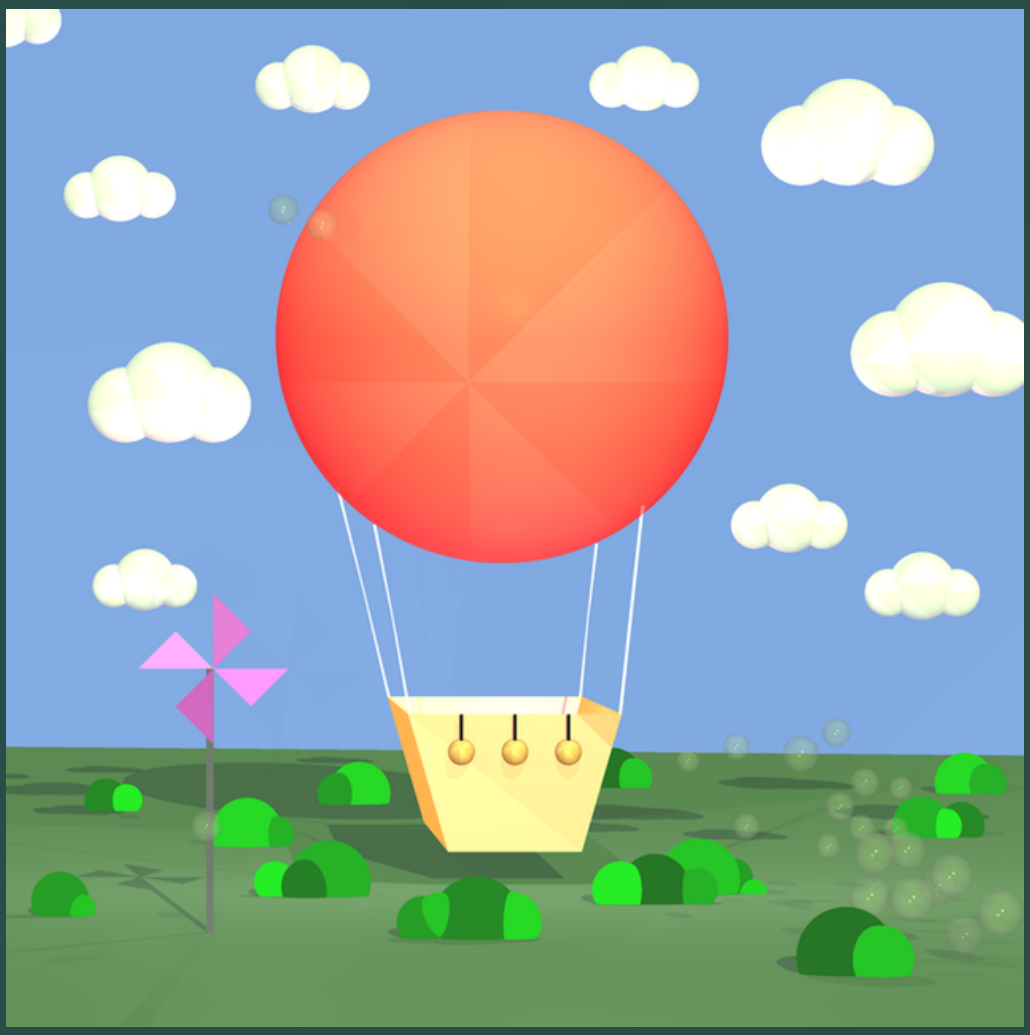
בע"ה

מיני פרויקט במבוא להנדסת תוכנה

סמסטר ב' תשפ"ג

נעה הראל 214939290 וטלאל גינזברג 215750514



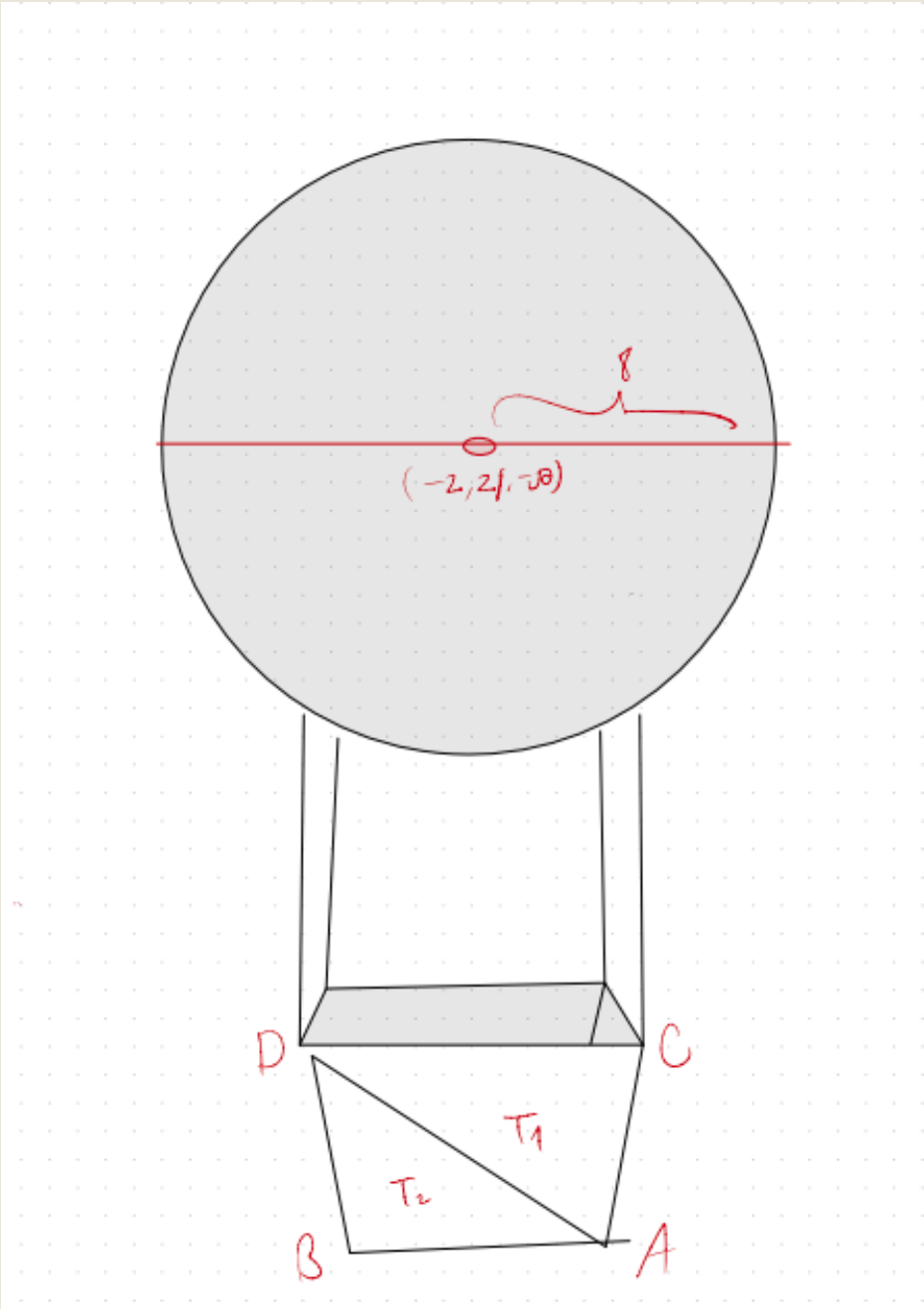
בע"ה

תוכן העניינים

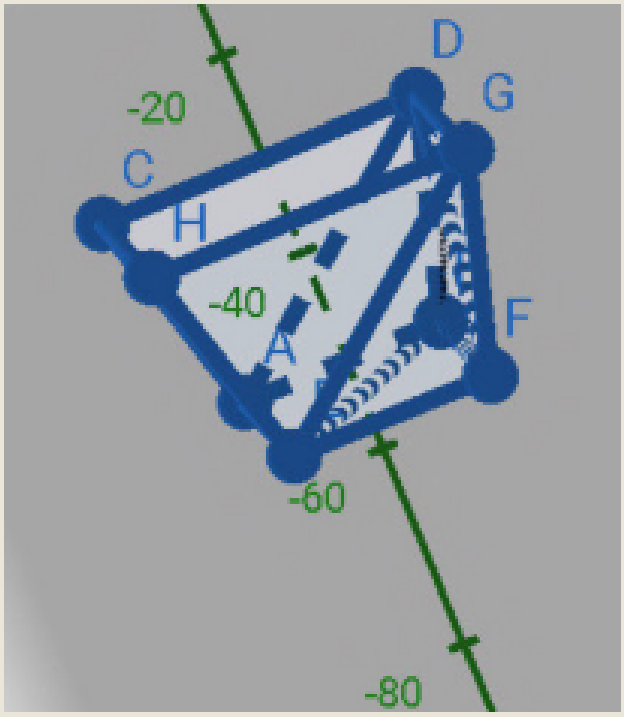
תכנון התמונה

שיפורים

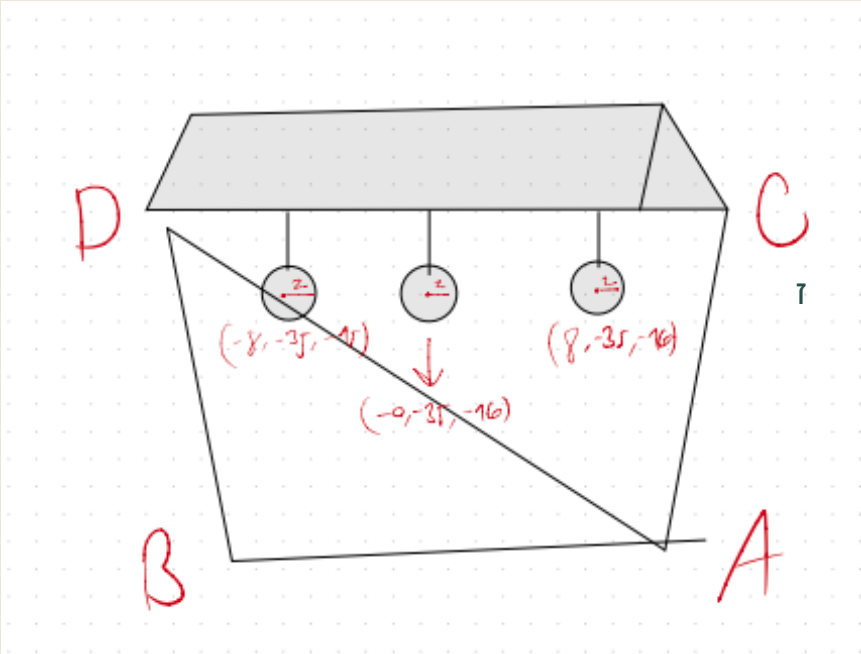
עיטוטים



רצינו ליצור תמונה שמתמקדת בכדור פורח, ולכן יצרנו בתלת ממד כדור שמורכב מסלסלה חוטים, ובלון. את דפנות הסלסלה יצרנו עם 2 משלושים בין 4 נקודות. את החוטים יצרנו ע"י שתי נקודות קרובות מאוד על הספרה ושתי נקודות קרובות מאוד על הסלסלה, ובניהם 2 משולשים שיוצרים פס.



