דוח מיני פרויקט

מגישים רז אברג'ל ואבי יאלאו

רקע כללי לפרוייקט:

מטרת הפרויקט הינה יצירת תוכנית לדימוי סצנה גרפית וירטואלית תל ת-ממדית .כלל הפרויקט נכתב בשפת JAVA תוך שמירה עבודה לפי עקרונות עיצוב והנדסת תכנה הנלמדות בקורס התאורטי.

הפרויקט נכתב בליווי בדיקות (TESTING) לפונקציות העיקריות כולל בדיקות מחלקות שקילות Equivalence Partitions Tests ובדיקת מקרי קצה . (Boundary Values Tests)
הפרויקט מורכב ממספר חבילות (PACKAGES) באמצעותם נממש את המודל ליצירת תמונה תלת ממדית להלן פירוט החבילות והמחלקות:

PRIMITIVE -מכילה את המרכיבים הבסיסיים להגדרת מרחב R3

COORDINATE-הגדרת קואורדינטה ,נקודה באחד משלושת המישורים שבמרחב התלת ממדי.

POINT- נקודה במרחב מורכבת משלוש קואורדינטות.

VECTOR-ייצוג ווקטור במרחב ,מיוצג על ידי נקודה POINT שמגדירה את ראש הווקטור ועל ידי כך את הכיוון שלו.

RAY-קרן ,קרן הינה ווקטור שמוגדר בנוסף על ידי מיקום התחלה כלומר הכיוון של הקרן מיוצג על ידי וקטור ובנוסף קיימת נקודה POINT שמגדירה את המיקום ההתחלתי שלו.

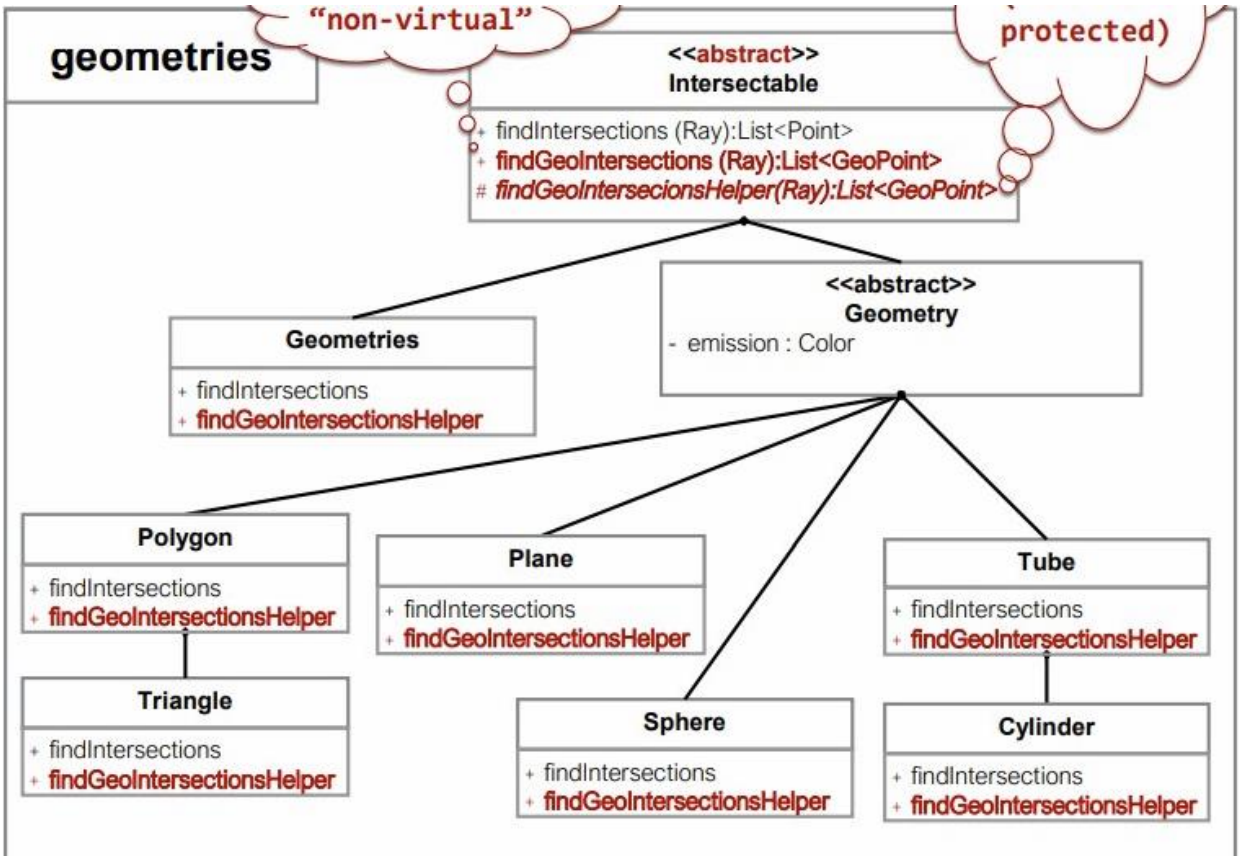
COLOR-הגדרת צבע לכל נקודה במרחב כלומר אם ישנו גוף במרחב אזי הגוף מסוגל לפלוט צבע מסוים ולכן נגדיר צבע לגופים ,תאורות ,ובאופן כללי נוכל להגדיר צבע רקע לסצנה עצמה או להגדיר העדר צבע כלומר צבע שחור.

MATERIAL - חומר ,נגדיר סוג חומר לגופים כלומר מה רמת התגובה של הגוף בפגיעת קרן אור מסוימת הגדרה זו תתבצע על ידי מקדמי הנחתה לאפקטים של החזרת אור ,מקדמי שקיפות והשתקפות ומידת הנצנוץ של החומר.

GEOMETRIES- מכילה את הצורות הבסיסיות שנוכל לשתול בסצנה.
כלל הצורות יורשות מהמחלקה האבסטרקטית GEOMETRY אשר יורשת ממחלקת container היורשת מintersectable
בנוסף יש לנו את מחלקת bounding box שמממשת את **Boundary Volume**

הפונקציה העיקרית שהמחלקות דורסות היא findGeoIntersectionhelper שמחזירה את נקודות החיתוך של הקרניים שנבנה מהמצלמה ושיעברו את המשטח -view plane - עם הגופים בסצנה.
בנוסף כל מחלקה שמייצגת גוף מסוים יודעת להחזיר את הוקטור הנורמלי(אנכי)לגוף זאת על מנת לחשב חיתוכים ואינטראקציות עם תאורה

דיאגרמה של המבנה (שלא כולל את התוספות האחרונות שהכנו לשיפור של הקופסאות)



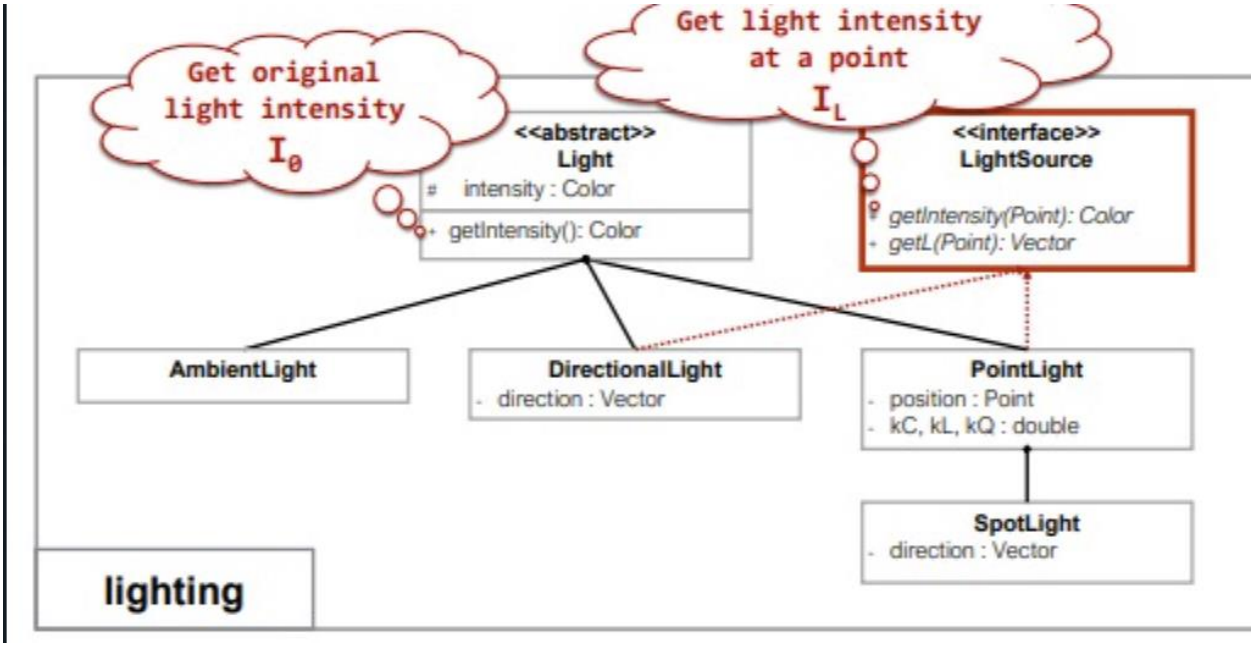
-LIGHTING

כפי שהוזכר לעיל אנו בונים סצנה גרפית על ידי משטח צפייה VIEW PLANE שמאחוריו ממוקמת מצלמה או לחילופין העין של המתבונן בסצנה.

יש לנו שני ממשקים , הממשק– LIGHTSOURCE והממשק Light

כלל סוגי התאורה חוץ מתאורה סביבתית AMBIENT ממשמים ממשק זה הכולל את הפונקציות getIntensityשמחשבת את עומת האור שמגיעה ממקור האור לנקודה ו getL שמחזירה את הוקטור ממקור האור לנקודה המוארת. (directional light, pointLight יורשים מ light וממשים את Lightsource) Ambient מממש את ממשק Light.

