דו"ח פרויקט

(מיני פרויקט 1)

- 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.

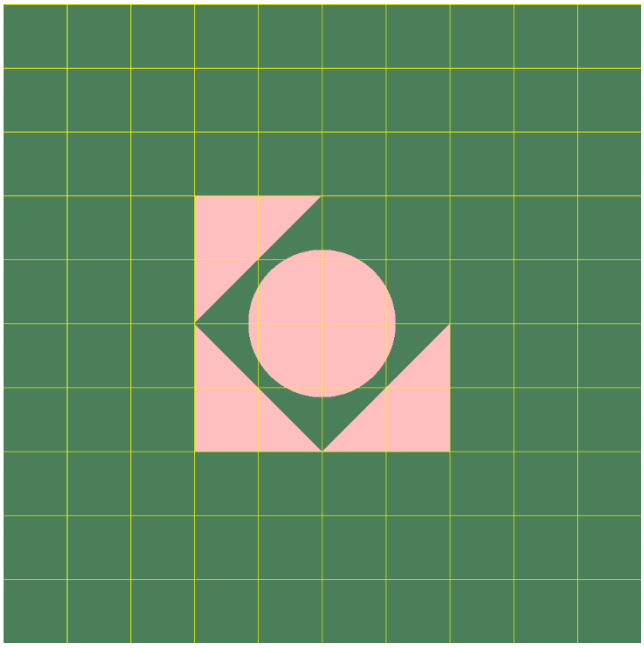
השיפורים:

הוספנו שני שיפורים עבור התמונה שלנו:

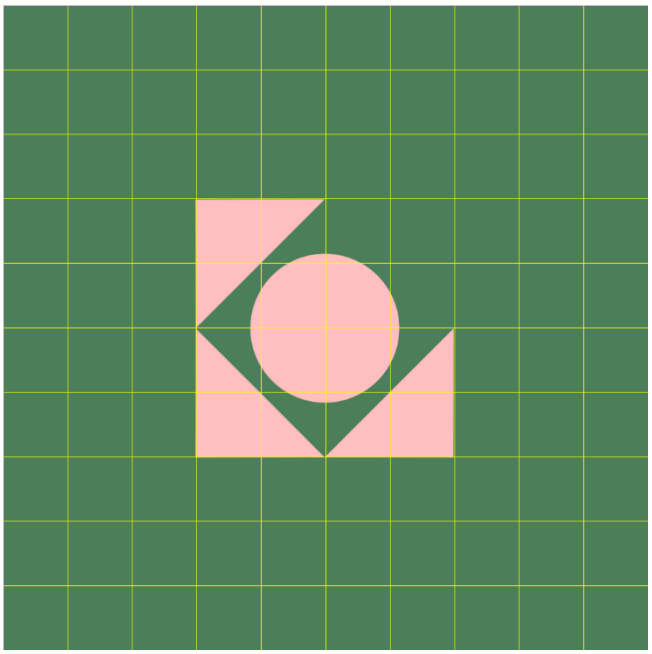
שיפור 1 - Anti-Aliasing

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

לפני השיפור, הקצוות של הכדור נראים משוננים בצורה לא טבעית



לאחר השיפור, קצוות הכדור 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:

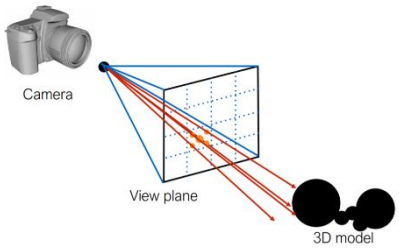
את אפקט antiAliasing להיות true.

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(
```

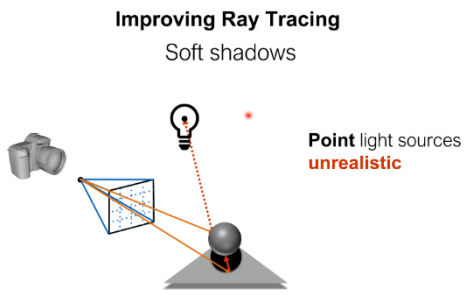
```
public List<Ray> randomRaysInGrid(
```

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));
    }
}
```

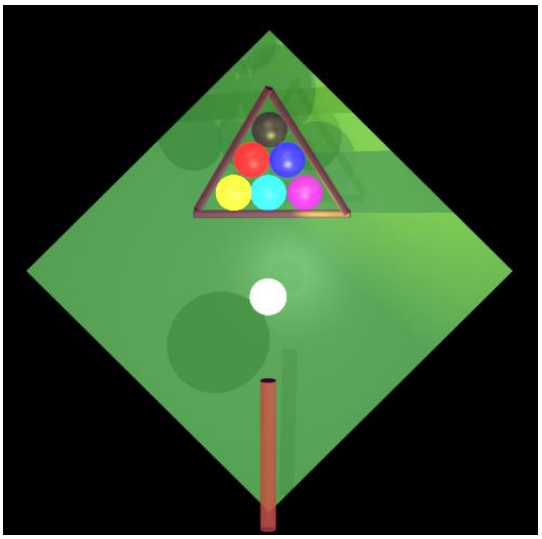
אביעד שלוסברג
אביתר לוי

שיפור 2 - Soft Shadows

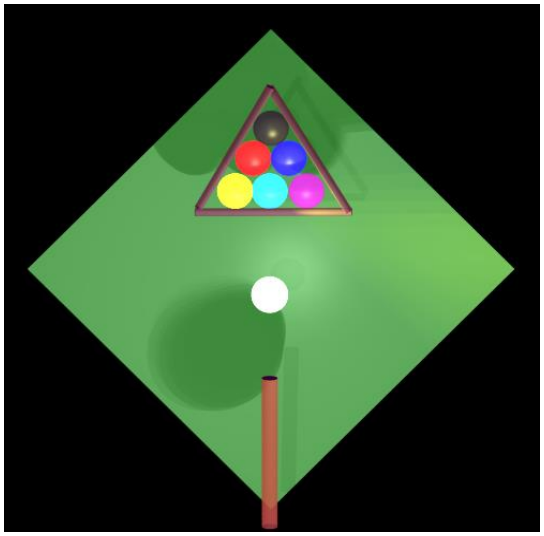


יצרנו צלליות המדמות צללים אמיתיים (לא חדים מדיי)

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()));
}
```