מגישים:
נעם שירה
ואברהם בן יקר

בס"ד

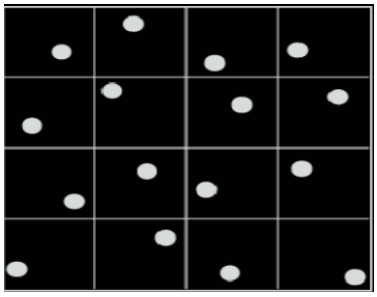
דו"ח פרויקט

- כל הפונקציות מחזירות פרמטרים חדשים תמיד – כולל הפונקציות של חיבור/חיסור, או הכפלה וקטורית (למעט נירמול שמתבצע על האיבר עצמו) .
- יצירת setters רק כשהיה חייב (כמובן שלכל משתנה יצרנו getter)
- השתדלנו להשתמש בשמות משתנים כמו במבנה המצגת של ד"ר אלישי
- שיפו של הצל לצל רך (soft shadow)
- הוספת ממשק של אורות בעלי שטח (שאינו אינסופי) לשם מימוש השיפור של הצל הרך

הסבר על השיפור של צל רך

השיטה בה עבדנו היא לירות מכל פיקסל אותו בודקים מספר קרניים לשטח מקור האור ולפיהם לחשב את רמת הצל. לשם הנוחות החלטנו להשתמש רק במקורות אור מרובעים. תחילה מחליטים על מספר קרניים שבהם נשתמש לדגימת רמת הצל, המספר צריך להיות שורש ריבועי של מספר שלם כדי לאפשר חלוקה נוחה של שטח פנים מרובע לחלקים שווים, לכן הוגדר בקוד אופציה להכניס את השורש הריבועי של מספר הקרניים כאשר ברירת המחדל היא 9 (9X9 = 81 קרניים). בנוסף הוספנו למחלקות האור הנקודתי ו spot light שדה שמכיל את גודל הצלע של מקור האור, שכן בריבוע כל הצלעות שוות ולכן מספיק גודל הצלע. שני שדות נוספים שנוספו הם השדות של וקטורים המשיקים למקור האור, U ו- V, כדי לדעת האופן בו מקור האור ניצב במרחב. כדי לאפשר שימוש בשדות אלו יצרנו ממשק חדש היורש מהמשק של מקורות אור שנקרא AreaLightSource שנועד לאפשר גישה לגודל האור הווקטורים המשיקים לו וכן לנקודה על האור (הפונקציה של הנקודה על האור מחזירה את תוכן השדה position שהיה קיים עוד קודם). החישוב של הצל מתבצע על ידי חלוקת מקור האור לריבועים לפי מספר הקרניים שאנו שולחים, בכל ריבוע להגריל בצורה אקראית נקודה אליה תשלח הקרן (השיטה של Monte Carlo)

תמונה להמחשה:



לכל קרן שנשלחת בודקים האם מגיעה למקור האור או שהיא נחסמת על ידי גוף כלשהו לפני כן, כאשר קרן אשר מצליחה להגיע למקור האור מקבלת את הערך 1 וקרן של הגיע מקבלת את הערך של 0. בנוסף לשליחת כל הקרניים בשביל הבדיקות נשלחת גם קרן למרכז האור כמו שנעשה לפני ההוספה של הצל הרך. לאחר שליחת כל הקרניים מחברים את הערכים של כל הקרניים ומחלקים במספר הקרניים שנשלחו וכך מקבלים את הערך הממוצע של הצל, שזהו מספר הנע בין 0 ל1, מספר זה ישמש כמכפיל לערך האור

בפיקסל הנתון וכך יתקבלו רמות צל שונות לפי המיקום של הפיקסל. חישוב זה נעשה כולו במחלקת RayTracerBasic, אשר מחשבת את הקרניים שצריכים לירות, בודקת האם ניתן לראות את קרן האור וחישוב הממוצע של הצל בנקודה. למחלקת RayTracerBase נוסף שד המכיל את כמות הקרניים שיוצרו עם setter תואם וכן במחלקת render נוסף setter שמשנה את התוכן של שדה זה בray tracer שהוא מחזיק בשדה שלו.

כדי לאפשר כיבוי והדלקה של התוספת הזו נוספו למחלקת RayTracerBase שדה בוליאני השולט על שימוש ואי שימוש באפשרות זו. נוסף setter לשדה זה למחלקת RayTracerBase וכן למחלקת render ששולט על הערך של הray tracer המוכל בתוכו.

כדי לאפשר שימוש ואי שימוש באפשרות זו וכדי להימנע מחזרה על קטעי קוד, נוספה למחלקת RayTracerBasic פונקציה פנימית visibility שמקבלת את המיקום של הפיקסל, הקרן לכיוון מקור האור, והמרחק של הנקודה ממקור האור והיא מחזירה מספר בין 0 ל1 בהתאם לנראות או האי נראות של הקרן. הפונקציה transparency בודקת האם משתמשים ב צל רך והאם מקור האור הוא בעל שטח (במקורות אור ללא שטח אין הבדל בין צל רך לצל הרגיל) אם צריך להשתמש בצל רך הפונקציה שולחת לפונקציה אחרת המחשבת צל רך ואם לא אז transparency מחזיר את תוצאת החישוב של visibility. כאשר אין שימוש בצל רך, אין התייחסות לגודל של מקור האור כלל.