דיאגרמה של התיקייה LIGHTING



-RENDERER

נגדיר בחבילת זו את המחלקה CAMERA ובנוסף את כלל סוגי התאורה אשר נתייחס אליהם בפרויקט.

CAMERA-מורכבת מנקודה שמגדירה את המיקום של מרכז העדשה

ובנוסף שלושה וקטורים:RIGHT , TO , UP – שמגדירים את כיוון העדשה ואת התזוזה של על הצירים.

בנוסף קיימות המחלקות IMAGE WRITER שמייצרת קובץ JPG על ידי מטריצת צבעים שנבנית

בתהליך הרנדור .

RAY TRACER BASIC - (המחלקה יורשת מrayTraceBase) כלל החישובים להשגת צבע בכל נקודה בסצנה מתבצע

במחלקה זו על ידי התחשבות באפקטים גלובליים , שקיפות והשתקפות(מראה)של חומרים

ואפקטים לוקלים של תגובת החומר בפגיעת קרן אור והטלת הצל.

בנוסף יש לנו את מחלקת pixel שהיא משמשת בתור מחלקת עזר למימוש תהליכונים.

מיני פרויקט 1

השיפורים שביצענו :

1.DEPTH OF FIELD

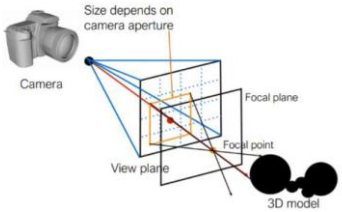
2.ANTI-ALIASING

DEPTH OF FIELD

מתבצע ע"י יצירת צמצם(במקרה שלנו בחרנו לעשות אותו עגול כדי להשיג אפקט טוב יותר) והעברתו על פני

כל פיקסל בנפרד ,כפי שנראה במצגת ההסבר על השיפורים.

השיפור "עומק השדה" הופך את התמונה ליותר מציאותית כך שלא כל התמונה תהיה מפוקסת אלא רק חלק ממנה יתפקס מה שיצור את המראה שיש לתמונה עומק.