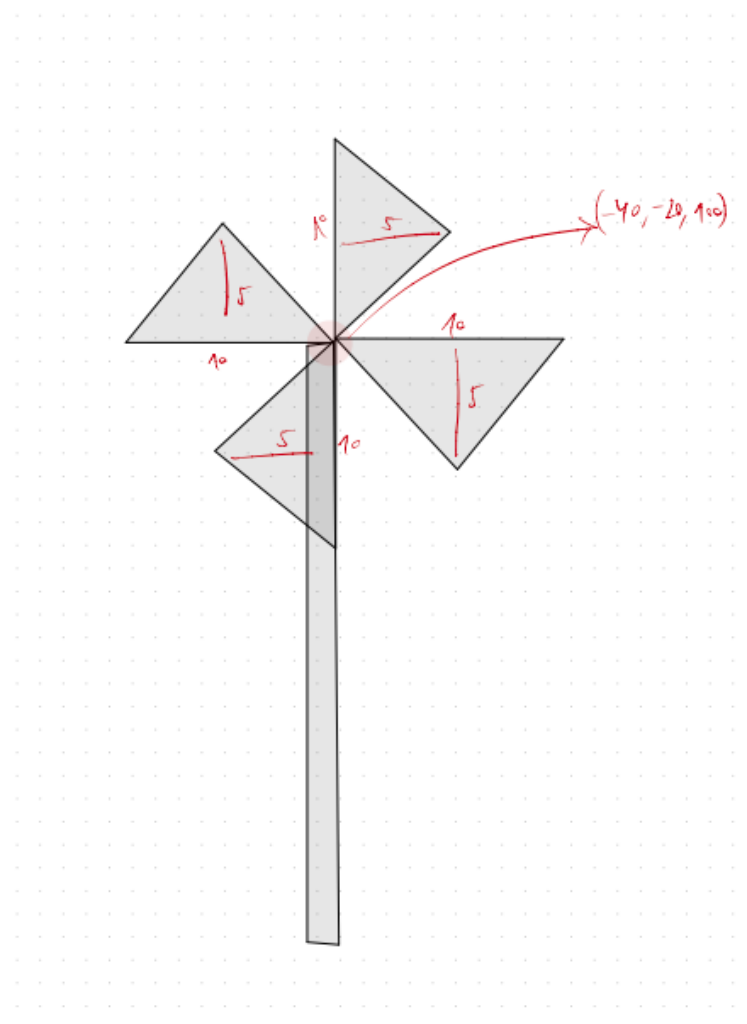
כדי ליצור את הסלסלה בתלת ממד נעזרנו בGeogebra.
יצרנו סלסלה שמתאימה לצורה שרצינו דרך האפליקציה, ואז הגדלנו את הפרופורציות בהתאם לגודל שהתאים לתמונה שהיה לנו בראש.



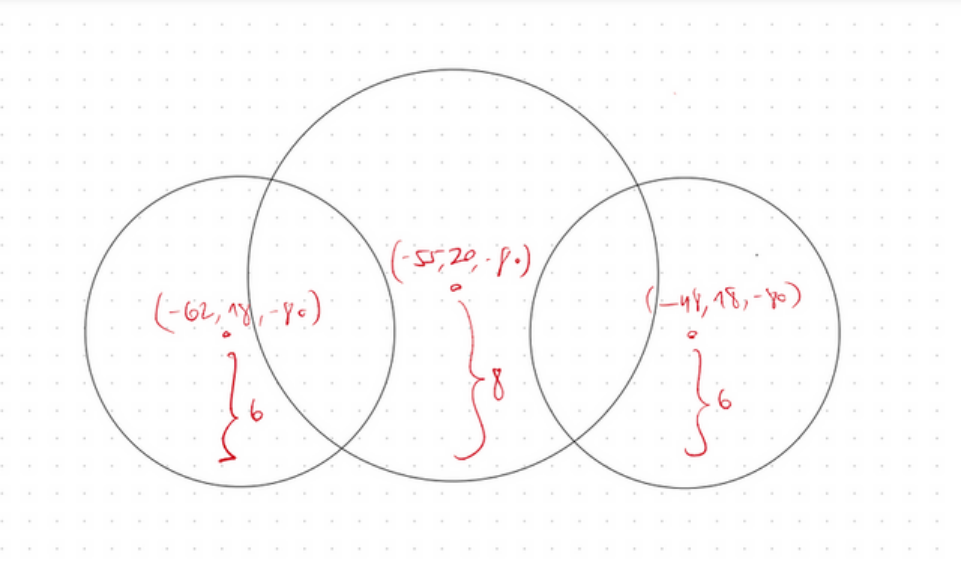
אחרי שכבר היה לנו סלסלה רצינו להוסיף לה שקי חול, כפי שיש בכדור פורח מציאותי. הורדנו חוט מהלמעלה של הסלסלה(באותה שיטה שיצרנו חוטים בין הסלסלה לבלון) לגובה אחיד ושם יצרנו ספרה.

ערטוטים

ערטוטים



יצרנו שבשבת שעומדת על הקרקע. השבשבת מורכבת מ-4 משולשים שיוצאים כולן מאותה נקודה. ומנקודת התחלת המשולשים יצרנו מלבן(2 משולשים) לקרקע שבתמונה.



יצרנו ענן בעזרת 3 ספרות. כאשר הן אחת על השנייה ויש עליהן תאורה סביבת שנותנת להן מראה תלת ממד. בכל ענן יצרנו ספרה יחסית גדולה יותר באמצע ושני ספרות קטנות יותר זהות לשני צידיה, כאשר המרחק בין כל ספרה צדדית מהאמצעית זהה.

