דו"ח מיני פרוייקט

בפרויקט יוצרים תמונה תלת מימדית מצורות גיאומטריות, אורות, צבעים ומצלמה.
בהתחלה מימשנו צורות פרימיטיביות (נקודה תלת מימדית, קרן, וקטור, צבע, חומר וכו'). אחר כך מימשנו צורות גיאומטריות (משולש, ספירה, מישור וכו'). המשכנו לבנות מחלקה שתייצג מצלמה, ובנינו מחלקות עזר לרינדור התמונה (ray tracer basic, render, scene וכו').
לאורך הפרויקט מימשנו עקרונות TDD, נמנענו מsmells ועשינו כל כמה זמן refactoring.

קישור לגיט:
https://github.com/yiskalevi/ISE5782_4622_5790

מגישות:
יסכה לוי 213114622
ליאל תהילה שמחי 213485790

התמונה שלנו

מה בתמונה? :)

אנחנו בחרנו לעשות חדר שישלב אפקטים וצורות שונות. בחדר יש מסלול של כדורים בכל מיני צבעים, גדלים וחומרים שונים כדי לאפשר לצופה לראות את האפקטים השונים והשפעת התאורה על כל סוג אחר של כדור. בקיר שממול הצופה (כמובן שניתן להזיז את המצלמה ולראות את התמונה מכיוון שונה אבל הכוונה לתנוחה שממנה לקחנו את התמונה הסופית) יש קיר מראה שמשקף את החדר ותכולתו. ניתן גם לראות דרכה את הקיר שמאחורי המצלמה. על הקירות בצדדים יש מנורות עגולות (spot light) שמאירות לתוך החדר. אפשר להבחין שעל כל כדור ישנם כמה צללים כתוצאה של השפעת מקורות האור השונים עליו. בקיר השמאלי יש גם דלת פתוחה לכיוון החוצה שדרכה נכנס אור השמש (direction light) המאיר על הכדורים שמול הפתח.

הקיר הימני בנוי מ2 משולשים אטומים בצבע בורדו:

```
Geometry rightWallTri1 = new Triangle(new Point(90,-50,-500),new Point(90,100,-500),new Point(90,-50,1300))
        .setEmission(new Color(RED).reduce(1.2));
Geometry rightWallTri2 = new Triangle(new Point(90,100,1300),new Point(90,100,-500),new Point(90,-50,1300))
        .setEmission(new Color(RED).reduce(1.2));
```

בדומה לקיר הימני, בנינו גם את התקרה, הרצפה והקיר האחורי.

הקיר השמאלי יותר מורכב כי יש בו דלת והשתמשנו ביותר משולשים:

```
Geometry trWallR1 = new Triangle(new Point(-90, 100, -500), new Point(-90, -50, -500), new Point(-90, 100, 50)).setEmission(new Color(RED).reduce(1.2));
Geometry trWallR2 = new Triangle(new Point(-90, 100, 50), new Point(-90, -50, 50), new Point(-90, -50, -500)).setEmission(new Color(RED).reduce(1.2));
Geometry trWallL1 = new Triangle(new Point(-90, 100, 250), new Point(-90, -50, 250), new Point(-90, 100, 1300)).setEmission(new Color(RED).reduce(1.2));
Geometry trWallL2 = new Triangle(new Point(-90, -50, 250), new Point(-90, 100, 1300), new Point(-90, -50, 1300)).setEmission(new Color(RED).reduce(1.2));
Geometry trDoor1 = new Triangle(new Point(-90, 65, 50), new Point(-90, -50, 50), new Point(-200, 65, 50)).setEmission(new Color(212, 207, 210).reduce(1.2))
        .setMaterial(new Material().setkD(0.6));
Geometry trDoor2 = new Triangle(new Point(-200, 65, 50), new Point(-200, -50, 50), new Point(-90, -50, 50)).setEmission(new Color(212, 207, 210).reduce(1.2))
        .setMaterial(new Material().setkD(0.6));
Geometry trUpDoor1 = new Triangle(new Point(-90, 65, 50), new Point(-90, 100, 50), new Point(-90, 100, 250)).setEmission(new Color(RED).reduce(1.2));
Geometry trUpDoor2 = new Triangle(new Point(-90, 65, 50), new Point(-90, 65, 250), new Point(-90, 100, 250)).setEmission(new Color(RED).reduce(1.2));
```

הקיר-מראה שנמצא ממול גם נבנה ממשולשים, אך מחומר שונה שנותן לו את האפקט של המראה:

```
Geometry frontMirrorWallTri1 = new Triangle(new Point(-90,-50,-500),new Point(-90,100,-500),new Point(90,-50,-500))
        .setMaterial(new Material().setKr(0.8).setkD(0.1)).setEmission(new Color(BLACK));
Geometry frontMirrorWallTri2 = new Triangle(new Point(90,100,-500),new Point(-90,100,-500),new Point(90,-50,-500))
        .setMaterial(new Material().setKr(0.8).setkD(0.1)).setEmission(new Color(BLACK));
```

המנורות spot light שעל הקיר, זהות ובמרחקים שווים:

```
Geometry Lamp1 = new Sphere(new Point(-92, 65, -200), 8d)
        .setEmission(new Color(249, 255, 195))
        .setMaterial(new Material().setKt(0.2).setkD(0.8));
Geometry Lamp2 = new Sphere(new Point(-92, 65, 400), 8d)
        .setEmission(new Color(249, 255, 195))
        .setMaterial(new Material().setKt(0.2).setkD(0.8));
Geometry Lamp3 = new Sphere(new Point(-92, 65, 1000), 8d)
        .setEmission(new Color(249, 255, 195))
        .setMaterial(new Material().setKt(0.2).setkD(0.8));
Geometry Lamp4 = new Sphere(new Point(92, 65, -200), 8d)
        .setEmission(new Color(249, 255, 195))
        .setMaterial(new Material().setKt(0.2).setkD(0.8));
Geometry Lamp5 = new Sphere(new Point(92, 65, 400), 8d)
        .setEmission(new Color(249, 255, 195))
        .setMaterial(new Material().setKt(0.2).setkD(0.8));
Geometry Lamp6 = new Sphere(new Point(92, 65, 1000), 8d)
        .setEmission(new Color(249, 255, 195))
        .setMaterial(new Material().setKt(0.2).setkD(0.8));
```

הכדורים שעל הרצפה צבעוניים, בגדלים שונים ובסוגים שונים. יש כדורים אטומים, יש מבריקים, חלקם משקפים את חלל החדר עליהם בצורה מתעגלת ומיוחדת.

חלק מהקוד של הכדורים:

```
Geometry greenSphere = new Sphere(new Point(-50, -35, -200), 15d)
        .setEmission(new Color(75, 255, 40))
        .setMaterial(new Material().setkD(1));
Geometry blackSphere = new Sphere(new Point(-65, -40, -110), 10d)
        .setEmission(new Color(BLACK))
        .setMaterial(new Material().setkD(0.7).setKr(0.3).setkS(1));
Geometry pinkSphere = new Sphere(new Point(-50, -43, -10), 7d)
        .setEmission(new Color(224, 66, 255))
        .setMaterial(new Material().setkD(1));
```

שיפורי תמונה

soft shadow

ה soft shadows הוא פתרון לבעיה של מעבר חד בין מצב "מוצל" למצב "לא מוצל". במציאות רואים שגבולות הצל הולכים ומתבהרים עד שמטשטשים לחלוטין, כלומר, צל של אובייקט לא כולו באותה רמת הצללה.

הפתרון ש soft shadows מממש הוא ליצור אפקט של צל הדרגתי שמושפע מהמרחק של מקור האור מהאובייקט (כמו שרואים במציאות).