על מנת להפעיל את השיפור ,יש להפעיל במצלמה המוגדרת בטסט את השורות הבאות:

1.קביעת הפעלה של האפקט

2. קביעת מרחק מישור הפוקוס (מרחק ה focal plane מ VIEW PLANE.מרחק גדול יותר ,יותר אזורים בפוקוס)

3. קביעת גודל הצמצם (צמצם גדול יותר ,יותר טשטוש)

4. קביעת מספר הקרניים בתוך הצמצם

בתמונה שניצור מרחק משטח הפוקוס יהיה 120,גודל הצמצם ל150 ומספר הקרניים לכל פיקסל 60.

```
camera3.setDepthOfField(true)
        .setFocalDistance(120) // depth
        .setApertureSize(150) // depth
        .setNumOfRaysInAperture(60); // depth
```

במקום להטיל קרן בודדת (castRay) נטיל כמה קרניים

```
    */
    private void castBeam(int nX, int nY, int col, int row) {
        List<Ray> rays = this.constructRaysThroughPixel(nX, nY, col,
```

ובמקום בנית קרן דרך הפיקסל (constructRayThroughPixel)נבנה כמה קרניים דרך הפיקסל

```
    */
    public List<Ray> constructRaysThroughPixel(int nX, int nY, int j, int i) {
```

בתוך הפונקציה הנ"ל נבדוק אם מספר הקרניים שונה מ1 ,אם אכן כך אני אבנה כמה קרניים

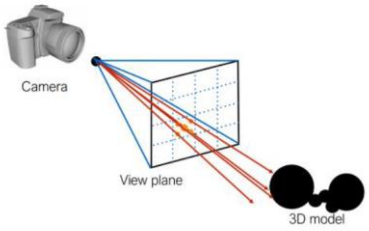
ואם לא פשוט נשלח את הקרן היחידה

```
// if more than one ray is omitted (bor effect)
if (numOfRaysInAperture != 1) {
    List<Ray> temp_rays = new LinkedList<>();
    // apertureSize is the value of how many pixels it spreads on
    double apertureRadius = Math.sqrt(ApertureSize * (pixelHeight * pixelWidth)) / 2d;
    for (Ray ray : rays) {
        // creating list of focal rays (from the aperture on the view plane to the point on the focal plane)
        temp_rays.addAll(ray.randomRaysInCircle(ray.getP0(), get_vUp(), get_vRight(), apertureRadius, numOfRaysInAperture, FocalDistance));
    }
    // the original rays included in the temp rays
    rays = temp_rays;
}

return rays;
```

ANTI-ALIASING

supersampling anti-aliasing היא שיטת spatial anti-aliasing כלומר שיטה להפחתת קצוות מעוותים או מפוקסלים מתמונה המרונדרת למשחק מחשב או לכל תוכנת מחשב שיוצרת תמונות. aliasing נוצר מפני שלעומת העולם האמיתי שמורכב מקימורים מתמשכים המחשב מייצג תמונה על ידיי הרבה ריבועים קטנים כלומר פיקסלים ולכן אם נרצה לצייר קו הוא יראה מפוקסל (אלה אם כן הוא מאונך או מאוזן לגמרי). המטרה של Supersampling הא הפחתת הaliasing דבר זה נעשה על ידיי רינדור התמונה ברזולוציה גבוהה יותר מהרזולוציה הרצויה ולאחר מכן מעבר על כל פיקסל ויצירת פיקסל חדש מממוצע הצבעים מרחבי הפיקסל ולא רק מאמצע הפיקסל כמו שעושים בדרך כלל. (ויקיפדיה)



המימוש של שיפור זה יתבצע ע"י שליחת קרניים מרובות לתוך הפיקסל וחישוב ממוצע של הצבע של כל הקרניים

גם בשיפור נשתמש במקום הפונקציות שמשתמשות בקרן אחד בפונקציות המשתמשות בקרניים מרובות

(עיין לעיל עמוד קודם)

השוני נראה כך

אם כמות הקרניים שונה מ1 (זאת אומרת שאנחנו רוצים להפעיל את השיפור)

ניצור רשימת קרניים מפוזרות

```
if (numOfRaysInPixel != 1) {
    rays.addAll(centerRay.randomRaysInGrid(
            get_vUp(),
            get_vRight(),
            numOfRaysInPixel,
            _distance,
            pixelWidth,
            pixelHeight)
    );
```

ולאחר ממכן נחשב את ממוצע הצבעים עבור כל פיקסל

בפונקציה castBeam

```
private void castBeam(int nX, int nY, int col, int row) {
    List<Ray> rays = this.constructRaysThroughPixel(nX, nY, col, row);

    List<Color> colors = new LinkedList<>();
    for (Ray ray : rays) {
        colors.add(rayTracerBase.traceRay(ray));
    }
    imageWriter.writePixel(col, row, Color.avgColor(colors));
}
```

מיני פרוייקט 2

תהליכונים :

על מנת לשפר את זמן הריצה של רנדור התמונה נשתמש בתהליכונים על מנת לחלק את העיבוד ולבצע אותו באופן מהיר יותר. נגדיר את מספר התהליכונים שאנחנו רוצים להריץ .

```
camera3.setMultithreading(6)
```

נגיע לחלק הזה שעבור כל תהליך שהגדרנו הוא יאתחל תהליכון חדש נוסף

```
} else {
    while (threadsCount-- > 0) {
        new Thread(() -> {
            for (Pixel pixel = new Pixel(); pixel.nextPixel(); Pixel.pixelDone())
                if (!this.is_DepthOfField() && !this.is_AntiAliasing()) {
                    CastRay(nX, nY, pixel.row,pixel.col);
                }
                else {
                    castBeam(nX, nY, pixel.row,pixel.col);
                }

        }).start();
    }
    Pixel.waitToFinish();
}
return this;
```

נראה שאכן המימוש עובד ומשפר את זמן הריצה:

קומקום בלי שיפורים :

```
Test Results        1 min 1 sec    C:\Users\razab\.jdks\openjdk-17.0.2\bin\java.exe ...
  TeaPotTest        1 min 1 sec
    teapot()        1 min 1 sec    Process finished with exit code 0
```

קומקום עם ארבעה תהליכונים :

```
Tests passed: 1 of 1
Test Results        29 sec 393 ms    C:\Users\razab
  TeaPotTest        29 sec 393 ms    100.0%
    teapot()        29 sec 393 ms
                                     Process finish
```

נראה שיפור של פי 2 מזמן הריצה הרגיל.

: Boundary Volume Hierarchy

שיפור משמש כדי לחסוך את חישוב חיתוכי הקרניים של האובייקטים בסצנה (למעשה אנחנו מחפשים את כל הגאומטריות שהקרן חתכה אותם מה שיכול להביא לזמן ריצה ארוך יותר) על מנת לפתור בעיה זאת נחסום כל צורה בסצנה שיצרנו בתוך קופסה חוסמת שתכיל את בצורה מלאה את כל האובייקט.

לאחר נבנה עץ היררכי שיעזור לנו להבין איפה בדיוק פגעה והאם נצטרך לבחון עוד צורות שהקרן פוגעת בה.אם קרן פגעה באחת האבות של העץ נדע שגם צריך להתייחס לבנים של אותו אבא. שיפור ימנע מאיתנו לבחון כל פעם את כל הגיאומטריות שהקרן פגעה בהם ובכך נוכל לחסוך ולשפר את זמן הריצה.ה

המחלקה boundingBox תממש את הקופסה החוסמת ובעזרתה כל צורה תדע איך לחסום את עצמה.

דוגמא :חסימה של ספירה

```java
/**
 * method sets the values of the bounding volume for the intersectable sphere
 */
@Override
public void setBoundingBox() {

    super.setBoundingBox();

    // get minimal & maximal x value for the containing box
    double minX = _center.get_x() - _radius;
    double maxX = _center.get_x() + _radius;

    // get minimal & maximal y value for the containing box
    double minY = _center.get_y() - _radius;
    double maxY = _center.get_y() + _radius;

    // get minimal & maximal z value for the containing box
    double minZ = _center.get_z() - _radius;
    double maxZ = _center.get_z() + _radius;

    // set the minimum and maximum values in 3 axes for this bounding region of the component
    boundingBox.setBoundingBox(minX, maxX, minY, maxY, minZ, maxZ);
}
```

הפעלת בניית העץ ב test דרך סצנה

```
scene1.geometries.BuildTree();
```

```
.setRayTracer(new RayTracerBasic(scene1).set_bb(true))
```

וכן במחלקת geometries נבצע את המימוש של בניית העץ

```
public void BuildTree() {

    // flatten the list of Geometries
    this.flatten();

    // define reusable variables (improved performance)
    double distance;
    Container bestGeometry1 = null;
    Container bestGeometry2 = null;

    // while any container contains more then one geometry
    while (_containers.size() > 1) {
        double best = Double.MAX_VALUE;
        // loop through the containers
        for (Container geometry1 : _containers) {
            for (Container geometry2 : _containers) {
                // measure the distance between every couple of bounding boxes
                distance = geometry1.boundingBox.BoundingBoxDistance(geometry2.boundingBox);
                // if the geometries are not the same geometry, and the distance is the lowest possible
                if (!geometry1.equals(geometry2) && distance < best) {
                    // define the best distance as the minimal one we have found
                    best = distance;
                    // define the best candidates to be together in a container
                    bestGeometry1 = geometry1;
                    bestGeometry2 = geometry2;
                }
            }
        }
        // after we have determined the best geometries to couple in a container,
        // create new container which contains the geometries
        _containers.add(new Geometries(bestGeometry1, bestGeometry2));
        // and remove the same ones from the original tree
        _containers.remove(bestGeometry1);
        _containers.remove(bestGeometry2);
    }
```

הפונקציה קודם כל משטיחה את כל הצורות שיש צורך להשטיח אותם (צורות מורכבות כמו פירמידה)

הלולאה בפונקציה תרוץ עד שכל הצורות יכנסו לעץ ותהיה קופסא אחת שמכילה את הכל .

עבור כל אב של העץ יהיו לנו שתי צורות שיהיו הכי טובות להיות "הבנים" של אותו האב.

ונמשיך כך עד שיבנה לנו העץ הבינארי המכיל את הגיאומטריות בסצנה.

שנרנדר את הסצנה הקוד יעבוד באופן זהה לפעמים הקודמות ,אלא שהפעם במקום לבדוק חיתוכים של הקרן עם כל הגיאומטריות בסצנה הפעם נשתמש בפונקציה מותאמת שתבדוק חיתוכים במקומות הרלוונטים לפי העץ.

אם הקרן לא תחתוך קופסא חוסמת רלוונטית לא יהיה טעם לבדוק חיתוכים שם ובכך נחסוך זמן יקר.

נעיר שתיתכן תקורה כזאת או אחרת בעקבות בניית העץ.

```java
public List<GeoPoint> findIntersectBoundingRegion(Ray ray) {
    if (boundingBox == null || boundingBox.intersectBV(ray)) {
        return findGeoIntersections(ray, Double.POSITIVE_INFINITY);
    }
    return null;
```

קומקום עם **Boundary Volume ותהליכונים**

```
Test Results                    45 sec 248 ms
  TeaPotTest                    45 sec 248 ms
    teapot()                    45 sec 248 ms

C:\Users\razab\.jdks\openjdk-17.0.2\bin\java.exe ...
100.0%

Process finished with exit code 0
```

נשים לב שישנה תקורה של הבניית העץ ולכן זה רץ טיפה יותר לאט

תמונה סופית לפני שיפורים

```
Test Results                    15 hr 10 min
  FinalProjectTest              15 hr 10 min
    miniProject()               15 hr 10 min
```

זמן ריצה עם כל השיפורי תמונה ושיפורי זמן הריצה

נראה כי זמן הריצה שופר פי 7.5

```
Test Results                    2 hr 0 min
  FinalProjectTest              2 hr 0 min
    miniProject()               2 hr 0 min

C:\Users\razab\.jdks\openjdk-17.0.2\bin\java.exe ...
100.0%

Process finished with exit code 0
```

לפני שיפורים:

אחרי שיפורים של עומק השדה ושיטוח עקומות

נבחין כי החלק השמאלי של התמונה מפוקס יותר ובכך נותן תחושת עומק לתמונה

לפני AA

אחרי AA