יצירת גופים בעזרת INTELLJ

```
22 usages  noharel *
void buildBubble(Point point, double radius){
    scene4.geometries.add(
            new Sphere(point, radius)
                    .setEmission(new Color( r: 255,  g: 255,  b: 255).scale( k: 0.05))
                    .setMaterial(new Material()
                            .setkD(0.1)
                            .setkS(0.9)
                            .setnShininess(300)
                            .setkT(0.8)
                    )
    );
}
```

יצרנו פונקציה שיוצרת בועות סבון כדי לא לחזור על אותה פעולה פעמים רבות. הפונקציה קיבלה רדיוס ונקודה למרכז הבועה והוסיפה בועת סבון לסצנה לפי פרמטרים אלו.

```
13 usages  new *
void buildBush(Point point, double radius, Color color){
    scene4.geometries.add(
            new Sphere(point, radius).setEmission(color)
    );
}
```

יצרנו פונקציה שיוצרת שיח כדי לא לחזור על אותה פעולה פעמים רבות. הפונקציה קיבלה רדיוס, נקודה למרכז השיח וצבע לשיח והוסיפה שיח לסצנה לפי פרמטרים אלו.

```
12 usages  new *
void buildCloud(Point p1, Point p2, Point p3, double rBig, double rSmall){
    scene4.geometries.add(
            new Sphere(p1, rBig)
                    .setEmission(new Color(gray)) //
                    .setMaterial(new Material().setkD(0.54).setkS(0.003).setnShininess(100).setkT(0.15)),
            new Sphere(p2, rSmall)
                    .setEmission(new Color(gray)) //
                    .setMaterial(new Material().setkD(0.54).setkS(0.003).setnShininess(100).setkT(0.15)),
            new Sphere(p3, rSmall)
                    .setEmission(new Color(gray)) //
                    .setMaterial(new Material().setkD(0.54).setkS(0.003).setnShininess(100).setkT(0.15))
    );

}
```

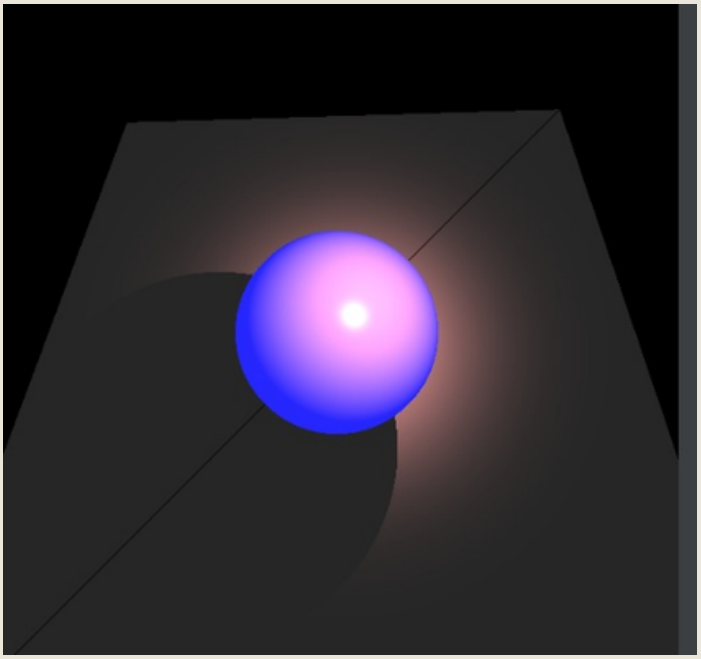
יצרנו פונקציה שיוצרת ענן כדי לא לחזור על אותה פעולה פעמים רבות. הפונקציה קיבלה 3 נקודות, רדיוס לכדור האמצעי בענן, ורדיוס לכדורים הצדדיים בענן והוסיפה ענן לסצנה לפי פרמטרים אלו.

```
3 usages   new *
void buildLittleBalls(Point point){
    scene4.geometries.add(
            new Sphere(point,  radius: 2)
                    .setEmission(new Color( r: 139,  g: 69,  b: 19))
                    .setMaterial(new Material().setkD(0.5).setkS(0.5).setnShininess(100))
    );

}
```

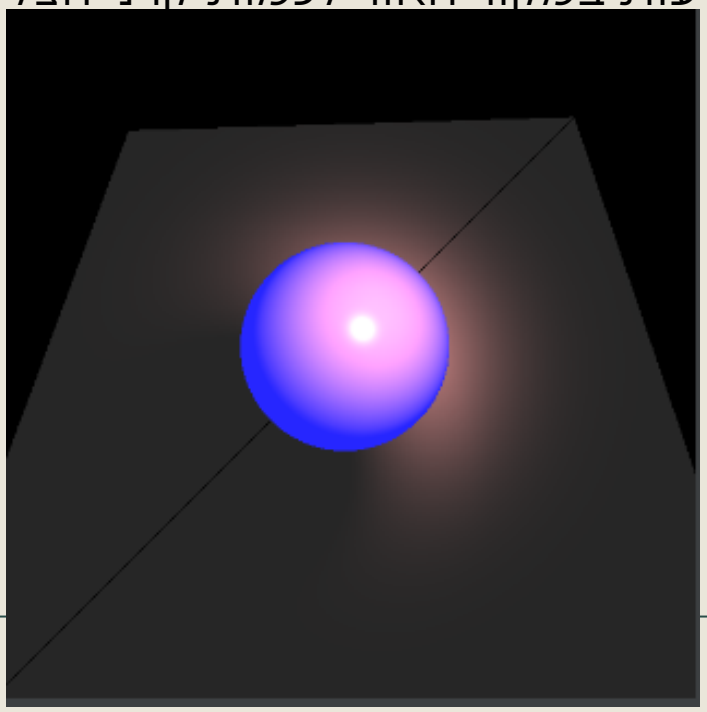
יצרנו פונקציה שיוצרת את הכדור עבור שקיות חול לסלסלה של הכדור פורח כדי לא לחזור על אותה פעולה פעמים רבות. הפונקציה קיבלה רדיוס ונקודה למרכז הכדור והוסיפה הכדור לסצנה לפי פרמטרים אלו.

תיאור-SOFT SHADOWS

הבעיה: הצל בתמונות נראה לא אמיתי מכיוון שהוא נראה כצל קשה ובעל מרקם אחיד בכל נקודה
שמוצלת על ידי הגוף.



