אביעד שלוסברג
אביתר לוי

דו"ח פרויקט

(מיני פרויקט 2)

- כל הפונקציות מחזירות פרמטרים חדשים תמיד – כולל הפונקציות של חיבור/חיסור, או הכפלה
- וקטורית (למעט נרמול שמתבצע על האיבר עצמו), בהתאם להנחיות רכז הקורס.
- השתמשנו כמעט תמיד בשמות משתנים אינדיקטיביים למעט במקומות בהם העתקנו נוסחאות מתמטיות, במקרה כזה, העתקנו את שמות המשתנים מהנוסחה עצמה.
- הסברים לפני כל פונקציה, ובמקרה הצורך גם in-code כדי לאפשר קריאה נוחה וקלה של הקוד גם למי שקורא אותו לראשונה.
- Javadoc - הסבר פורמלי עבור כל מחלקה.

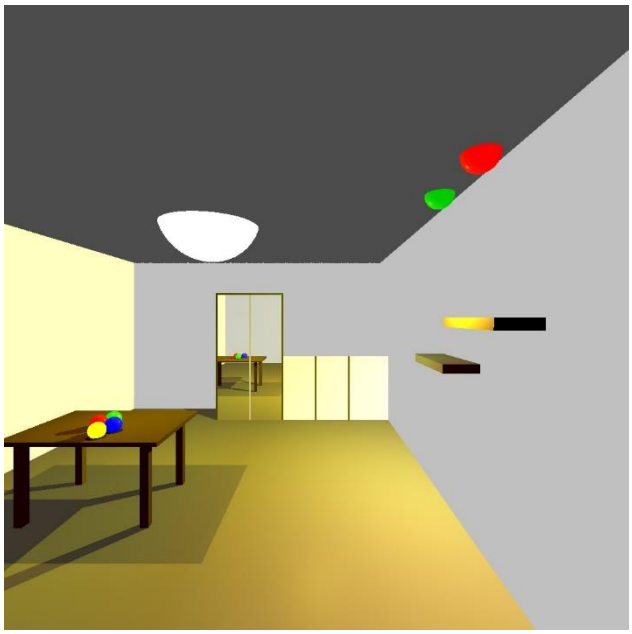
השיפורים:

הוספנו שני שיפורים עבור התמונה שלנו:

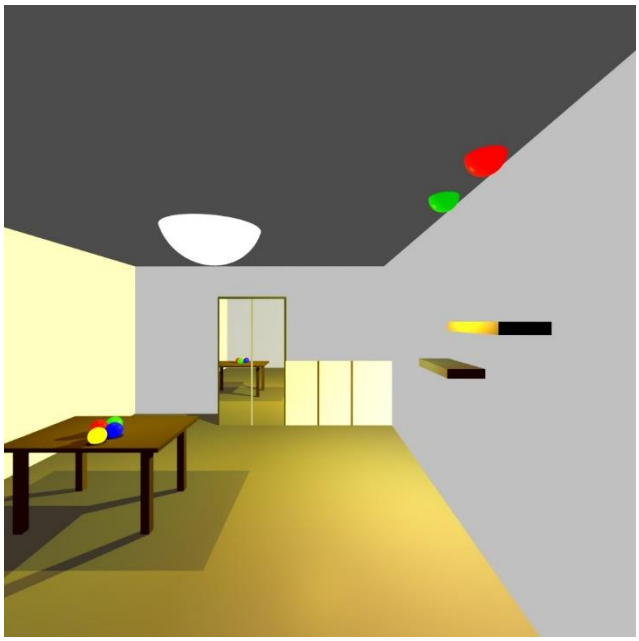
שיפור 1 - Anti-Aliasing

שיפור Anti-Aliasing ליצירת מראה טבעי ומוחלק יותר בקצוות של אובייקטים המופיעים בתמונות, כפי שניתן לראות בהדגמה הפשוטה הזאת:

