * v.length() - 1e-10) {
        return null;
    }

    // Compute the quadratic equation coefficients for intersection
    Vector vCrossV0 = v.crossProduct(v0);
    Vector deltaPCrossV0 = deltaP.crossProduct(v0);

    double a = alignZero(vCrossV0.lengthSquared());
    double b = alignZero( number: 2 * vCrossV0.dotProduct(deltaPCrossV0));
    double c = alignZero( number: deltaPCrossV0.lengthSquared() - radius * radius * v0.lengthSquared());
}

```

```

    }
}

// Compute the quadratic equation coefficients for intersection
Vector vCrossV0 = v.crossProduct(v0);
Vector deltaPCrossV0 = deltaP.crossProduct(v0);

double a = alignZero(vCrossV0.lengthSquared());
double b = alignZero( number: 2 * vCrossV0.dotProduct(deltaPCrossV0));
double c = alignZero( number: deltaPCrossV0.lengthSquared() - radius * radius * v0.lengthSquared());

```

```

// calculate the discriminant
double discriminant = alignZero( number: b * b - 4 * a * c);

// If no real roots, then there is no intersection
if (isZero(discriminant) || discriminant < 0 || isZero(a)) {
    return null;
}

// Solve the quadratic equation
double sqrtDisc = alignZero(Math.sqrt(discriminant));
double denom = alignZero( number: 2 * a);

double t1 = alignZero( number: (-b + sqrtDisc) / denom);
double t2 = alignZero( number: (-b - sqrtDisc) / denom);

// Collect valid intersection points within the maxDistance (excluding exactly equal)
List<AbstractMap.SimpleEntry<Double, Point>> temp = new java.util.LinkedList<>();
final double EPSILON = 1e-10;

if (t1 > EPSILON && t1 < maxDistance) {
    temp.add(new AbstractMap.SimpleEntry<>(t1, ray.getPoint(t1)));
}

if (t2 > EPSILON && t2 < maxDistance) {
    Point point2 = ray.getPoint(t2);
    boolean alreadyExists = false;

    // Avoid adding duplicate intersection points
    for (AbstractMap.SimpleEntry<Double, Point> entry : temp) {

```

```

        if (entry.getValue().distance(point2) < EPSILON) {
            alreadyExists = true;
            break;
        }
    }

    if (!alreadyExists) {
        temp.add(new AbstractMap.SimpleEntry<>(t2, point2));
    }
}

// If no valid intersections found, return null
if (temp.isEmpty()) {
    return null;
}

// Sort intersections by distance from the ray origin
temp.sort(Comparator.comparingDouble(AbstractMap.SimpleEntry::getKey));

// Convert to Intersection objects
List<Intersection> result = new java.util.LinkedList<>();
for (AbstractMap.SimpleEntry<Double, Point> entry : temp) {
    result.add(new Intersection( geometry: this, entry.getValue()));
}

return result;
}

```

## -4 - חישוב חיתוכים בין קין למשולש בשיטה מהירה.

```

    ...
@Override ล Tovat
public List<Intersection> calculateIntersectionsHelper(Ray ray, double maxDistance) {
    // Ray's origin point and direction vector
    Point p0 = ray.getHead();
    Vector dir = ray.getDirection();

    // Vertices of the triangle
    Point v0 = vertices.get(0);
    Point v1 = vertices.get(1);
    Point v2 = vertices.get(2);

    // Calculate the edges of the triangle
    Vector edge1 = v1.subtract(v0);
    Vector edge2 = v2.subtract(v0);

    // Calculate the determinant to check if the ray is parallel to the triangle
    Vector h = dir.crossProduct(edge2); // Cross product of ray direction and edge2
    double a = alignZero(edge1.dotProduct(h)); // Determinant

    // If determinant is near zero, the ray is parallel to the triangle
    if (isZero(a)) return null;

    // Calculate the inverse of the determinant
    double f = alignZero( number: 1.0 / a);

    // Calculate the vector from the ray's origin to the first vertex of the triangle
    Vector s = p0.subtract(v0);

    // Calculate the barycentric coordinate u

    double u = alignZero( number: f * s.dotProduct(h));
    // If u is outside the range [0, 1], the intersection is outside the triangle
    if (u <= 0 || u >= 1) return null;

    // Calculate the barycentric coordinate v
    Vector q = s.crossProduct(edge1);
    double v = alignZero( number: f * dir.dotProduct(q));
    // If v is outside the range [0, 1] or u + v >= 1, the intersection is outside the triangle
    if (v <= 0 || u + v >= 1) return null;

    // Calculate the distance t from the ray's origin to the intersection point
    double t = alignZero( number: f * edge2.dotProduct(q));

    // If t is less than or equal to zero, the intersection point is behind the ray's origin
    if (t <= 0) return null;

    // If t is greater than the maximum distance, the intersection point is too far away
    if (alignZero( number: maxDistance - t) <= 0) return null;

    // Calculate the intersection point
    Point intersectionPoint = ray.getPoint(t);

    // Reject the intersection if it lies exactly on a vertex or edge of the triangle
    if (intersectionPoint.equals(p0) ||
        intersectionPoint.equals(v0) || intersectionPoint.equals(v1) || intersectionPoint.equals(v2)) {
        return null;
    }

    // Return the intersection point as a list
    return List.of(new Intersection( geometry: this,intersectionPoint));
}

```

5- טרנספורמציה סיבוב והזזה של מצלמה.  
ב במחלקה מצלמה הוספו מתודות:

```

/**
 * Moves the camera position by a specified delta vector.
 *
 * @param delta the vector to move the camera position
 * @return the Builder instance
 */
public Builder moveP0(Vector delta) { 1 usage ± Tovat
    camera.p0 = camera.p0.add(delta);
    return this;
}

/* Rotates the camera around the vTo vector by a specified angle in degrees.
 *
 * @param angleDegrees the angle in degrees to rotate the camera
 * @return the Builder instance
 */
public Builder rotateAroundVTo(double angleDegrees) { 1 usage ± Tovat
    // Normalize the angle to the range [0, 360)
    double normalizedAngle = ((angleDegrees % 360) + 360) % 360;

    // Get the current camera vectors
    Vector vUp = camera.vUp;
    Vector vRight = camera.vRight;
    Vector vTo = camera.vTo;

    // Handle special cases for angles 0, 90, 180, and 270 degrees
    if (Math.abs(normalizedAngle - 90) < 1e-6) {
        camera.vUp = vRight;
        camera.vRight = vTo.crossProduct(camera.vUp).normalize();
        return this;
    } else if (Math.abs(normalizedAngle - 180) < 1e-6) {
        camera.vUp = vUp.scale( scalar: -1);
        camera.vRight = vRight.scale( scalar: -1);
        return this;
    } else if (Math.abs(normalizedAngle - 270) < 1e-6) {
        camera.vUp = vRight.scale( scalar: -1);
        camera.vRight = vTo.crossProduct(camera.vUp).normalize();
        return this;
    } else if (Math.abs(normalizedAngle) < 1e-6) {
        return this;
    }
}

```

6- את שלב 6 מימושנו תוך שבוע ימים.

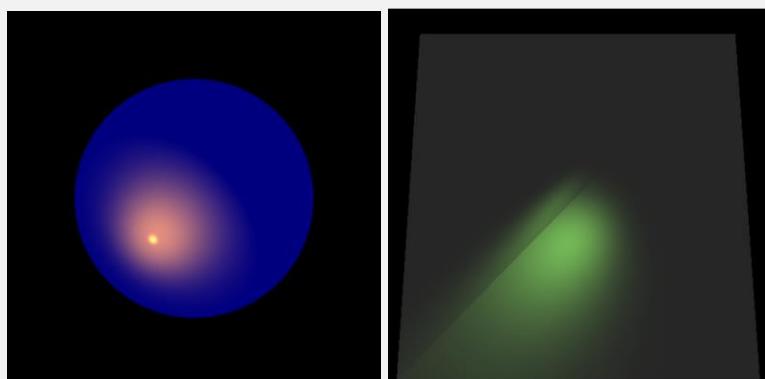
-7 מימושנו אלומת אור צרה-

```
public class SpotLight extends PointLight{ 21 usages ± Tovat +1

    /** The direction of the light beam. */
    private final Vector direction; 2 usages
    /** The narrowness of the beam, which affects the intensity of the light. */
    private double narrowBeam = 1; 2 usages

    * Sets the narrowness of the beam for the spotLight source.
    * The narrow beam affects the intensity of the light.
    * @param narrowBeam The narrowness of the beam.
    * @return The current SpotLight object, allowing for method chaining.
    * @throws IllegalArgumentException if the narrowBeam is less than or equal to 0.
    */
    public SpotLight setNarrowBeam(double narrowBeam) { 3 usages ± Tovat
        if (alignZero(narrowBeam) <= 0) {
            throw new IllegalArgumentException("Narrow beam must be greater than 0");
        }
        this.narrowBeam = narrowBeam;
        return this;
    }

    /**
     * Calculates the intensity of the light at a specific point in the scene.
     * The intensity is affected by the angle between the direction of the light and the direction to the point being calculated.
     * The intensity is also affected by the distance from the light source to the point.
     * @param p - The point in the scene where the intensity is being calculated.
     * @return The intensity of the light at the specified point, represented by a Color object.
     */
    @Override 2 usages ± Tovat
    public Color getIntensity(Point p) {
        Color intensityPoint = super.getIntensity(p);
        double factor = Math.max(0, alignZero(direction.dotProduct(getL(p))));
        if (isZero(factor)) {
            return Color.BLACK;
        }
        return intensityPoint.scale(Math.pow(factor, narrowBeam));
    }
}
```



8- פתרון בעית מרחק בהצללה.

מתודת חישובי חיתוכים מקבל גם טווח מירבי:

```
protected abstract List<Intersection> calculateIntersectionsHelper(Ray ray, double  
maxDistance);
```

המתודה תחזיר רק את נקודות החיתוך שבתווח זה מהמצלמה.

כך בבדיקה הצללה, נבדק אם גוף מסה תאוריה לגוף אחר רק אם רשימת החיתוכים בין הקן מהגוף לתאורה תחזיר גופים במרחב של הגוף לתאורה.

### **נימה אישית:**

במהלך העבודה על הפרויקט זכינו להעמיק בהבנה של עולם הגרפיקה הממוחשבת ולישם עקרונות של תכנות מונחה עצמים ב- Java.

זו הייתה חוויה מأتגרת אך מעשרה, שבה רأינו כיצד רעיונות תאורתיים מתורגםים לתמונה דיגיטלית.

העבודה בצוות, פתרון בעיות והיצירתיות שבפיתוח – חייזקו בנו את האהבה לתחום התוכנות.

אנו מודות על ההזדמנויות לקחת חלק בפרויקט זה, וראות בו צעד משמעותי נוסף במסע הלמידה וההתפתחות שלנו בעולם התוכנות.