פתרון: במקום להתייחס למקור אור כנקודה אחת במרחב נתייחס למקור האור ככדור בעל מספר
נקודות במרחב. כעת נשלח מספר קרני צל אל מספר אזורים שונים במקור האור ואחוז התאורה בכל נקודה יהיה היחס בין מספר הפגיעות במקור האור לכמות קרני הצל שיצרנו.



מימוש-SOFT SHADOWS

בקוד המסופק, ביצענו מימוש לפתרון צלליות רכות באמצעות טכניקה הנקראת "distributed ray tracing" או "soft shadow mapping".
ראשית הוספנו שדה size למקורות התאורה הנקודתיות שמתאר עד כמה ההצלה רכה, השדה אוטומטית מאותחל ל0 במקרה והצל לא רך בכלל. ובנוסף הוספנו לקרן שדה numOfRays שמייצג את כמות הקרניים שמשומשות. ככל שהכמות יותר גדולה פחות רואים את ההבדל בפיקסלים בהצללה.

המתודה "calcLocalEffects" מחשבת את ההשפעות המקומיות של התאורה עבור נקודה נתונה על משטח. היא עוברת על כל מקורות האור בסצנה ומבצעת את החישובים הנדרשים.

כאשר בבדיקה נתקלים במקור אור, בודקים האם מקור האור וכיוון הצפייה נמצאים באותה צד של המשטח. אם הם נמצאים באותו צד, נמשיך לחישובי הצלליות.

בתוך המתודה "calcLocalEffects", יש חלק שמתמודד עם סוגים שונים של מקורות אור. עבור מקורות אור שאינם מקורות אור נקודתיים (תאורה כיוונית), או מקורות אור נקודתיים עם גודל אפס (מיוצגים כמקורות אור נקודתיים בעלי גודל אפס, כלומר שלא רוצו הצללה רכה, כי אם לא משנים את הערך זה אוטומטית 0), נחשב את השקיפות באמצעות המתודה "transparency" באופן ישיר.
אך, במקרה של מקורות אור נקודתיים עם גודל לא אפס נכנס למתודה "calcLocalEffectsSoftShadows".

```
// Determine the type of light source and calculate transparency accordingly
if (lightSource instanceof DirectionalLight || (lightSource instanceof PointLight && ((PointLight) lightSource).getSize() == 0)) {
    ktr = transparency(gp, lightSource, l, n);
} else {
    ktr = calcLocalEffectsSoftShadows((PointLight) lightSource, gp, n, nv, v, material);
}
```

בתוך "calcLocalEffectsSoftShadows", יצרנו אובייקט מחודד (Blackboard) לניהול הקרניים עבור צללים רכים. הלוח מייצג טבלה מדומה המסייעת בתפקוד רכיבי הקרניים השונים על מנת לפזר את הקרניים באופן שווה במקור האור

עשינו מעבר על הקרניים באמצעות לולאות מקוננות, באמצעות המתודה "constructRay" שבמחלקת "Blackboard".

אם קרן תקינה נוצרה (לא null), נגדיל את מונה הקרניים "rays" ונוסיף את ערך השקיפות של נקודת החיתוך לערך "ktr" המצטבר.

לאחר סיום הלולאה על הקרניים, נפחית את ערכי השקיפות המשולבים על ידי חילוק שלהם בערך "rays" כדי לקבל ערך השקיפות ממוצע. זה מייצג את האפקט של הצללה רכה.

לבסוף, נחזיר את הערך "ktr" (השקיפות הממוצעת) שחושב במתודה "calcLocalEffectsSoftShadows".

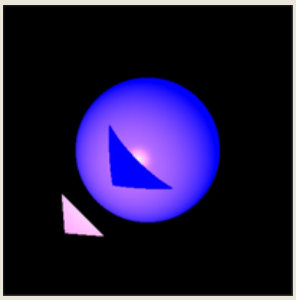
מימוש-SOFT SHADOWS

```java
private Double3 calcLocalEffectsSoftShadows(PointLight lightSource, GeoPoint gp, Vector n, double nv, Vector v, Material material) {
    Vector l = lightSource.getL(gp.point);
    Blackboard blackBoard = new Blackboard((int) Math.round(Math.sqrt(numOfRays)),
            lightSource.getSize(),
            l,
            new Vector( x: 0, -l.getZ(), l.getY()),
            lightSource.getPosition());
    Ray ray;
    int rays = 0;
    Double3 ktr = Double3.ZERO;

    // Iterate over the rays for soft shadows
    for (int i = 1; i < Math.round(Math.sqrt(numOfRays)); i++) {
        for (int j = 1; j < Math.round(Math.sqrt(numOfRays)); j++) {
            // Construct a ray from the blackboard for the current iteration
            ray = blackBoard.constructRay(j, i, gp.point);
            if (ray != null) {
                rays++;
                // Add the transparency value of the intersection point to the accumulated ktr value
                ktr = ktr.add(transparency(gp, lightSource, ray.getDir(), n));
            }
        }
    }

    // Reduce the combined transparency values by the number of rays to obtain the average transparency
    ktr = ktr.reduce(rays);

    return ktr;

}
```

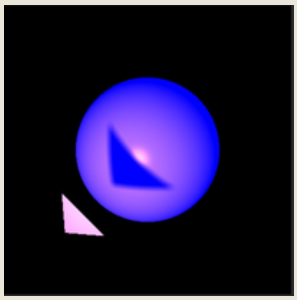
על ידי הכללת השלבים הנ"ל, ביצענו מימוש שיוצר קרניים רבות עבור מקור אור נקודתי כדי להדמות הצללה רכה.

תוצאות-SOFT SHADOWS

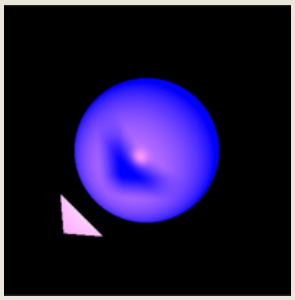
השפעת הפרמטר size על ההצללה הרכה:



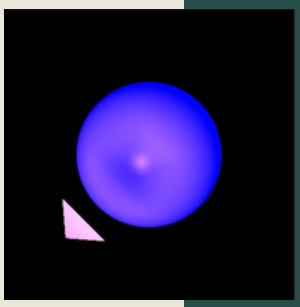
size=0



size=10

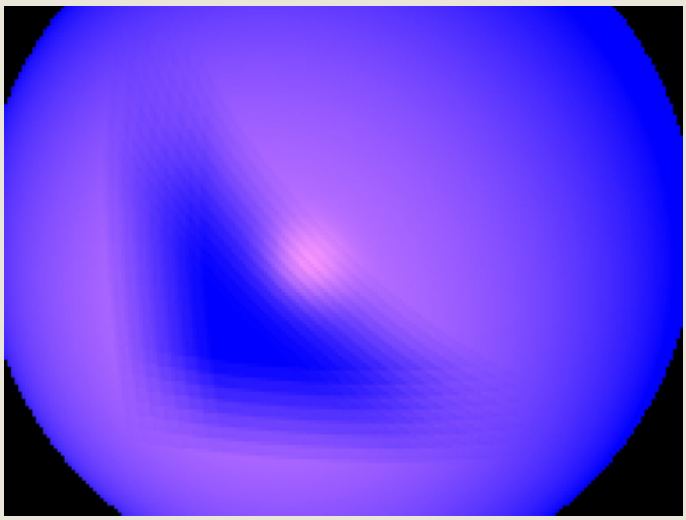


size=30

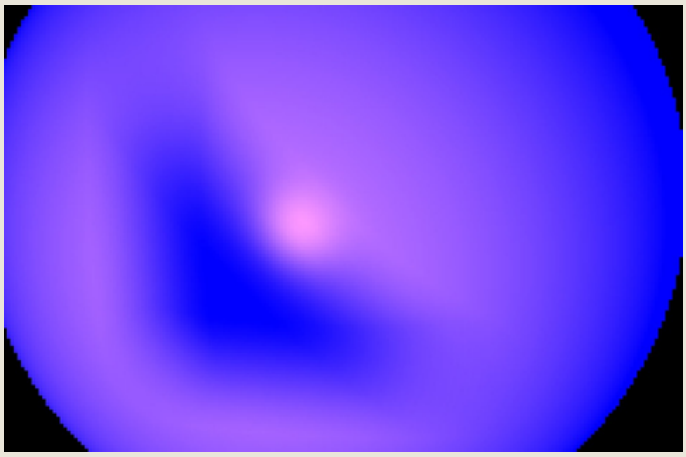


size=60

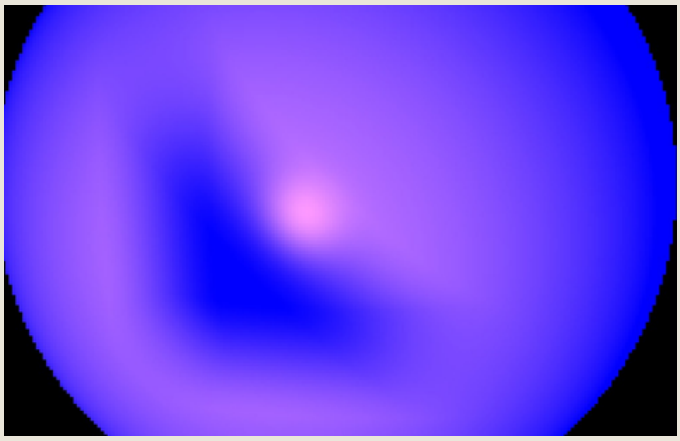
השפעת הפרמטר numOfRays על ההצללה הרכה:



numOfRays=80



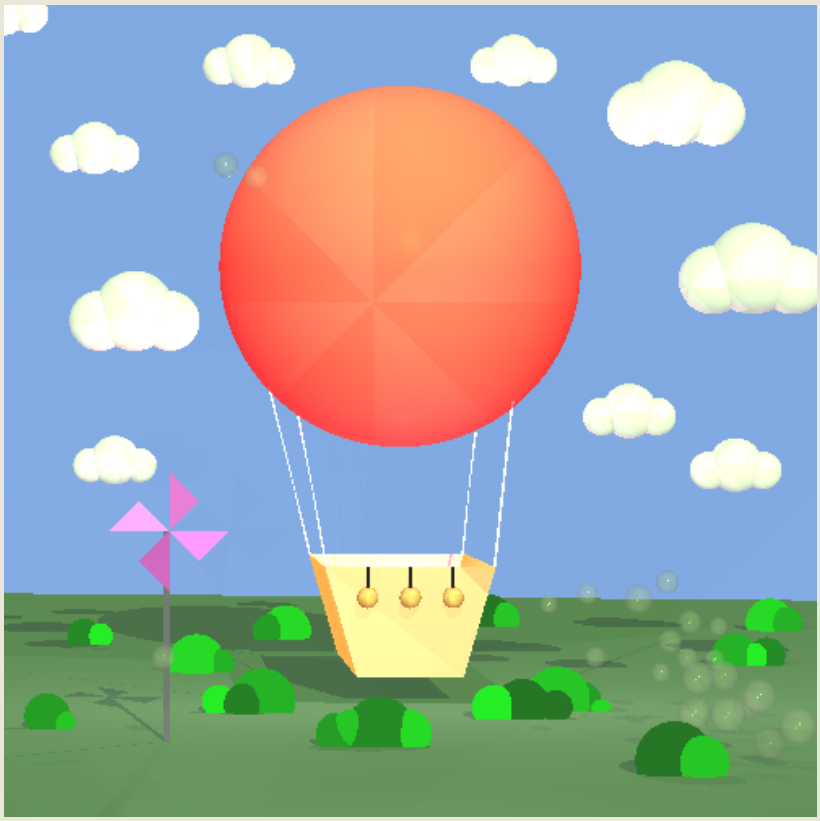
numOfRays=1000



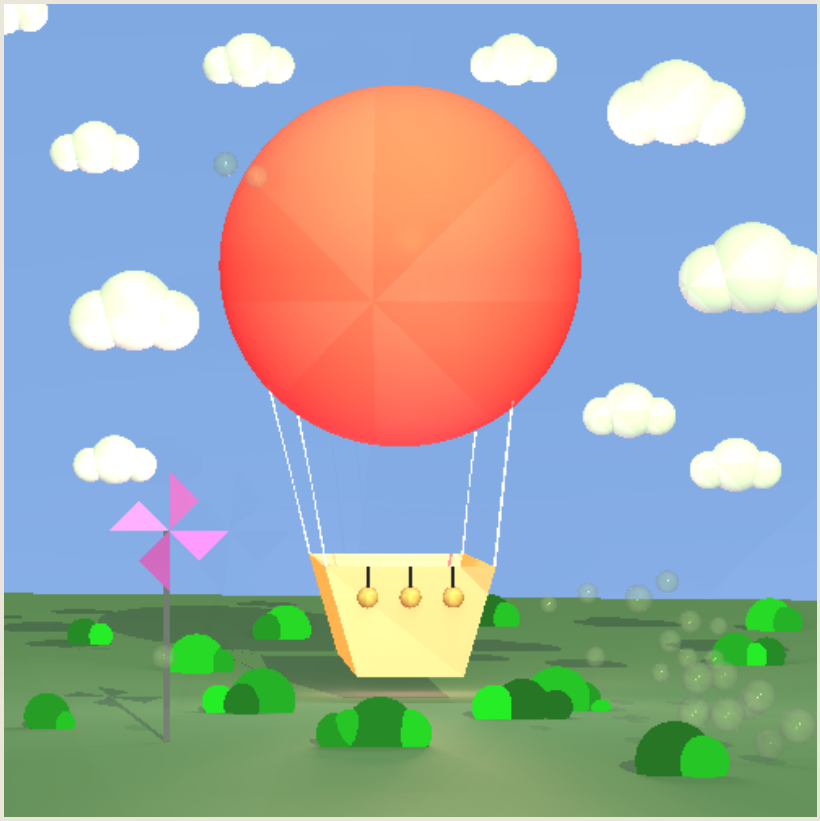
numOfRays=3000

SOFT SHADOWS-תוצאות

השפעת ההצללה הרכה על התמונה שלנו:



לפני



אחרי

הבעיה: עד עכשיו לא ניצלנו את יכולות המעבד עד הקצה כי הרצנו את התמונות בתהליך אחד, דבר שעיקב את הריצה. כעת ברצוננו לשפר את זמני הביצוע בריצה.

הפתרון: על כן נפריד את התהליך בעת ריצתו לתהליכונים שונים ונפרדים אשר כל אחד צובע פיקסלים אחרים ולא חופפים. כדי לוודא שהתהליכונים באמת צובעים פיקסלים שונים ולא עובדים על אותו פיקסל - יצרנו את המחלקה פיקסל שבאחריותה לבחור את הפיקסל הבא שיעוצב תוך כדי שמירה על התהליכונים השונים שלא יצרו בעיות ויעבדו במקביל בלי חפיפות.

תיאור-MULTI THREADING

הפונקציה העיקרית במימוש במחלקה pixel היא הפונקציה הבאה - nextPixel.
הפונקציה דואגת לתת לכל תהליך שמבקש את הפיקסל הבא שפנוי לעיצוב תוך כדי שמירה על סנכרון ומניעת בעיות.

```
1 usage   new *
public boolean nextPixel() {
    synchronized (mutexNext) {
        if (cRow == maxRows)
            return false;
        ++cCol;
        if (cCol < maxCols) {
            row = cRow;
            col = cCol;
            return true;
        }
        cCol = 0;
        ++cRow;
        if (cRow < maxRows) {
            row = cRow;
            col = cCol;
            return true;
        }
        return false;
    }
}
```

לאחר מכן התאמנו את הפונקציה renderImage במחלקת camera לרנדור הנעשה על ידי מספר תהליכונים.

מימוש-MULTI THREADING

```
    //if not using multi threads
    if (threads < 1) {
        //goes through every pixel in view plane  and casts ray, meaning creates a ray for every pixel and sets the color
        for (int row = 0; row < nY; row++) {
            for (int column = 0; column < nX; column++) {
                castRay(nX, nY, row, column);
            }
        }
        return imageWriter;

    }
    //if using multi threads
    Pixel.initialize(nY, nX,  interval: 1);
    while (threads-- > 0) {
        new Thread(() ->
        {
            for (Pixel pixel = new Pixel();
                 pixel.nextPixel();
                 Pixel.pixelDone()) {
                imageWriter.writePixel(pixel.col, pixel.row, castRay(nX, nY, pixel.row, pixel.col));
            }
        }).start();
    }
    Pixel.waitToFinish();
    return imageWriter;
}
```

אם מספר התהליכונים קטן מ-1, זאת אומרת הריצה נעשת ללא השיפור - על ידי תהליך אחד ולא על ידי פיצול העבודה למספר תהליכונים, אז הפונקציה תפעל ללא שינוי.

במידה ומספר התהליכונים גדול מ-1, זאת אומרת הריצה נעשת עם השיפור והעבודה מתבצעת על ידי מספר תהליכונים - אז הפונקציה מחלקת את העבודה בין מספר התהליכונים שהתבקשה.

הבעיה – מכיוון שהסצנה מורכבת מגופים רבים, חלקם מאוד קטנים או מורכבים, נקבל שכל קרן בודקת
חיתוכים עם כל הגופים למרות שעבור רוב המקרים נקבל שאין חיתוך.