לפני השיפור, הקצוות של הגופים נראים משוננים בצורה לא טבעית



לאחר השיפור, קצוות הגופים מוחלקים ונראים טבעיים יותר



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

- קביעת הפעלה של האפקט
- קביעת מספר הקרניים לכל פיקסל

הדגמה של הפעלת האפקט ביצירת הטסט:

```
.setAntiAliasing(true)
.setNumberOfRaysInPixel(81);
```

בתמונה המשופרת למעלה קבענו:

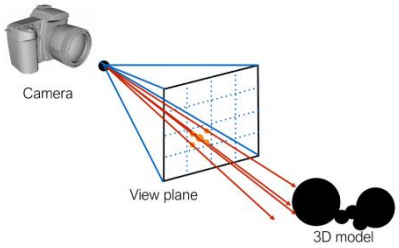
את אפקט antiAliasing להיות true.

את מספר הקרניים לכל פיקסל ל־80.

הערה:

*מומלץ להגדיל את התמונות כדי לראות ביתר בירור את השפעת האפקט.

הסבר קצר על מימוש השיפורים – Anti-Aliasing:



מתבצע ע"י שליחת קרניים מרובות המפוזרות ברחבי הפיקסל, וחישוב הממוצע עבור כל הקרניים.

שינוי א: מחלקת camera
במקום להשתמש בפונקציה הזו שמטילה קרן בודדת

```
private Color castRay(int col,int row)
{
    Ray rayForCast= constructRay(imageWriter.getNx(), imageWriter.getNy(), col,row);
    return rayTracerBasic.traceRay(rayForCast);
}
```

שקוראת לפונקציה

```
public Ray constructRay(int nX, int nY, int j, int i) {...}
```

השתמשנו בפונקציה שמטילה אלומה

```
private Color castBeam(int nX, int nY, int col, int row) {
    List<Ray> rays = constructRays(nX, nY, col, row);

    List<Color> colors = new LinkedList<>();
    for (Ray ray : rays) {
        colors.add(rayTracerBasic.traceRay(ray));
    }
    return Color.avgColor(colors);
}
```

שקוראת לפונקציה שמעבירה יותר מקרן אחת לכל פיקסל

```
public List<Ray> constructRays(int nX, int nY, int j, int i) {
```

בתוך הפונקציה הזו, אם המשתנה שמגדיר כמות קרניים לפיקסל אינו 1

```
if (numberOfRaysInPixel != 1) {
```

ניצור רשימת קרניים מפוזרות בעזרת הפונ' randomRaysInGrid (נוצרה במחלקת ray):

```
if (numberOfRaysInPixel != 1) {
    rays.addAll(centerRay.randomRaysInGrid(
```

```
public List<Ray> randomRaysInGrid(
```

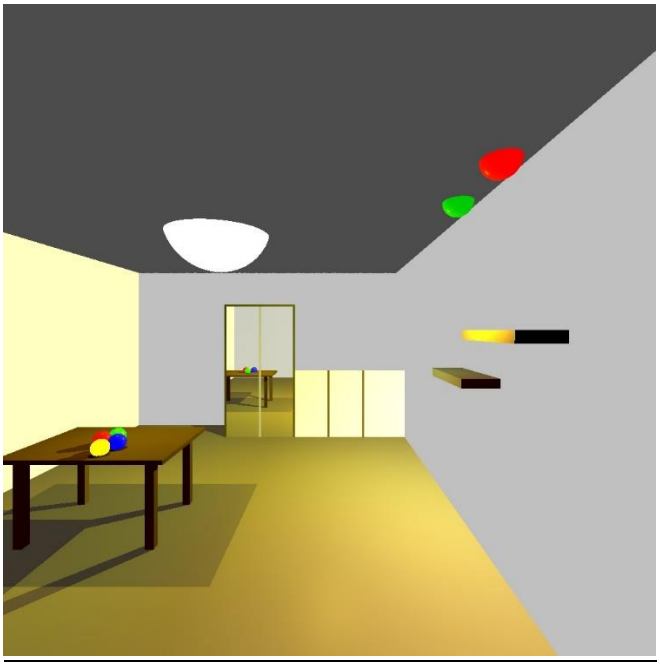
נחזיר את רשימת הקרניים הרנדומליות לפונק' castBeam, נעקוב אחרי כל קרן, נחשב את ממוצע הצבעים עבור הפיקסל, ונחזיר את הצבע הממוצע לקריאה של renderImage :

```
for (int i=0; i< imageWriter.getNx();i++){
    for (int j=0; j< imageWriter.getNy();j++){
        imageWriter.writePixel(j,i,castBeam(nX,nY,j,i));
    }
}
```

שיפור 2 - Soft Shadows

יצרנו צלליות המדמות צללים אמיתיים (לא חדים מדיי). ההשפעה שלהם נכרת בתמונה, לפני השיפור התאורות עושות רושם רב מה שיוצר תמונה מוארת ללא עידון צלליות, ואילו לאחריו יש איזון מדהים בין התאורות ובין הצלליות.

לפני השיפור:



אחרי השיפור

הפעלת התוסף בעזרת הצבת מס' לפרמטר MIN_SHADOW_POINTS (כאן הצבנו 50)

```
camera.setImageWriter(new ImageWriter(  imageName: "Billiard",  nX: 500,  nY: 500))
        .setRayTracer(new RayTracerBasic(myScene).setMIN_SHADOW_POINTS(50)); //
```

כמו כן, הוספת רדיוס למקור התאורה הנוסף לתמונה:

```
myScene.lights.add(
        new PointLight(new Color(WHITE).reduce(3.2), new Point( x: 10,  y: 0,  z: 15), radius: 2)
```

אופן המימוש:

יצרנו target area בצורת עיגול דו מימדי אליו נשלח מס' קרניים עבור ריכוך השפעת הצל.

לשם כך, הוספנו פרמטר "רדיוס" לכל בנאי מקורות התאורה שלנו:

```
public PointLight(Color color, Point position, double radius)
```

בממשק lightSource הוספנו מתודה אבסטרקטית שתמומש במקורות התאורה של point,spot.
תפקידה לחשב את מס' הנקודות הרנדומלי:

```
List<Point> lightPoints(Vector lightDirection, int minPoints);
```

מימוש בpoint light (באותו אופן גם בspot – אלא ששם נשלח וקטור כיוון שונה) :

```
public List<Point> lightPoints(Vector lightDirection, int minPoints) {
    Plane targetArea = new Plane(position, lightDirection);
    return pointsOnTarget(position,radius,targetArea,minPoints);
}
```

פונק' זו יוצרת משטח שאליו נזרוק את הקרניים, ועבור חישוב מס' הנקודות שימצאו על המשטח מתבצעת קריאה לפונקציית עזר :

```
public List<Point> pointsOnTarget(Point center, double radius, Plane plane, int minPoints)
```

במחלקת RayTracerBasic

יצרנו פרמטר המגדיר את מס' הנקודות המינימלי לזריקת קרניים על משטח היעד.

```
//A number of minimum points required to distribute to the surface
int MIN_SHADOW_POINTS = 0;
```

עבור חישוב הקרניים עצמן עדכנו את מימוש הפונ' של transparency , מקדם השקיפות מתחשב כעת במכלול הקרניים הנשלחות ולא רק בקרן יחידה כפי שחושב לפני התוסף:

במידה ומס' הנקודות המינימלי שונה מאפס (הופעל התוסף) הרי שנחשב את סך הנקודות ובהתאם את מס' הקרניים:

```
if (MIN_SHADOW_POINTS != 0){                    //activate soft shadow
    lightPoints = ls.lightPoints(l, MIN_SHADOW_POINTS);
}
```

בהתאם לכך מושפע מקדם ההנחתה של השקיפות הממוצע:

```
for(Ray ray:lightRays){
    var intersections : List<Intersectable.GeoPoint> = scene.geometries.findGeoIntersections(ray);
    if (intersections == null)
        ktrTotal= ktrTotal.add(ktr);
    else{
        double lightDistance = ls.getDistance(geoPoint.point);
        for (GeoPoint gp : intersections) {
            if (alignZero( number: gp.point.distance(geoPoint.point) - lightDistance) <= 0) {
                ktr = ktr.product(gp.geometry.getMaterial().kT);
                if (ktr.lowerThan(MIN_CALC_COLOR_K)) break;
            }
        }
        ktrTotal = ktrTotal.add(ktr);
    }
}
return (ktrTotal.scale((double) 1/lightRays.size()));
}
```

האצת התכנית

הוספנו שני שיפורים למנוע שלנו:

1. Multi-Threading

על מנת לאפשר למנוע להשתמש ביותר מתהליכון בודד, מה שמשיג שיפור ביצועים של פי 2.5 (כמו שכתב רכז הקורס: "כשמשתמשים ב-3 תהליכונים נצפה שיפור של פי 2.5 בערך בזמן ריצה עבור תמונות כבדות יחסית בהשוואה לרינדור ללא תהליכונים").

על מנת להפעיל את השיפור, יש להגדיר ב־Camera המוגדר בטסט:

- הפעלת Multi-Threading עם הגדרת מספר התהליכונים שנרצה להקצות

הדגמה של הפעלת האפקט

```
.setNumberOfRaysInPixel(120)
.setMultithreading(3);
```

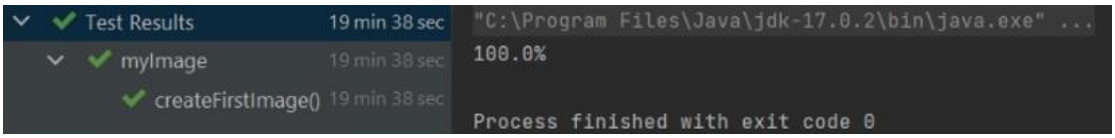
בתמונה המשופרת למעלה קבענו:

את מספר התהליכונים המוקצים לתיצוג (רינדור) ל־3.

זמן ריצה לפני השיפור:

לקח לנו 52 דקות.

זמן ריצה לאחר השיפור (הוקצו 3 ליבות):



.

2. Bounding Boxes

על מנת לחסוך את חישוב חיתוך הקרניים עם כל האובייקטים בסצנה (כלומר עבור כל קרן נצטרך לבדוק את כל הגיאומטריות), נבצע חסימה פשוטה של כל אובייקט בסצנה בקופסה חוסמת (bounding box) שתכיל בצורה מלאה את האובייקט (כמובן למעט אובייקטים אינסופיים, שלא ניתן להגדיר עבורם קופסה חוסמת).

לשם הפעלת התוסף, בנינו מחלקה גאומטרית בשם Box , המכילה את המתודה IsRayHitBox , שבעזרתה נבצע מיפוי משמעותי בבדיקות הקרניים שכן נבחן בדיקה כללית האם קיים חיתוך בין הקרן לקופסא.

במידה וקיים חיתוך, אזי נבחן חיתוך בין הקרן לגוף. לשם כך, הגדרנו מתודה אבסטרקטית בממשק intersectable :

```
public abstract void setBox();
```

מימשנו מתודה זו בכל מחלקת גוף גאומטרי ע"פ הנוסחאות המתמטיות המתאימות.

בשלב הבא, בצענו בנייה היררכית של קופסאות, כלומר כל קבוצת גופים שנמצאת באזור מסוים אוחדה תחת קופסא אחת, ומעל גבי כל הקופסאות עוטפת קופסא אחת גדולה.

```
public void setGeometriesBoxes() {
    for(Intersectable geo : geometriesBodies) {
        geo.setBox();
    }
}
```

לאחר יצירת ההיררכיה, ה־RayTracer שלנו ימשיך לעבוד כרגיל, אבל הפעם, במקום לבדוק חיתוכים של הקרן עם כל גיאומטריה בעזרת הפונקציה findGeoIntersections, נשתמש בפונקציה findIntersectBoundingRegion, שהמימוש שלה כך

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

כלומר נבדוק אם לא קיימת קופסה חוסמת (כלומר אנחנו בודקים חיתוך עם גיאומטריה ממש, כלומר הגענו לעלה בעץ) או שהקרן חותכת את הקופסה החוסמת, נבצע בדיקת חיתוך, אחרת נוכל לומר מראש שאין טעם לבדוק חיתוכים ולכן נקבל null (ובסופו של דבר הקרן תחזיר את צבע הרקע).

הפעלת התוסף בעזרת הפעלת הפונקציה turnAllBoxesOn , שיפעיל את כל הקופסא הגדולה של geometries שתפעיל את כל הקופסאות שתחתיה:

```
public RayTracerBasic turnAllBoxesOn() {
    scene.geometries.setBox();
    return this;
}
```

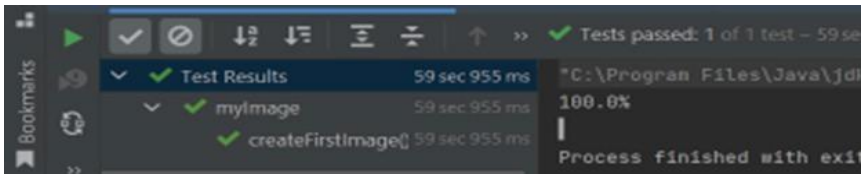
```
.setRayTracer(new RayTracerBasic(myScene)
        .setMIN_SHADOW_POINTS(200)
        .turnAllBoxesOn());
```

שיפור ביצוע: לפני הפעלת התוסף, במשוער 14 שעות.

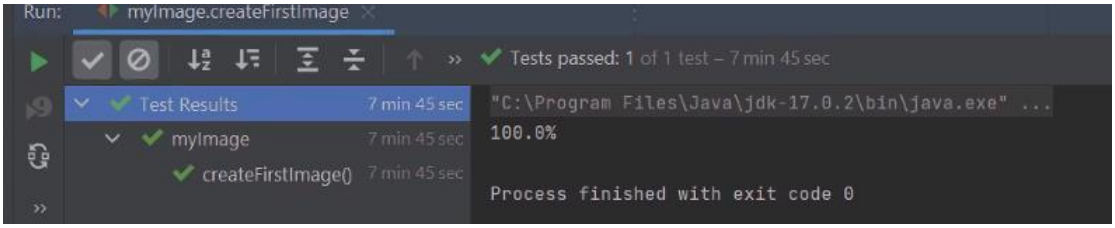
לאחר שיפור: שעה וחצי.

שיפור ביצועים

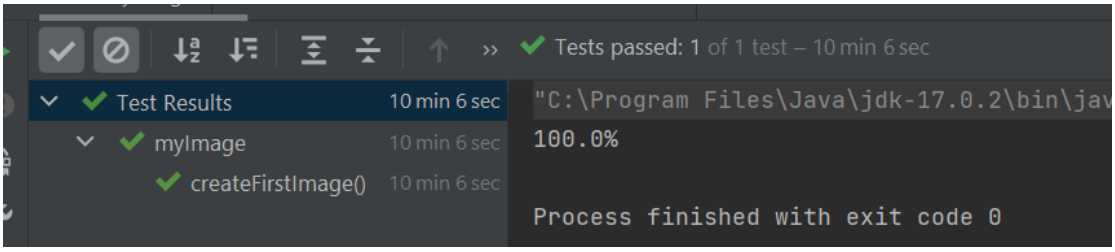
ללא תוספים:



עבור Soft בלבד (200 קרניים):



עבור AA בלבד (120 קרניים):



עבור שני התוספים יחד (200,120): שעה וחצי.

נספח בונוסים

1. אין חובה לממש את הפעולה עבור גליל סופי – מי שיממש בצורה נכונה – יקבל בונוס של 1 נק' לציון הכללי
2. בדיקות + מימוש חיתוכי קרן ומצולע (1 נק')
3. בדיקות + מימוש חיתוכי קרן וגליל [סופי] (1 נק')
4. השלמת שלב 6 תוך שבוע אחד במקום שבועיים. (1 נק')
5. מימוש דרך 2 בפתרון בעיית ההצללה (עדכון Intersectable בכל הגופים עם פרמטר של מרחק)