מראה הפונקציות במחלקת RayTracerBasic

```
private double transparency(LightSource light, Vector l, Vector n, GeoPoint geopoint) {
    Vector lightDirection = l.scale(-1); // from point to light source
    if (softShadows && light instanceof AreaLightSource)
        return calcSoftShadows((AreaLightSource) light, lightDirection, n, geopoint);
    Ray lightRay = new Ray(geopoint.point, lightDirection, n);//add delta
    double lightDistance = light.getDistance(geopoint.point);
    return Visibility(lightDistance, lightRay, geopoint);
}

private double Visibility(double lightDistance, Ray lightRay, GeoPoint geopoint) {
    List<GeoPoint> intersections = _scene.geometries.findGeoIntersections(lightRay);
    if (intersections == null) return 1.0;
    double ktr = 1.0;
    for (GeoPoint gp : intersections) {
        if (alignZero( number: gp.point.distance(geopoint.point) - lightDistance) <= 0) {
            ktr *= gp.geometry.getMaterial().kT;
            if (ktr < MIN_CALC_COLOR_K) return 0.0;
        }
    }
    return ktr;
}
```

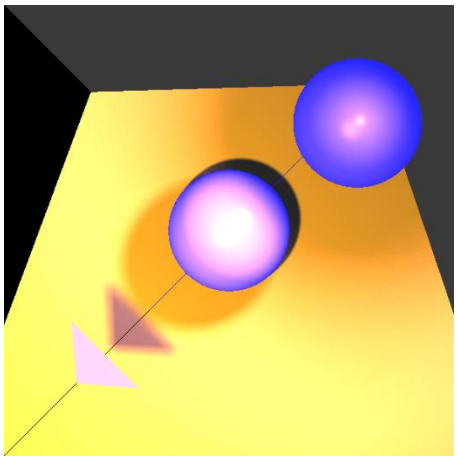
```java
private double calcSoftShadows(AreaLightSource light, Vector lightDirection, Vector n, GeoPoint geopoint) {
    //calc ther visibility of the basic ray from the origin point
    Ray lightRay = new Ray(geopoint.point, lightDirection, n);
    double lightDistance = light.getDistance(geopoint.point);
    double ktr = Visibility(lightDistance, lightRay, geopoint);
    //calc the sample rays and add the result of every ray to ktr
    Point3D sample;
    double size = light.getEdges(); //the size of edge of the light
    //create random points on the light
    //calculate by divide the light area to squares and add choose random point in the square
    //the number of the square are _sqrtBeamNum*_sqrtBeamNum
    double rand[][] = new double[_sqrtBeamNum * _sqrtBeamNum][2];
    double div = 1 / (double) _sqrtBeamNum;
    for (int i = 0; i < _sqrtBeamNum; i++) {
        for (int j = 0; j < _sqrtBeamNum; j++) {
            rand[i * _sqrtBeamNum + j][0] = Util.random(i * div, (i + 1) * div);
            rand[i * _sqrtBeamNum + j][1] = Util.random(j * div, (j + 1) * div);
        }
    }
    //construct ray to every random point and calculate its visibility
    Point3D center = light.getCenter();
    Vector v = light.getV();
    Vector u = light.getU();
    for (int i = 0; i < _sqrtBeamNum * _sqrtBeamNum; i++) {
        sample = center.add(u.scale((0.5 - rand[i][0]) * size).add(v.scale((0.5 - rand[i][1]) * size)));
        lightRay = new Ray(geopoint.point, sample.subtract(geopoint.point), n);//add delta to the shadow ray
        lightDistance = geopoint.point.distance(sample);
        ktr += Visibility(lightDistance, lightRay, geopoint);
    }
    //the soft shadows calculate by average the results (the +1 is the original ray)
    return ktr / ((_sqrtBeamNum * _sqrtBeamNum) + 1);
}
```

תמונות להמחשה

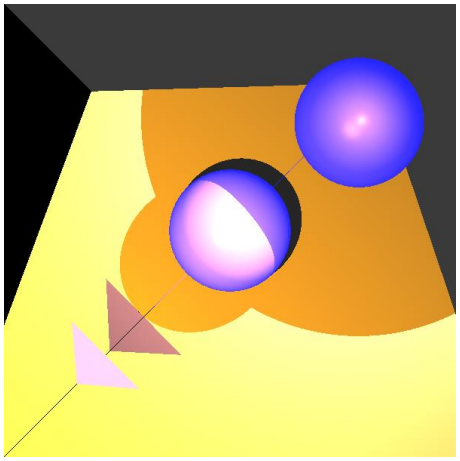
להמחשת התוצאה של הצל הרך נוסיף 2 דוגמאות להשפעה:

דוגמא א':

עם צל רך:

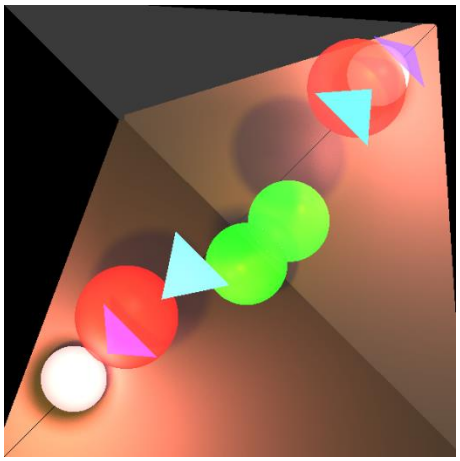


ללא צל רך:



דוגמא ב':

עם צל רך:



ללא צל רך: