דו"ח סיכום
למיני פרויקט
במבוא להנדסת תוכנה

מרצה מנחה – אליעזר גינסבורגר

מגישות:
ברכי טרלקלטויב, תז-325925626
חן אלקיים , תז-214243602

תוכן עניינים

תיאור הפרויקט

<u>רקע:</u>

הפרוייקט נכתב בשיטת Pair programing– מתכנת אחד כותב ומתכנת אחד בודק שאין טעויות.

פיתחנו כלי שנוכל בעזרתו לרנדר תמונה, ע"י הגדרת סצנה המכילה צורות גיאומטריות ומקורות אור.

הפרויקט נכתב ב- JAVA בInteliJ עם שימוש בJunit.

<u>מבנה</u>

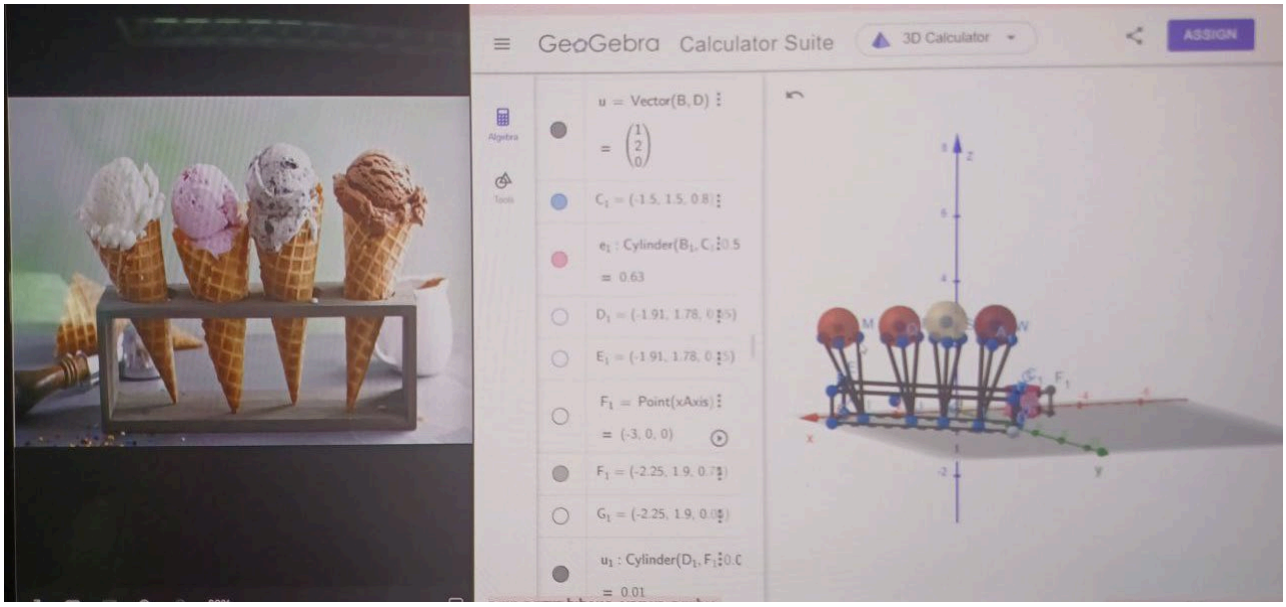
חילקנו את המחלקות לחבילות כך שכל מחלקה בחבילה שקשורה אליה.

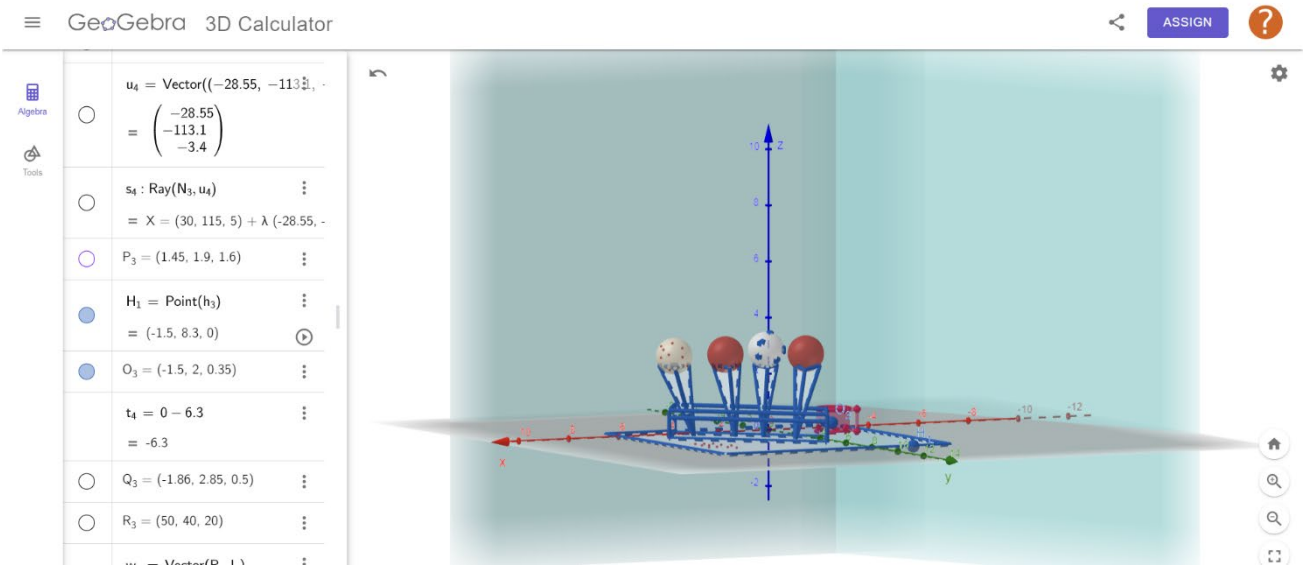
- החבילה Primitives מכילה את כל הצורות הפרימיטיביות, ביניהם נקודה, וקטור, צבע, סוג חומר.
- החבילה Geometries מכילה ממשק לצורות גאומטריות המחייב כל צורה גיאומטרית לממש פונקציות חיתוכים ופונקציות בסיסיות בזמן יצירת תמונה. היא מכילה גאומטריות כגון ספרה, מישור, משולש ועוד.
- החבילה Lighting מכילה ממשק המחייב כל מקור אור להחזיר את הכיוון והעוצמה שלו וכן מקורות אור כמו אור כיווני, אור ממוקד ואור סביבתי.
- החבילה Scene מכילה את המחלקה Scene שמגדירה Scene עם רשימת גאומטריות ומקורות אור שאפשר לרנדר.
- החבילה Renderer שאחראית על הרנדור של התמונה, מכילה את המחלקות RayTracerBase, Camera , פונקציות אלא אחראיות על שליחת קרניים וחישוב הצבע של הפיקסלים בתמונה.
בנוסף יש שם את המחלקה ImageWriter שכותבת אל התמונה את מה שיצרנו ויוצרת קובץ jpg.

- <u>שיטת עבודה</u>

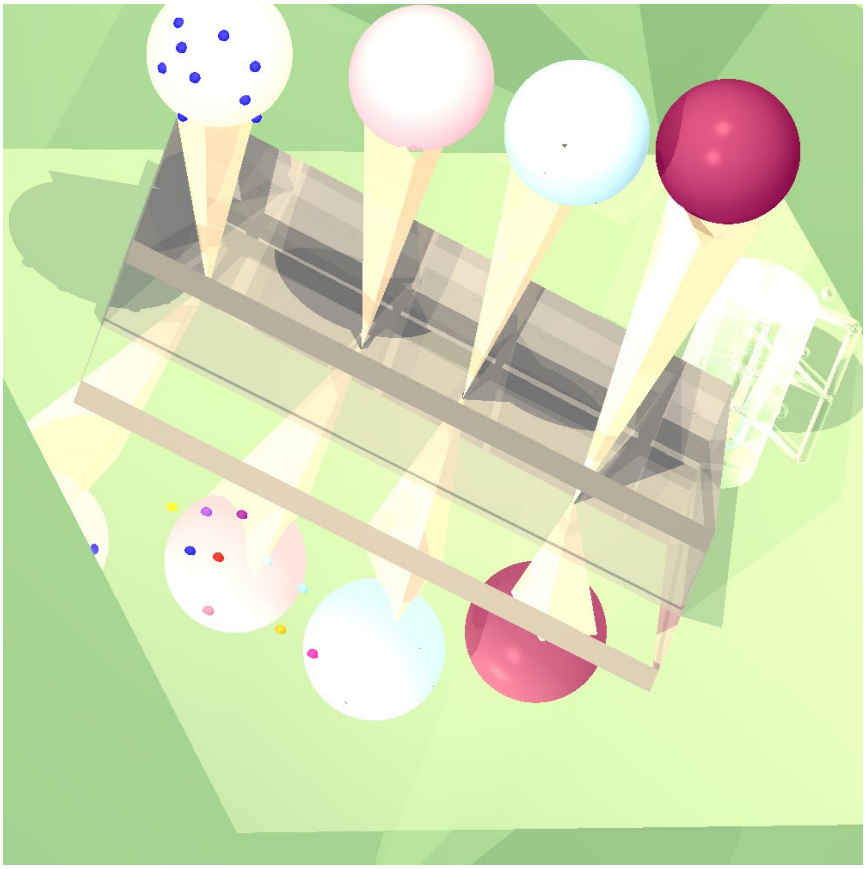
לאורך כל הפרוייקט הקפדנו על עקרון ה-TDD, עבור כל מתודה ומחלקה כתבנו טסט שבודק את תקינות המימוש שלהם ובנוסף מימשנו טסטים הבודקים את האינטגרציה בין המחלקות.

<u>שלבי יצירה ב- GeoGebra</u>





תוצאה בIntel:



MINIP 1

שיפור תמונה

Anti-Aliasing:

הבעיה- בשיטה הנוכחית, כל פיקסל מקבל את הצבע שיש במרכז שלו, ללא התחשבות במה שקורה בשאר הפיקסל. לכן בתמונה שנוצרת יש מעברים חדים בין צבע של פיקסל לצבע של הפיקסל לידו, ונקבל שצורות שאינן ישרות יראו מחוספסות והחלוקה לפיקסלים תבלוט ונקבל מראה פחות טוב.

הפתרון- יצירת קרניים לנקודות נוספות בפיקסל מלבד המרכז, וחישוב ממוצע הצבעים מכל הצבעים של הקרניים שהטלנו לאותו פיקסל.
ככה, בקצוות הצורה יתקבל צבע משולב והגבולות יהיו "רכים" יותר.

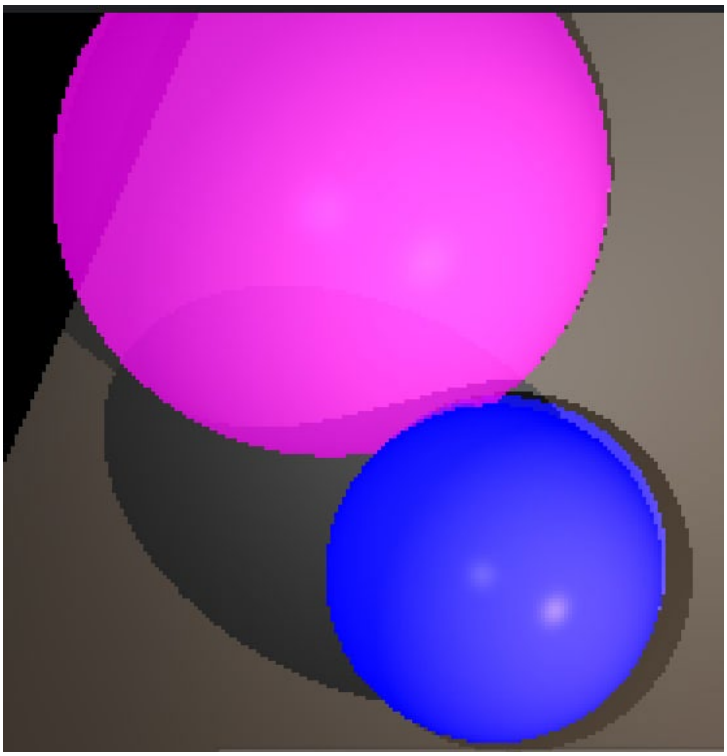
נשתמש ב- super sampling על פי האלגוריתם הרנדומלי – נשלח קרניים שמממוקמות באופן רנדומלי בתוך הפיקסל בנוסף לקרן המקורית.

השינויים שביצענו:

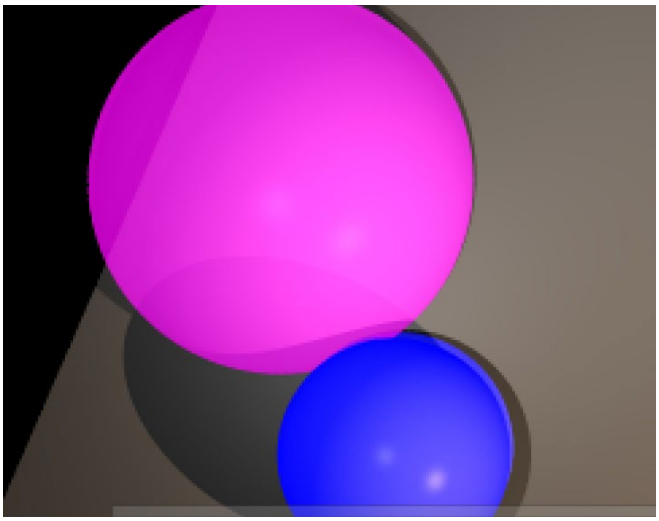
הוספנו במחלקת Camera בפונקציית randerImage (שאחראית להטיל קרן דרך מרכז כל פיקסל) שתקבל פרמטר numRays עבור מספר הקרניים שרוצים להטיל ולא קרן בודדת כמו שהיה עד עכשיו, המימוש ישאר אותו דבר רק שעבור כל פיקסל נטיל את מספר הקרניים שקיבלנו כפרמטר על ידי פונקציה חדשה ConstuctRays שיוצרת כמות של קרניים דרך כל פיקסל לטווח מטרה של מלבן מסביב למרכז הפיקסל.

הפונקציה הזאת תופעל על ידי הפונקציה castRay הקיימת כאשר מספר הקרניים שרוצים להטיל הוא יותר מ1, ותחזיר רשימה של כל הקרניים כך שנעבור על כל רשימת הקרניים ונחשב את הצבע של הפיקסל על ידי ממוצע של כל הנקודות שקיבלנו.

לפני השיפור



אחרי השיפור



MINIP

שיפור ביצועים

Multi- Threading:

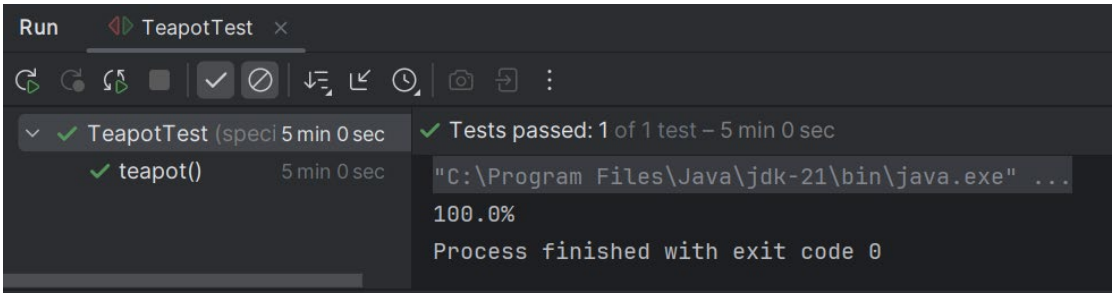
נריץ את התוכנית על מספר תהליכונים במקביל כדי לנצל זמן. ככה נוכל לצבוע יותר פיקסלים בו זמנית והתהליך של יצירת התמונה יהיה הרבה יותר מהיר!

השינויים שביצענו –

הוספנו בCamera הוספנו שדה PixelManager מטיפוס המחלקה PixelManager שקיבלנו מהמרצה.
השדה עוזר לנהל את מספר הפקיסלים שכל תהליך אחראי עליהם.
בפונקציה renderImage התבצעו השינויים הנל:

```java
public Camera renderImage(int numRays) {  24 usages  Brachi20
    int nX = imageWriter.getNx();
    int nY = imageWriter.getNy();
    // Verify that nX and nY are not zero to avoid division by zero
    if (nY == 0 || nX == 0)
        throw new IllegalArgumentException("It is impossible to divide by 0");
    // Initialize the pixel manager
    pixelManager = new PixelManager(nY, nX,  interval: 0.1);
    // Check if the number of threads is 0
    if (threadsCount == 0) {
        for (int i = 0; i < nY; ++i)
            for (int j = 0; j < nX; ++j)
                castRay(nX, nY, j, i, numRays);
    }
    else { // see further... option 2
        var threads = new LinkedList<Thread>(); // list of threads
        while (threadsCount-- > 0) // add appropriate number of threads
            threads.add(new Thread(() -> { // add a thread with its code
                PixelManager.Pixel pixel; // current pixel(row,col)
                // allocate pixel(row,col) in loop until there are no more pixels
                while ((pixel = pixelManager.nextPixel()) != null)
                    // cast ray through pixel (and color it - inside castRay)
                    castRay(nX, nY, pixel.col(), pixel.row(),numRays);
            }));
        // start all the threads
        for (var thread : threads) thread.start();
        // wait until all the threads have finished
        try {
            for (var thread : threads) thread.join();
        } catch (InterruptedException ignore) {
        }
    }
    return this;
}
```

זמן הריצה אחרי שיפור זמן adaptive ואחרי שיפור זמן על ידי threads



Adaptive Super Sampling:

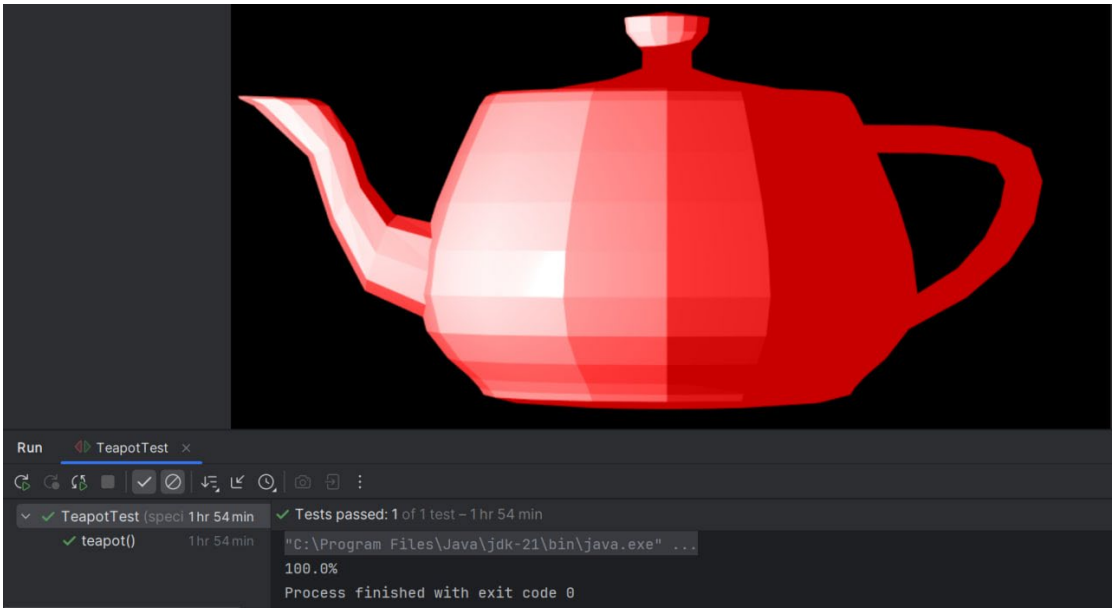
כדי לשפר את התמונה הוספנו את השיפור anti-aliasing שעבור כל פיקסל, במקום לירות קרן אחת יורה כמה קרניים ואז מחשב את הצבע הממוצע של כל הנקודות שנפגעו.
זה משפר את התמונה כאשר יש מעבר צבע בתוך הפיקסל ואז המעבר הזה יהיה פחות חד לעין.
כמובן שבמקרה שכל הפיקסל שאותו אנו רוצים לצבוע הוא באותו צבע, אז זה מיותר לירות עשרות קרניים ולחשב את הצבע של כל אחד.
הפתרון של adaptive בא לטפל במקרה הזה. לפיו , לפני שנחשב את הצבע של כל הנקודות דרך הטלת כל קרן, נבדוק קודם עבור מספר מצומצם של נקודות שנבחר , ואם נקבל שם את אותו צבע אז אין סיבה להטיל עוד קרניים ולבדוק את השאר.
בחרנו לבדוק את הנקודות במרכז ובפינות, שהרי אם הם באותו צבע אז סביר להניח שכל הפיקסל באותו צבע ואז פשוט נצבע בצבע הזה.
אחרת, נבצע את החישוב רגיל לפי השיטה שתארנו בשיפור anti- aliasing.

השינויים שביצענו-

הוספנו ב Camera שדה בוליאני חדש בשם adaptive עם פונקציית set שנוכל לסמן אם רוצים להפעיל את השיפור adaptive או לא ב cameraBilder, ושינינו את הפונקציה castRay ככה שאם רוצים שיפור האצה אז נשלח בכל פעם 5 קרניים דרך כל פיקסל ונבדוק אם בכולם חזר אותו צבע אז זה יהיה הצבע של הפיקסל אבל אם חזר צבעים שונים אז צבע הפיקסל יהיה הצבע הממוצע שיחושב על ידי הפונקציה החדשה averageColor .

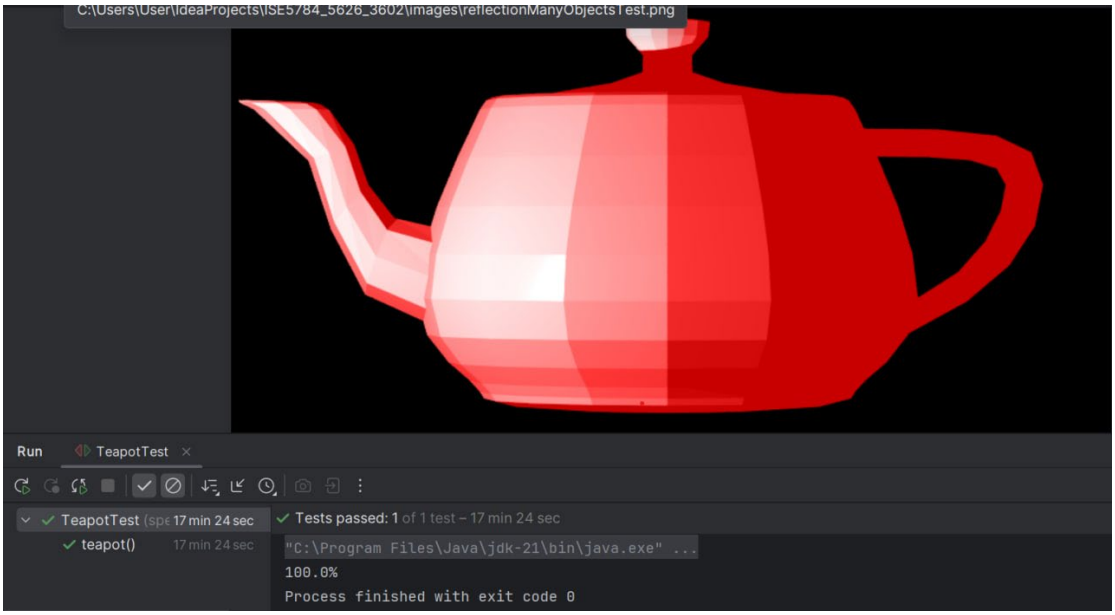
זמן הריצה לפני השיפור :

שעה ו54 דקות כאשר מטילים 100 קרניים דרך כל פיקסל



זמן הריצה אחרי השיפור:

17 דקות ו24 שניות כאשר מטילים 100 קרניים דרך כל פיקסל



בונוסים

- מימוש get Normal עבור גליל סופי
- מימוש חיתוכי קרן עם Polygon
- מימוש חיתוכי קרן עם Cylinder
- טרנספורמציית הזזה וסיבוב
- תאורת ספוט עם אלומה צרה יותר
- בניית תמונה עם המון גופים
- פתירת בעיית המרחק עם הצללה בדרך השנייה (מימוש FindIntersection עם פרמטר maxDistance)
- הרצה בעזרת threading

ביבליוגרפיה

- חומרי עזר מהמודל-
  מצגות הקורס התאורטי, מצגות המעבדה.

- הסבר על המצגות של הקורס ע"י הסטודנטית לאה חיים
  .055-971-8363

- תודה רבה לאליעזר על העזרה המרובה.