בס"ד

דוח סיכום MP01, MP02

מגישות: הילה מילר והילה רוזנטל

התמונה שיצרנו בפרויקט:

- תמונה זו מכילה כ 80 גופים שונים.
- ניתן לראות בה את אפקט ההשתקפות ברצפה, ובקיר התכלת.
- אפקט השקיפות מתבטא בכוס ובמנורה.
- התמונה מכילה שני מקורות אור, אחד נמצא במנורה השקופה ואחד מחוץ לחלון לדמות אור שמש מה שיצר אפקט מרהיב של צל בחדר.

בס"ד

דוח סיכום – מיני פרויקט 1: הצללה רכה (Soft Shadows)

🟡 מטרת השיפור

מטרת שיפור זה הייתה להוסיף תמיכה בצללים רכים (Soft Shadows), כדי לדמות בצורה ריאליסטית יותר את תופעת ההצללה שמתרחשת כאשר מקור אור רחב מואר חלקית בלבד על ידי גוף חוסם.

🛠️ אופן המימוש

1. יצירת קרני צל מרובות

במקום קרן צל אחת לכל נקודת חיתוך, יצרנו מספר קרני צל מנקודת הפגיעה אל דגימות שונות על פני שטח מקור האור (Area Light).
כל קרן נבדקת האם היא חסומה על ידי עצם אחר – ואם כן, האם העצם שקוף באופן חלקי או מלא.

2. שימוש במתודת `transparency`

הוספנו העמסה למתודה `transparency` כך שתקבל משתנה נוסף שמייצג את מספר הקרניים, אשר מחזירה ערך מסוג `Double3` המייצג את כמות האור שעוברת דרך העצמים החוסמים.
ככל שיותר קרני צל חסומות על ידי עצמים אטומים – האזור כהה יותר. אם הקרניים עוברות דרך עצמים שקופים – הצל רך וחלקי.

```
**/

 Calculates the overall transparency (ktr) from a point to an area light source *

 .(by sampling multiple rays across the light's surface (for soft shadow effect *

*

 param intersection the intersection point being shaded@ *

 param numberOfSamples the number of soft shadow samples to use@ *

 return averaged transparency factor from the samples@ *

/*

    } (private Double3 transparency(Intersection intersection, int numberOfSamples

If the light is effectively a point light (no radius or no position), or not //
                                                    ,enough samples requested

                fall back to the standard hard-shadow transparency computation //

                                    || if (intersection.light.getRadius() == 0

                            || intersection.light.getPosition() == null

                                                } (numberOfSamples <= 1
```

```
                                        ;(return transparency(intersection
                                                                        {

                    Direction from the intersection point toward the light //
                                        ;Vector l = intersection.lightDirection

      Create an orthogonal basis (vUp and vRight) around the light direction //
                                            ;()Vector vUp = l.createOrthogonal
                                           ;(Vector vRight = vUp.crossProduct(l

                       Create a virtual "board" around the light's position //
                  (to serve as the sampling area (radius * 2 = full diameter //
                                                        )Board board = new Board
                                        ,()intersection.light.getPosition
                                                            ,vUp, vRight
                                      intersection.light.getRadius() * 2
                  setCircle(true); // Limit sampling area to a circular shape.(

                 Generate sampling points over the circular area of the light //
List<Point> lightSamples = board.getPoints(numberOfSamples); // 8x8 samples = 64
                                                                (rays (customizable

            Double3 ktrSum = Double3.ZERO; // Accumulator for total transparency
      int contributingSamples = 0;   // Counter for how many samples contributed

                    Loop through all sampled points on the light's surface //
                                    } (for (Point areaLightPoint : lightSamples
  Create a direction vector from the hit point to the sample point on the //
                                                                          light
                                                        Vector areaLightDir =
                          ;()areaLightPoint.subtract(intersection.point).normalize
```

```
            Skip samples where light is coming from behind the surface //

                                                (if (intersection.nl <= 0

                                                            ;continue


                Measure the distance to the sampled point on the light //

          ;(double lightDist = intersection.point.distance(areaLightPoint


                 Construct a shadow ray toward the sampled light point //

                Ray shadowRay = new Ray(intersection.point, areaLightDir,
                                                            ;(intersection.normal


  (Get all intersections along the shadow ray (up to the light distance //

                                            List<Intersection> shadowHits =
                  ;(scene.geometries.calculateIntersections(shadowRay, lightDist


                            (Start with full transparency (no blockage //

                                                ;Double3 ktr = Double3.ONE


For each object intersected along the way, multiply by its transparency //

                                                } (if (shadowHits != null

                                } (for (Intersection hit : shadowHits


      ktr = ktr.product(hit.material.kT); // Multiply by object's
                                                                    transparency

                 Stop early if transparency becomes negligible //

                            } ((if (ktr.lowerThan(MIN_CALC_COLOR_K

                                            ;ktr = Double3.ZERO

                                                        ;break

                                                                {

                                                                    {
```

```
{
// Accumulate the transparency from this sample
ktrSum = ktrSum.add(ktr);
contributingSamples++;
}

// If no samples contributed (all were back-facing or blocked), return full blockage
if (contributingSamples == 0)
return Double3.ZERO;
// Return the average transparency based on number of samples attempted
return ktrSum.reduce(lightSamples.size());
}
```

3. דגימת מקור האור

הוספנו שדה חדש לכל מקור אור: `radius`, שמייצג את רדיוס מקור האור.
במקרה של Spot ו-Point light, אנו מדגמים את מקור האור כעיגול קטן ומגרילים ממנו נקודות רבות.
הקרניים נורות מנקודת הפגיעה לכיוון כל אחת מהנקודות שנדגמו.

4. ביצועים

- שיפרנו את מחלקת `Ray` כך שתשתמש בקבוע `DELTA` כדי להזיז את ראש הקרן ולמנוע חיתוך עצמי (Self-shadowing).
- בחרנו מספר דגימות סביר (למשל 15 קרניים) כדי לאזן בין איכות הצללים לזמן הריצה.
- השיפור נבדק גם עם תהליכונים (Multithreading) כדי לקצר את זמן הרינדור של התמונות.

5. בדיקות

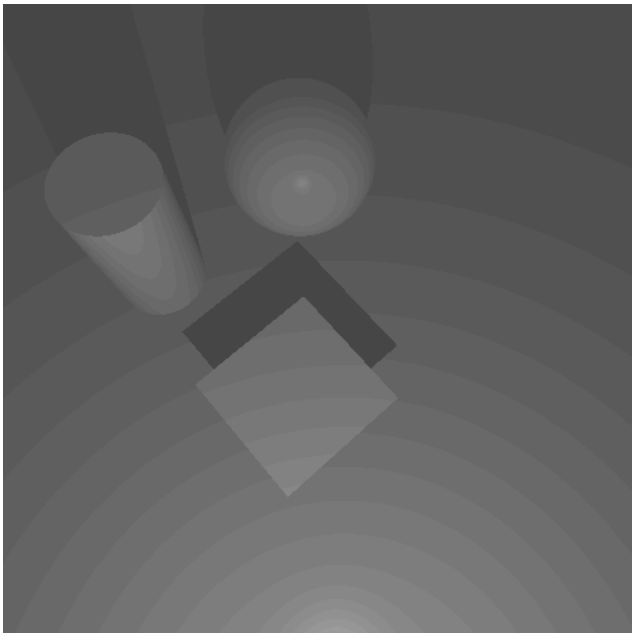
- בנינו תמונה ייעודית הכוללת גוף שמטיל צל רך (כגון כדור או גליל) ומקור אור עם רדיוס.
- וידאנו שהתוצאה מייצרת אזור צל הדרגתי ולא חד, בהתאם לציפיות.

בס"ד

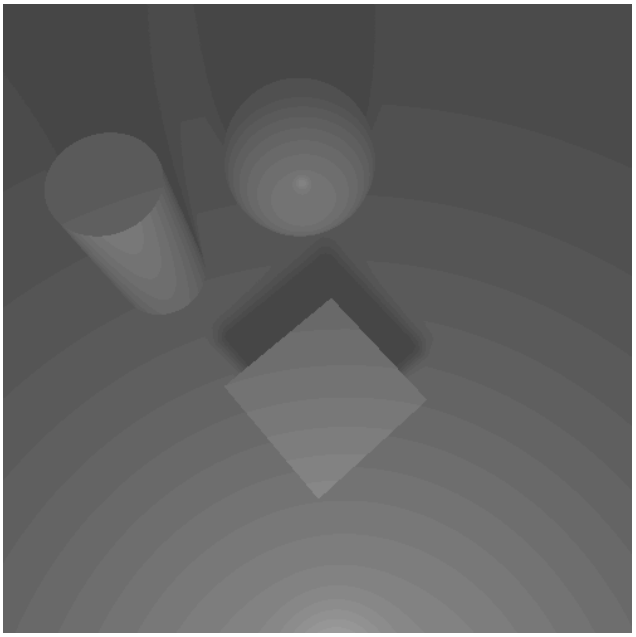
✅ תוצאה סופית

- הצללים בתמונה כעת רכים, מדמים בצורה טבעית יותר את מעבר האור דרך עצמים חלקיים ואת הטשטוש של גבול הצל.
- זמן הריצה התארך אך בצורה צפויה.
- התמונות מדגימות הבדל ברור בין מקור אור "קשה" (ללא רדיוס) למקור אור "רך" (עם רדיוס ודגימה).

לפני



אחרי



🧠 עקרונות פיתוח שיושמו

- **DRY** – הוספנו בנאי חדש למחלקת `Ray` למניעת שכפול קוד של הזזת נקודת הקרן.
- **SRP** – מתודת `transparency` אחראית אך ורק על חישוב הצללה חלקית.
- **Performance awareness** – איזון בין איכות הצל לרזולוציית התמונה ומהירות הריצה.

🏁 סיכום

באמצעות שיפור זה הצלחנו ליישם צללים רכים בפרויקט שלנו בצורה מדויקת, גמישה וריאליסטית, תוך הקפדה על עקרונות תכנות תקינים ויכולת שדרוג עתידית למספר מקורות אור ומערכות מורכבות יותר.

בס"ד

דוח סיכום – מיני פרויקט 2: האצת BVH

🎯 מטרת הפרויקט

ליישם מנגנון האצת חישובים גיאומטריים באמצעות **BVH (Bounding Volume Hierarchy)**,
ובכך לקצר משמעותית את זמני הרינדור של תמונות מורכבות בסצנה תלת־ממדית.

🧠 עקרון הפעולה

BVH משתמש במבנה היררכי של **תיבות תחימה (Bounding Boxes)** אשר עוטפות קבוצות של גיאומטריות.
במקום לבדוק חיתוך של קרן מול כל גוף בנפרד, הקרן נבדקת תחילה מול תיבות תחימה, וכך ניתן **לדלג על בדיקות רבות** לגופים שממילא לא רלוונטיים.

🛠️ מימוש

שינויים שבוצעו:

- מימוש מלא של `BoundingBox` עם שדות `min` ו־`max` מסוג `Point`.
- עדכון מחלקת `Geometries` לתמיכה ב־BVH אוטומטי ואופציונלי:
  - `buildBvhTree()` – חלוקה מבוססת תיבות
  - `buildBinaryBvhTree()` – חלוקה מבוססת מרחקים בין גופים
- הטמעה בשיטת `calculateIntersections` כך שקרן שלא חוצה Bounding Box כלל לא תמשיך לבדוק גיאומטריות שבתוכה.

📷 בדיקות שבוצעו

המערכת נבדקה תחת **שתי תצורות עיקריות**:

1. **ללא צללים רכים** – כלומר, קרן אחת לצל.

2. **עם צללים רכים** – שימוש באלומת קרניים (למשל 64 קרניים לנקודה).

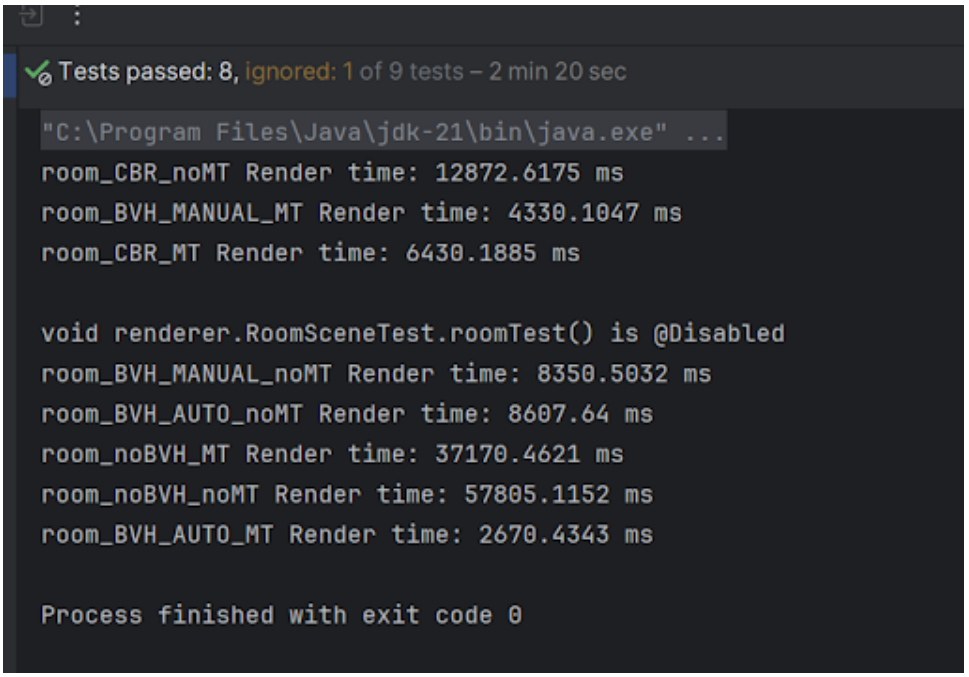
לכל תצורה נבדקו שילובים של:

- עם/בלי BVH
- עם/בלי Multi-Threading

📊 תוצאות (ללא צללים רכים)

תצורה	זמן רינדור
noBVH, noMT	57,805 ms
noBVH, MT	37,170 ms
BVH Manual, noMT	8,350 ms
BVH Manual, MT	4,330 ms
BVH Auto, noMT	8,607 ms
BVH Auto, MT	**2,670 ms**

שיפור של פי **21.6~** לעומת ללא BVH וללא Multi-Threading.



📊 תוצאות (עם צללים רכים)

תצורה	זמן רינדור
noBVH, noMT	2,055,717 ms (~34 דקות)
noBVH, MT	1,341,216 ms
BVH Manual, noMT	268,006 ms
BVH Manual, MT	164,177 ms
BVH Auto, noMT	289,320 ms
BVH Auto, MT	**106,268 ms** (~1.8 דקות)

שיפור של **פי ~19.3** לעומת הריצה ללא האצה.

```
Tests passed: 8, ignored: 1 of 9 tests – 1 hr 28 min
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
room_CBR_noMT Render time: 764358.014 ms
room_BVH_MANUAL_MT Render time: 164177.8889 ms
room_CBR_MT Render time: 305627.1842 ms

void renderer.RoomSceneTest.roomTest() is @Disabled
room_BVH_MANUAL_noMT Render time: 268006.8986 ms
room_BVH_AUTO_noMT Render time: 289320.9919 ms
room_noBVH_MT Render time: 1341216.1222 ms
room_noBVH_noMT Render time: 2055717.5913 ms
room_BVH_AUTO_MT Render time: 106268.0169 ms

Process finished with exit code 0
```

🧩 מסקנות

- מנגנון ה־BVH **הפחית משמעותית את זמני הרינדור** – לעיתים פי 15–20.
- גם במקרים כבדים כמו **צללים רכים** (שדורשים 64 קרניים לנקודה), ההאצה השפיעה מאוד.
- שימוש ב־`Point` ל־min/max בתוך `BoundingBox` שיפר את הקריאות ואולי אף השפיע לטובה על הביצועים.
- שילוב של **BVH + Multi-threading** הביא לזמני הרצה מיטביים.

✅ סיכום

הפרויקט השיג את מטרתו: מערכת BVH אפקטיבית, מותאמת היטב לקוד הקיים, שהביאה לשיפור מדיד ומהותי בביצועים.
הודות למימוש נכון של תיבות תחימה, היררכיה יעילה והפחתת בדיקות מיותרות – חווית הרינדור שודרגה משמעותית.