הפתרון – ניצור היררכיית גופים תוחמים, המוגדרים כקופסאות סביב האובייקטים, ובכל פעם נבדוק האם
הקרן נחתכת איתם)חישוב פשוט ומהיר לעומת החישוב המלא(. אם לא, נעצור את הבדיקה, ואם כן, נמשיך
מטה בהיררכיה לבדוק האם הקרן נחתכת עם הגופים עצמם.

BVH-תיאור

מימוש-BVH

נוסיף שני שדות למחלקה האבסטרקטית Intersectable שני שדות: bvh – מקבל ערך true כאשר השיפור דולק עבור הגאומטריה הזו, ושדה box עבור הקופסה התוחמת, מסוג BoundingBox, מחלקה פנימית חדשה בתוך Intersectable:

```
/**
 * class representing boundary box
 */
5 usages   new *
public class BoundingBox {
    7 usages
    public Point _minimums;
    7 usages
    public Point _maximums;

    /**
     * Constructs a bounding box with the specified minimum and maximum points.
     *
     * @param minimums The minimum point of the bounding box.
     * @param maximums The maximum point of the bounding box.
     */
    4 usages   new *
    public BoundingBox(Point minimums, Point maximums) {
        _minimums = minimums;
        _maximums = maximums;
    }
}
```

בנוסף, נוסיף פונקציה עבור חישוב חיתוכים עם קופסה תוחמת:

```
/**
 * return true if ray intersects object
 *
 * @param ray ray to check
 * @return whether ray intersects box
 * code taken from scratchapixel.com
 * https://www.scratchapixel.com/lessons/3d-basic-rendering
 * /introductionSTXacceleration-structure/bounding-volume-hierarchy-BVH-part1
 */
1 usage   new *
public boolean intersectingBoundingBox(Ray ray) {
    if (!BVH || box == null)
        return true;
    Vector dir = ray.getDir();
    Point p0 = ray.getP0();

    // Calculate the intersection intervals on the x-axis
    double xMin = (box._minimums.getX() - p0.getX()) / dir.getX();
    double xMax = (box._maximums.getX() - p0.getX()) / dir.getX();

    // Ensure xMin is smaller than xMax
    if (xMin > xMax) {
        double temp = xMin;
        xMin = xMax;
        xMax = temp;
    }

    // Calculate the intersection intervals on the y-axis
    double yMin = (box._minimums.getY() - p0.getY()) / dir.getY();
    double yMax = (box._maximums.getY() - p0.getY()) / dir.getY();
```

מימוש-BVH

```java
// Ensure yMin is smaller than yMax
if (yMin > yMax) {
    double temp = yMin;
    yMin = yMax;
    yMax = temp;
}

// Check for non-overlapping intervals on the x-axis and y-axis
if ((xMin > yMax) || (yMin > xMax))
    return false;

// Update xMin to the maximum of yMin and xMin
if (yMin > xMin)
    xMin = yMin;

// Update xMax to the minimum of yMax and xMax
if (yMax < xMax)
    xMax = yMax;

// Calculate the intersection intervals on the z-axis
double zMin = (box._minimums.getZ() - p0.getZ()) / dir.getZ();
double zMax = (box._maximums.getZ() - p0.getZ()) / dir.getZ();

// Ensure zMin is smaller than zMax
if (zMin > zMax) {
    double temp = zMin;
    zMin = zMax;
    zMax = temp;
}


    // Check for non-overlapping intervals on the x-axis and z-axis
    if ((xMin > zMax) || (zMin > xMax))
        return false;

    // Update xMin to the maximum of zMin and xMin
    if (zMin > xMin)
        xMin = zMin;

    // Update xMax to the minimum of zMax and xMax
    if (zMax < xMax)
        xMax = zMax;

    return true;
}
```

בנוסף, נוסיף פונקציה אבסטרקטית שכל מחלקה שתירש מ-Intersectable תצטרך לממש:

```java
/**
 * create the boundary box for the objects
 */
7 usages   6 implementations   new *
public abstract void createBoundingBox();
```

מימוש-BVH

כעת נוסיף בכל בנאי של גאומטריה חוץ ממישור וגליל אינסופי וגליל את השורות הבאות:

```
//if bvh improvement is used
if (BVH){
    //create bounding box
    createBoundingBox();
}
```

לא הוספנו לגליל האיסופי ולמישור מכיוון שהם אינסופיים ולכן לא נוכל לחסום אותם בתוך גוף אחר. ולגליל גם לא הוספנו משום שלא השלמנו את מימושו ולכן לא השתמשנו בו בשום מקום.

ונשנה את המימוש של findGeoIntersections בתוך Intersectable:

```
/**
 * Finds the intersection points between this object and a given ray, in the form of GeoPoints.
 *
 * @param ray The ray to intersect with.
 * @return A list of GeoPoints representing the intersection points, or null if there are no intersections.
 */
3 usages  Eliezer *
public final List<GeoPoint> findGeoIntersections(Ray ray) {

    if (BVH && !intersectingBoundingBox(ray))
    {
        return null;
    }
    return findGeoIntersectionsHelper(ray);
}
```

כלומר, כעת נחשב את החיתוך של הקרן עם הגאומטריה רק אם היא חותכת את הקופסה התוחמת. אחרת, נדע ישר להחזיר NULL.

שיפורי זמני ריצה-תוצאות

זמן ריצה	רזולוציה	Multii threading	BVH	Soft shadow
1 דקות 57 שניות	1000x1000	✗	✗	✗
18 דקות 15 שניות	3000x3000	✗	✗	✗
42 שניות	1000x1000	✓	✗	✗
6 דקות 4 שניות	3000x3000	✓	✗	✗
3 דקות 40 שניות	1000x1000	✗	✓	✗
59 שניות	1000x1000	✓	✓	✗
4 שעות 27 דקות	500x500	✗	✗	✓
שעה 54 דקות	500x500	✓	✓	✓

בונוסים:

שלב 2: מימוש הפונקציה getNormal עבור גליל אינסופי

שלב 6: ביצוע תוך שבוע