יישמנו את זה על ידי התייחסות למקור האור כאוסף של נקודות אור סמוכות ובדיקה עבור כל אחת האם קרן האור נחסמת בדרך לגוף או שלא. סכמנו את המקדמים של כולם ולבסוף חילקנו במספר הקרניים וזה היה המקדם לצבע של הפיקסל בהשפעת מקור האור.

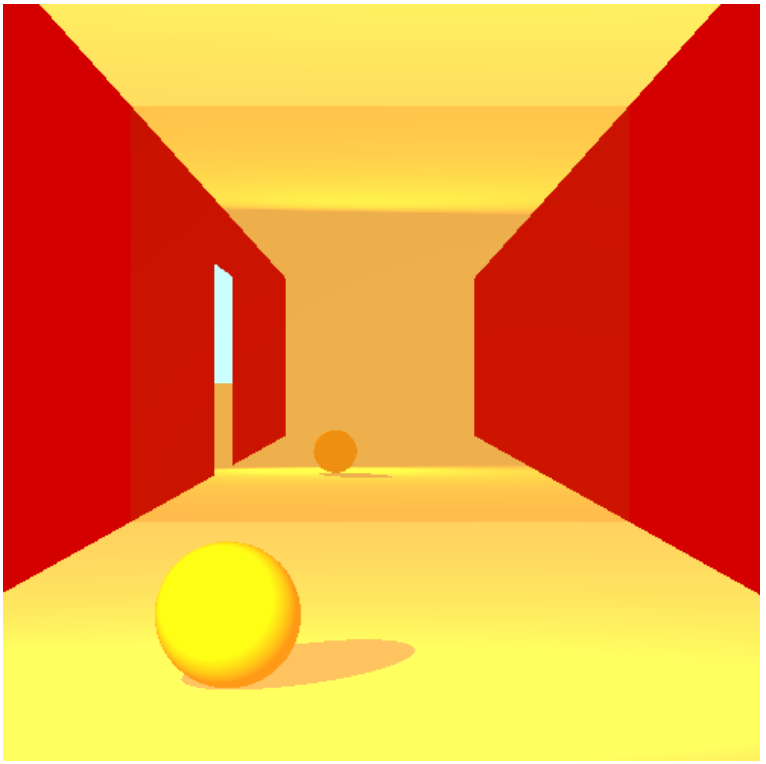
```
private Double3 transparencyWithSoftShadows(GeoPoint geoPoint, LightSource lightSource, Vector n) {
    Double3 ktr;
    List<Vector> beamL = lightSource.getListRound(geoPoint.point,  beamRadius: 5,  SsRayCounter: 5);
    Double3 tempKtr = Double3.ZERO;
    for (Vector vl : beamL) {
        tempKtr = tempKtr.add(transparency(geoPoint, lightSource, vl, n));
    }
    ktr = tempKtr.reduce(beamL.size());

    return ktr;
}
```

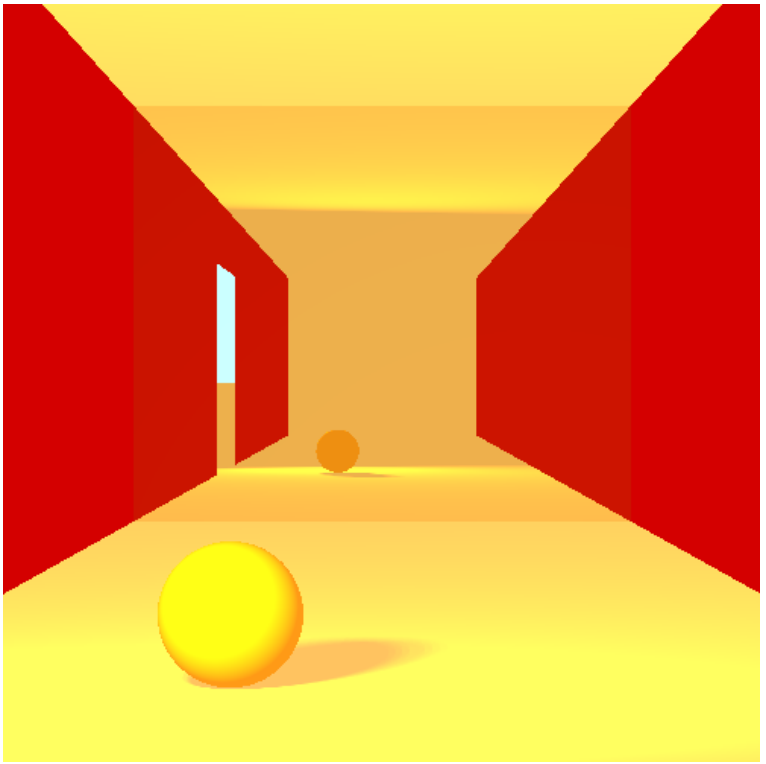
הפונקציה transparencyWithSoftShadows קוראת לפונקציה getListRound שמחזירה אוסף וקטורים ברדיוס מסויים מהנקודה המקורית לכיוון מקור האור שעבור כל אחד - פועלת כמו transparency - מקבלת את מקדם הצל. בסוף היא סוכמת את המקדמים של כולם וכך מקבלת את רמת הצל בפיקסל.

תמונות עם השיפור ובלי השיפור בעמוד הבא.

בלי שיפור:



עם שיפור:



anti aliasing

הבעיה היא שביצוג קווים עקומים במחשב במקום לראות קו רצוף אנחנו רואים אותו בצורה של מדרגות. כיוון זה קורה כיוון שבמחשב הכול הוא בייצוג של פיקסלים שצבעו מחושב על ידי הטלת קרן למרכזו וכך נוצרות מדרגות במקום קווים.

עשינו זאת כך שחילקנו כל פיקסל להרבה פיקסלים קטנים ולכל אחד חישבנו צבע בנפרד. בסוף חיברנו את כל הצבעים שהתקבלו וחילקנו במספר הקרניים וזה היה הצבע הסופי של הפיקסל המקורי.

דבר זה גרם לפתרון הבעיה כיוון שבמקום לבחור לפיקסל צבע בגוון מסויים שבו הקרן נוגעת הצבע הנבחר הוא שילוב צבעים של כמה נקודות באותו פיקסל כך שהם יש שינוי בצבעים בתוך הפיקסל הצבע של הפיקסל יהיה שילוב שני הצבעים האלה וכך יהיה לנו מעבר צבעים יותר עדין.

```
private void castRayRegularAntialiasing(int Nx, int Ny, int i, int j) {
    //improving of anti-aliasing
    int bigNy = 9 * Ny;
    int bigNx = 9 * Nx;
    Color pixelColor = new Color(java.awt.Color.BLACK);
    for (int iColumn = i * 9; iColumn < i * 9 + 9; iColumn++) {
        for (int jRow = j * 9; jRow < j * 9 + 9; jRow++) {

            Ray ray = constructRay(bigNx, bigNy, jRow, iColumn);
            pixelColor = pixelColor.add(rayTracer.traceRay(ray));
        }
    }
    pixelColor = pixelColor.reduce(81);
    imageWriter.writePixel(j, i, pixelColor);
}
```

תמונות עם השיפור ובלי השיפור בעמוד הבא.

בלי השיפור

עם השיפור

שיפורי זמן ריצה

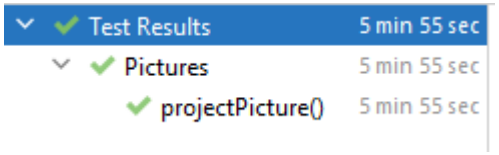
Adaptive Super-sampling

לפעמים ישנם פיקסלים שכול הפיקסל הוא אותו צבע ולכן יש פעולות חישוב מיותרות כי אנחנו סתם שולחים כמה קרנים לאותו הפיקסל ומחשבים למרות שאין שוני בצבעים.

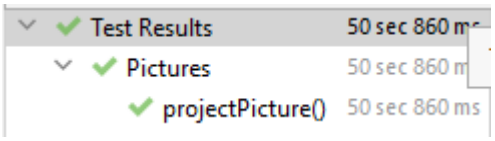
כדי לפתור זאת במקום לחשב את הצבע בכל נקודה חילקנו את הסצנה ל4 חלקים ולכל חלק הטלנו קרן ובדקנו אם הצבעים של הקרנים שונים או שווים לצבע הקרן המקורית לפיקסל.
במקרה שהם שונים חלקנו את אותו קטע לעוד 4 חלקים, כך בצורה רקורסיבית שוב ושוב. ובסוף שקללנו את הצבעים שהתקבלו מהחלוקות וזה היה הצבע הסופי של הפיקסל. כדי למנוע לולאה אינסופית הגדרנו עומק מקסימלי לרקורסיה.
בתחילה הגדרנו שצבע שונה זה צבע שיש בו פער של 5 יחידות. ראינו שזמן הריצה השתפר אבל עדיין איתי לכן שינינו ל10 וראינו שכמעט ולא היה הבדל באיכות אבל זמן הריצה השתפר משמעותית.

```java
private void castRayImprovedAntialising(int Nx, int Ny, int i, int j) {
    //improving of anti-aliasing
    int bigNy = 2 * Ny;
    int bigNx = 2 * Nx;
    Ray middleRay = constructRay(Nx, Ny, j, i);
    Color pixelColor = new Color(java.awt.Color.BLACK);
    pixelColor = pixelColor.add(rayTracer.traceRay(middleRay));
    java.awt.Color c1 = rayTracer.traceRay(middleRay).getColor();
    for (int iColumn = i * 2; iColumn < i * 2 + 2; iColumn++) {
        for (int jRow = j * 2; jRow < j * 2 + 2; jRow++) {
            Ray ray = constructRay(bigNx, bigNy, jRow, iColumn);
            java.awt.Color c2 = rayTracer.traceRay(ray).getColor();
            if (Math.abs(c1.getBlue() - c2.getBlue()) > 10 && Math.abs(c1.getGreen() - c2.getGreen()) > 10 && Math.abs(c1.getRed() - c2.getRed()) > 10)
                pixelColor = pixelColor.add(castRayHelp(Nx, Ny, iColumn, jRow, rayTracer.traceRay(ray)));
            else
                pixelColor = pixelColor.add(rayTracer.traceRay(ray));
        }
    }
    pixelColor = pixelColor.reduce(5);
    imageWriter.writePixel(j, i, pixelColor);
}
```

משך זמן ריצה בלי השיפור (ועם תהליכונים)-



משך זמן הריצה עם השיפור (ועם תהליכונים)-



תהליכונים

שיפור נוסף שעשינו בשביל שזמן הריצה יהיה יותר יעיל זה שהוספנו תהליכונים בפונקציה של render כדי שיחשב כמה פיקסלים במקביל במקום לחשב אחד אחד.

התהליכונים הם משפרי זמן ריצה מעולים ומנצלים ביעילות את המעבד.

```
public Camera renderImage() {
    int Nx = imageWriter.getNx();
    int Ny = imageWriter.getNy();
    Pixel.initialize(Ny, Nx, printInterval);
    IntStream.range(0, Ny).parallel().forEach(i -> {
        IntStream.range(0, Nx).parallel().forEach(j -> {
            if (!antialiasing) {  //build picture without antialiasing improve
                castRaySimple(Nx, Ny, i, j);
                Pixel.pixelDone();
                Pixel.printPixel();
            } else if (antialiasing && adaptiveSuperSampling) {  //build picture with antialiasing improve and run-time improve
                castRayImprovedAntialising(Nx, Ny, i, j);
                Pixel.pixelDone();
                Pixel.printPixel();
            } else { //  //build picture with antialiasing improve
                castRayRegularAntialiasing(Nx, Ny, i, j);
                Pixel.pixelDone();
                Pixel.printPixel();
            }

        });
```

משך זמן ריצה עם תהליכונים-

Test Results	5 min 55 sec
Pictures	5 min 55 sec
projectPicture()	5 min 55 sec

משך זמן ריצה בלי תהליכונים- הרצנו במשך 7 שעות וזה לא גמר, אז עצרנו...

<u>החלפת המישורים במשולשים</u>

בהתחלה קירות החדר, התקרה והרצפה היו כולם מישורים. משיקולי יעילות ושיפור זמן ריצה החלפנו את כולם (מלבד הרצפה) לייצוג ע"י משולשים. בכך מנענו חישובים ארוכים של נקודות חיתוך במישורים האינסופיים שאינן רלוונטיות באמת. וכיוון שבתמונה הרבה אובייקטים זה עלול היה לקחת זמן רב מיותר.

דוגמת קוד של החלפת מישור במשולשים:

```
//      Geometry frontMirrorWall = new Plane(
//              new Point(0, 0, -500),
//              new Vector(0, 0, 1)).setMaterial(new Material().setKr(0.8).setkD(0.1))
//                .setEmission(new Color(BLACK));
        Geometry frontMirrorWallTri1 = new Triangle(
                new Point( x: -92, y: -52, z: -500),
                new Point( x: -92, y: 102, z: -500),
                new Point( x: 91, y: -51, z: -500))
                .setMaterial(new Material().setKr(0.8).setkD(0.1)).setEmission(new Color(BLACK));
        Geometry frontMirrorWallTri2 = new Triangle(
                new Point( x: 92, y: 102, z: -500),
                new Point( x: -92, y: 102, z: -500),
                new Point( x: 91, y: -51, z: -500))
                .setMaterial(new Material().setKr(0.8).setkD(0.1)).setEmission(new Color(BLACK));
```

אופציות להרצת התמונה

הוספנו בקוד משתנים גלובליים בוליאנים המציינים את הגדרות הרצת התמונה- עם\בלי תהליכונים, עם\בלי antialiasing, עם\בלי softshadows, עם\בלי adaptive super sampling.
כשמריצים את התמונה שולחים לפונקציה renderImage את הערכים למשתנים הללו ובהתאם לכך התמונה מרונדרת.
בתוך הפונקציה renderImage הוספנו תנאים שבודקים באיזו צורה לרנדר את התמונה, ולפי זה שולחים לפונקציית castRay המתאימה.

```
static public boolean withTheads = true;
static public boolean antialiasing = true;
static public boolean softShadows = true;
static public boolean adaptiveSuperSampling = true;
```

בתוך renderImage:

```
if (!antialiasing) {  //build picture without antialiasing improve
    castRaySimple(Nx, Ny, i, j);
    Pixel.pixelDone();
    Pixel.printPixel();
} else if (antialiasing && adaptiveSuperSampling) {  //build picture with antialiasing improve and run-time improve
    castRayImprovedAntialising(Nx, Ny, i, j);
    Pixel.pixelDone();
    Pixel.printPixel();
} else { //  //build picture with antialiasing improve
    castRayRegularAntialiasing(Nx, Ny, i, j);
    Pixel.pixelDone();
    Pixel.printPixel();
}
```

בתוך calacLocalEffects:
לפי המשתנה softShdows הוא יודע לשלוח לפונקציה המתאימה

```
Double3 ktr;
if (Camera.softShadows) {  //build picture with soft shadows improve
    ktr = transparencyWithSoftShadows(gp, lightSource, l);
} else {  //build picture without soft shadows improve
    ktr = transparency(gp, lightSource, l, n);
}
```

התמונה מזוויות נוספות

התמונה מהצד:

התמונה קרוב למראה: