

דוח מיני פרויקט

מטרת הפרויקט: ייצור שיפורים בתמונות שנוצרות במנוע רינדור תלת-ממדי, באמצעות טכניקות של **Soft Shadows**, ובהמשך האצת הביצועים בעזרת תהליכיים מרובים (Multithreading) וטכניקת **Boundary Volume Hierarchy**.

חלק 1:

שיפור איכות בתמונה בעזרת **Soft Shadows**

הגדרת הבעיה:

במעקב קרנויים סטנדרטי (Ray Tracing), חישוב הצללים מתבצע באמצעות קרן צל אחת לכל מקור אור. גישה זו יוצרת צללים חדים ובלתי-מציאותיים, שאינם משקפים את ההתנהגות הפיזיקלית של אור המוקדם משטח אוור ולא מנוקודה בודדת.

החדשון שלנו:

ישמןנו הצללות רכות (Soft Shadows) באמצעות קרני צל מבוזרות. עבור כל מקור אוור, במקומם לבדוק כיון אחד בלבד, אנו יוצרים מספר קרני צל מהנקודה הפוגעת לערך מיקומים אקראיים על פני שטח האור. פעולה זו מדמה הסתרה חלקית של האור, מה שMBOLIL לשוללים רכים יותר של הצללים ולתחושת תאורה מציאותית יותר. המערכת שוקלת את פגיעת הקרנויים – תוך הבחנה בין קרנויים שמושתרות לאלו שאין מושתרות – כדי לקבוע את צבע הפיקסל הסופי.

חישוב עצמת הצללים הרכבים:

להבדיל מהשיטה הקלסית, בה נשלחת קרן צל בודדת לכל מקור אוור, אנו מבצעים דגימה מרובת קרני צל באזורי שטח האור.

בפונקציה calcLocalEffectsSoftShadows אנו מייצרים רשת של קרני צל היוצאות מנוקודת המפגש אל נקודות שונות על פני שטח האור (לפי רדיוס האור והכוון). לכל קרן אנו בודקים האם היא נחסמת על ידי גיאומטריה כלשהי באמצעות הפונקציה isBlocked.

כדי ליעיל את תהליך הבדיקה ולצמצם את זמן החישוב, אנו משתמשים במבנה האצה מסוג **BVH (Bounding Volume Hierarchy)** – עץ היררכי של קופסאות תחומות, שמאפשר סינון מוקדם של גיאומטריות לא רלוונטיות. קר ניתן לבדוק חיתוך רק מול עצמים פוטנציאליים במקום לסרוק את כל הסצנה עבור כל קרן.

לאחר אישור תוצאות הקרנויים, אנו מחשבים את מקדם השקיפות הממוצע (ערך בין 0 ל-1), אשר מתאר את מידת העמידות של הצללים. ערך זה משמש כמשקל לקביעת עצמת התאורה הסופית, ומאפשר ייצור הצללות רכות עם שוללים מוטשטשים יותר – הקרובים להתנהגות האור הפיזיקלית ומעשירים את תחושת העומק והמציאות.

```

/**
 * Determines whether a ray is blocked before reaching the light source.
 *
 * @param shadowRay the ray toward the light
 * @param lightDistance distance to light source
 * @return true if the ray is blocked
 */
private boolean isBlocked(Ray shadowRay, double lightDistance) { 1 usage ↳ efratYi +1
    List<Intersection> intersections = scene.geometries.calculateIntersections(shadowRay);
    if (intersections == null) return false;

    for (Intersection i : intersections) {
        if (shadowRay.getHead().distance(i.point) < lightDistance - DELTA &&
            i.geometry.getMaterial().KT.lowerThan(MIN_CALC_COLOR_K)) {
            return true;
        }
    }
    return false;
}

```

```

/**
 * Calculates soft shadows using a sampling strategy.
 *
 * @param lightSource area light source
 * @param intersection the intersection
 * @return shadow intensity modifier
 */
private Double3 calcLocalEffectsSoftShadows(PointLight lightSource, Intersection intersection) { 1 usage ↳ efratYi +1
    Vector l = lightSource.getL(intersection.point);
    Vector vUp;

    try {...} catch (Exception e) {...}

    int numSamples = 5;
    TargetArea area = new TargetArea(numSamples, size: lightSource.getRadius() * 2, l.scale(t: -1), vUp, lightSource.getPosition());
    double totalShadow = 0;
    int validRays = 0;

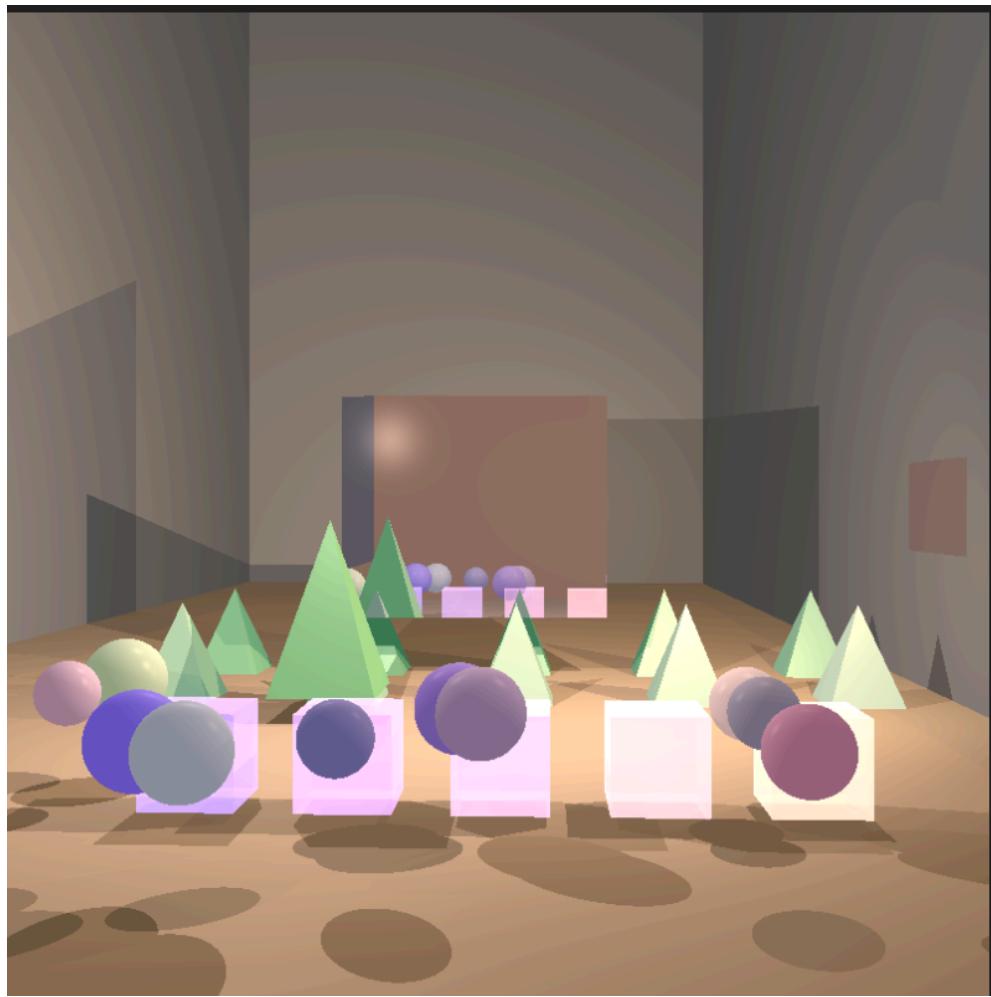
    for (int i = 0; i < numSamples; i++) {...}

    return validRays > 0 ? new Double3(value: 1 - (totalShadow / validRays)) : transparency(intersection);
}

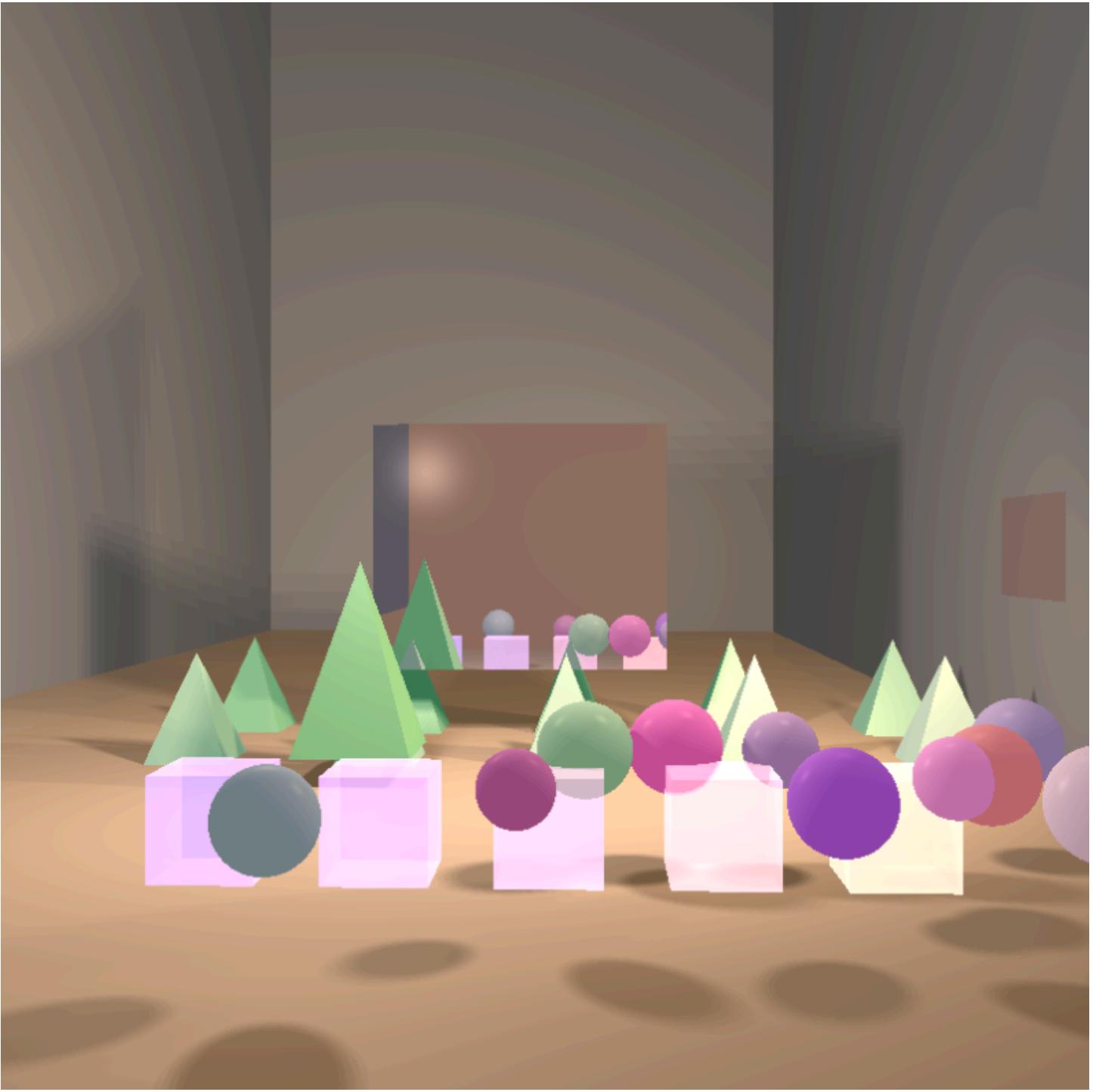
```

השווות התמונות:

תמונה לפניו שיפור:



תמונה אחרי השיפוץ:



התוצאה:

הבדל ויזואלי:

צללים רכים אפשרים קצויות צללים מטושטשים וטבעים יותר במקומות קווים חדים ונוקשים. ההטמעה באמצעות דגימת מספר קרנויים מכל מקור או רדיאוס האור על פני שטח, וכך מקבלת הצללה הדרגתית והדרגתית יותר.

השלכות ביצועים:

דגימה מרובת קרנויים לכל מקור או מגדילה את עלות החישוב, אך התוצאה הויזואלית המשופרת מצדיקה את ההוצאה החישובית.

מודולריות וונישות:

הפעלת הצללים הרכים מתבצעת דרך פרמטר רדיוס האור – רדיוס אפס לא מפעיל את השיפור (הצללים קשים),

וככל שהרדיווס גדול יותר הצללים רכים יותר, מפעיל דוגמה לייצור צללים רכים. ניתן לשלב ולהתאים בקבוקות בפרמטרים השונים.

חלק 2:

ריבוי תהיליכונים-Multithreading

הגדרת הבעיה – זמן רינדור ארוך

לאחר שיישמנו שיטות לשיפור איקות התמונה באמצעות Soft Shadows, זהינו כי זמן הרינדור לכל תמונה עולה במידה משמעותית.

לדוגמה:

רינדור עם 16 קרניים לכל פיקסל (4x4) עשוי להאריך את זמן החישוב פי עשרה ואף יותר, בהשוואה לרינדור בסיסי.

בסצנות מורכבות הכוללות שקיפות, החזרים ותאורות מרובות, כל קרן נוספת מגדילה משמעותית את העומס החישובי.

למעשה, איקות התמונה באה על חשבון ביצועים וזמן ריצה.

הפתרון – מעבר לרינדור מקבילי (Parallel Rendering)

כדי להתגבר על הבעיה, הטמענו מנגןון רינדור במקביל באמצעות תהיליכונים (Threads).
ישמןנו שלוש גישות עיקריות:

- `renderImageNoThreads()` – רינדור סידרתי, פיקסל אחר פיקסל (ברירת מחדל).
- `renderImageRawThreads()` – ייצור תהיליכונים ידנית, כאשר ניתן להגדיר את מספר התהיליכונים.
- `renderImageStream()` – שימוש ב-Streams Parallel Java לビיצו רינדור במקביל באופן אוטומטי.

שליטה במערכת

הפרמטר `threadsCount` מגדיר את אופן הרינדורו:

- 0 → רינדור סידרתי רגיל
- 1 → שימוש ב-Streams Parallel
- כל ערך חיובי → ייצור מספר תהיליכונים ידני לפי המספר שהוגדר

ארQUITטורת המערכת

PixelManager – מחלקת עוז לניהול חלוקת העבודה בין תהיליכונים:

- מנהלת את חלוקת הפיקסלים לעיבוד בצורה בטוחה
- מודדת שתהיליכונים שונים לא יעבדו על אותו פיקסל בו זמןית
- מסנכרנת בין תהיליכונים לשמרות יציבות ודיקוק בתהיליך

מנגןון `renderImageRawThreads()`

- יוצרת רשימת תהליכיים (Threads)
- כל Thread מקבל תור לעיבוד פיקסלים דרך PixelManager ופועל באופן עצמאי
- געשה שימוש ב-thread.join() על מנת להמתין לסיום כל התהליכיים לפני המשך הרינדור

```
/***
 * Render image using multi-threading by creating and running raw threads* @return the camera object itself
 */
private Camera renderImageRawThreads() { 1 usage  ruchamabruker
    var threads = new LinkedList<Thread>();
    while (threadsCount-- > 0)
        threads.add(new Thread(() -> {
            PixelManager.Pixel pixel;
            while ((pixel = pixelManager.nextPixel()) != null)
                castRay(pixel.col(), pixel.row());
        }));
    for (var thread : threads) thread.start();
    try {
        for (var thread : threads) thread.join();
    } catch (InterruptedException ignore) {}
    return this;
}
```

:renderImageStream

ממש רינדור מקבילי על ידי שימוש ב-`IntStream.range(...).parallel()` שמאפשר לעבוד במקביל את כל שורות ועמודות הפיקסלים.
היתרון בשיטה זו הוא פשוטות וקומפקטיות הקוד.
החיסרון הוא שליטה מוגבלת על אופן תזמון התהליכיים והיכולת לעקוב אחריהם באופן מדויק.

```
/***
 * Render image using multi-threading by creating and running raw threads* @return the camera object itself
 */
public Camera renderImageStream() { 1 usage  ruchamabruker
    IntStream.range(0, nY).parallel() //
        .forEach( int i -> IntStream.range(0, nX).parallel() //
            .forEach( int j -> castRay(j, i)));
    return this;
}
```

שדרוגים ב-Builder – ניהול ביצועים חכם

כדי לאפשר למשתמש לשנות באופן גמיש בשיטת הרינדור, הוספנו ב-Builder של המצלמה אפשרות להפעיל רינדור במקביל באמצעות מספר תהליכיים שנקבע על ידי המשתמש.
פונקציה: `setMultithreading(int threads)`

```
public Builder setMultithreading(int threads) { 5 usages  ruchamabruker
    if (threads < -2) throw new IllegalArgumentException("Multithreading must be -2 or higher");
    if (threads >= -1) camera.threadsCount = threads;
    else { // == -2
        int cores = Runtime.getRuntime().availableProcessors() - SPARE_THREADS;
        camera.threadsCount = cores <= 2 ? 1 : cores;
    }
    return this;
}
```

פונקציה: (setDebugPrint(double interval)

מאפשרת להציג עדכוני התקדמות בזמן הרינדור ישירות בקונסולה. לדוגמה, ערך של 0.5 יארום להדפסת אחווי ההתקדמות כל חצי שניה. פונקציה זו שימושית במיוחד בעת דיבוג סצנות מורכבות, כדי לוודא שהרינדור אכן מתקדם בצורה תקינה.

```
/**  
 * Sets the interval for printing progress percentage.  
 *  
 * @param interval the interval in seconds  
 * @return this Builder instance  
 * @throws IllegalArgumentException if the interval is negative  
 */  
public Builder setDebugPrint(double interval) { 4 usages  ↳ ruchamabrunner  
    if (interval < 0) throw new IllegalArgumentException("Interval value must be non-negative");  
    camera.printInterval = interval;  
    return this;  
}
```

שיטת האצה BVH לאופטימיצית בדיקות חיתוך וזמן ריצה

לצורך ייעול בדיקות החיתוך בין קרניזים לאובייקטים בסצנה, מימשו מגנון Bounding Volume Hierarchy. מדובר בעץ היררכי שבו בכל צומת קיימת תיבה תחומה (bounding box) המקיפה תת-קובוצה של גיאומטריות. בעזרת סינון מוקדם של קבוצות לא רלוונטיות, ניתן לצמצם משמעותית את מספר בדיקות החיתוך בפועל.

שלבי השיפור

שלב ראשון – BVH יידי:

בשלב זה כל Shape מחשב לעצמו תיבה תחומה על ידי קרייה ל-'setBoundingBox'. זה מאפשר לבצע בדיקת חיתוך מהירה ראשונית – ואם הקרן לא חותכת את התיבה, נחסכת בדיקת החיתוך המדויקת והירה באיזומטריה עצמה.

שלב שני – BVH יידי:

בטיטים אידנו ידנית קבוצות של גיאומטריות לאובייקטי Geometries נוספים. לאחר מכן קראנו ל-'enableCBR'. כך נבנות תיבות תחומות גדולות יותר שעוטפות קבוצות של צורות, ומאפשרות סינון גם נוסף לפני מעבר לבדיקה פרטנית על כל Shape.

שלב שלישי – BVH אוטומטי:

בשלב זה קוראים ל-'BVHenable'. זה מפעיל בניית רקורסיבית אוטומטית של עצם BVH על כל הסצנה. בתהליך זה מתבצע:

- קביעת תיבה תחומה כוללת: הפונקציה `setBoundingBox` מחשבת את התיבה המקיפה את כל הצורות בצומת הנוכחי.
- תנאי עצירה לרקורסיה: אם קיימות 2 צורות או פחות – מפסיקים את הפיצול ומשאירים את הצומת עלה.
- בחירת ציר הפיצול: נבחר הציר הארוך ביותר מבין Z, Y, X – כדי להשיג חלוקה מאוזנת יותר של הצורות.

- מיען הצורות לפי הציר: ממיינים את הצורות לפי המיקום שלהם לאורך הציר.
- פיצול לרשימות משנה: מחלקים לשתי קבוצות (שמאל וימין) ומכניםים כל קבוצה לאובייקט جديد.
- רקורסיה: קוראים ל-`buildBVH` לכל אחד מהצדדים.
- החלת הצורות בזומת האב: במקום רישימת הצורות המקורית נשמרים שני תתי-עצים – וכך מתקבל מבנה היררכי שלם.

תרונות:

- חיסכון משמעותי בזמן רינדור, במיוחד בסצנות מורכבות הכוללות מאות או אלפי גיאומטריות.
- הפחתה דרמטית במספר בדיקות החיתוך לכל קרן – קרניות נבדקות רק מול תיבות רלוונטיות.
- מפחית במיוחד בשילוב עם השיפור של ריבוי תהליכיון.

```
/**
 * Build BVH tree with sorting by longest axis of bounding box.
 * Splits shapes into two halves recursively.
 */
public void buildBVH() { 3 usages  ↳ efratYi
    setBoundingBox();
    if (shapes.size() <= 2) return;

    double dx = boundingBox.maxX - boundingBox minX;
    double dy = boundingBox maxY - boundingBox minY;
    double dz = boundingBox maxZ - boundingBox minZ;

    int axis = 0; // 0=X, 1=Y, 2=Z
    if (dy > dx && dy > dz) axis = 1;
    else if (dz > dx && dz > dy) axis = 2;
```

```
final int sortAxis = axis;
shapes.sort((Intersectable a, Intersectable b) -> {
    double aCoord = 0, bCoord = 0;
    switch (sortAxis) {
        case 0 -> {
            aCoord = a.getBoundingBox().minX;
            bCoord = b.getBoundingBox().minX;
        }
        case 1 -> {
            aCoord = a.getBoundingBox().minY;
            bCoord = b.getBoundingBox().minY;
        }
        case 2 -> {
            aCoord = a.getBoundingBox().minZ;
            bCoord = b.getBoundingBox().minZ;
        }
    }
    return Double.compare(aCoord, bCoord);
});
```

```
int mid = shapes.size() / 2;
List<Intersectable> leftList = shapes.subList(0, mid);
List<Intersectable> rightList = shapes.subList(mid, shapes.size());

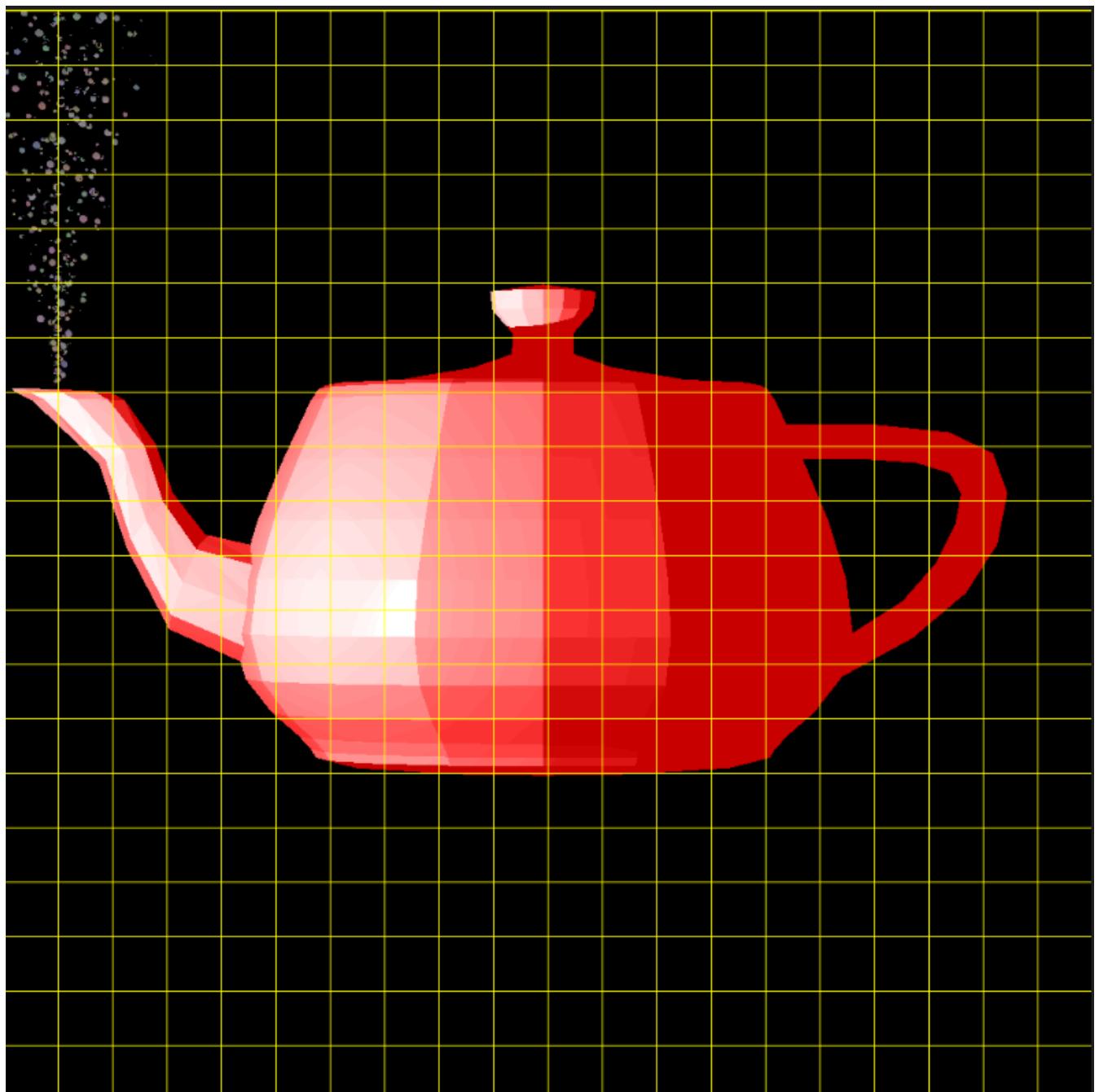
Geometries left = new Geometries();
Geometries right = new Geometries();
left.add(leftList.toArray(new Intersectable[0]));
right.add(rightList.toArray(new Intersectable[0]));

left.buildBVH();
right.buildBVH();

shapes.clear();
shapes.add(left);
shapes.add(right);

}
```

תמונה מרובות אובייקטים, הבדלים בזמן ריצה:



ללא האצות כלל:

Run TeapotTest.testTeapot1 ×

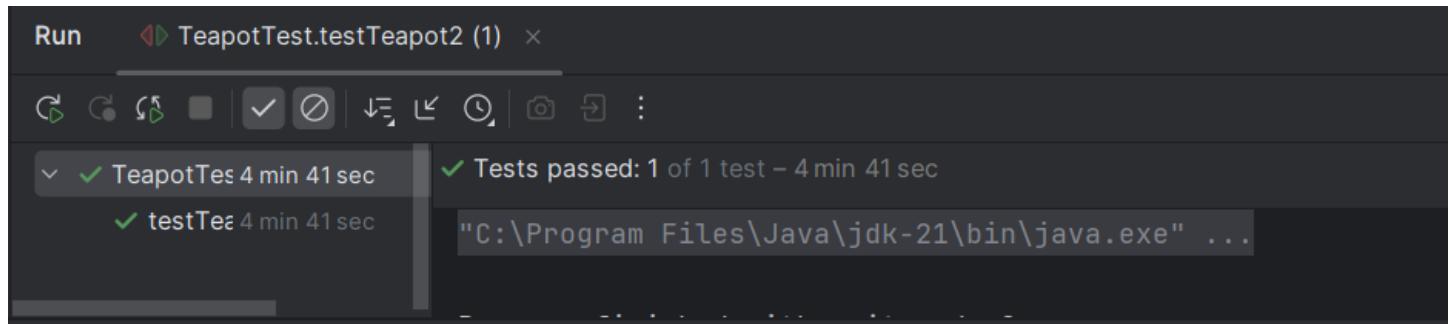
↻ ⏴ ⏵ ⏵ ⏵ ⏵ ⏵ | ✓ ⏵ | ⏵ ⏵ ⏵ ⏵ | ⏵ ⏵ ⏵ ⏵ | ⏵ ⏵ ⏵ ⏵ : ⏵

✓ TeapotTest (8 min 13 sec) ✓ Tests passed: 1 of 1 test – 8 min 13 sec

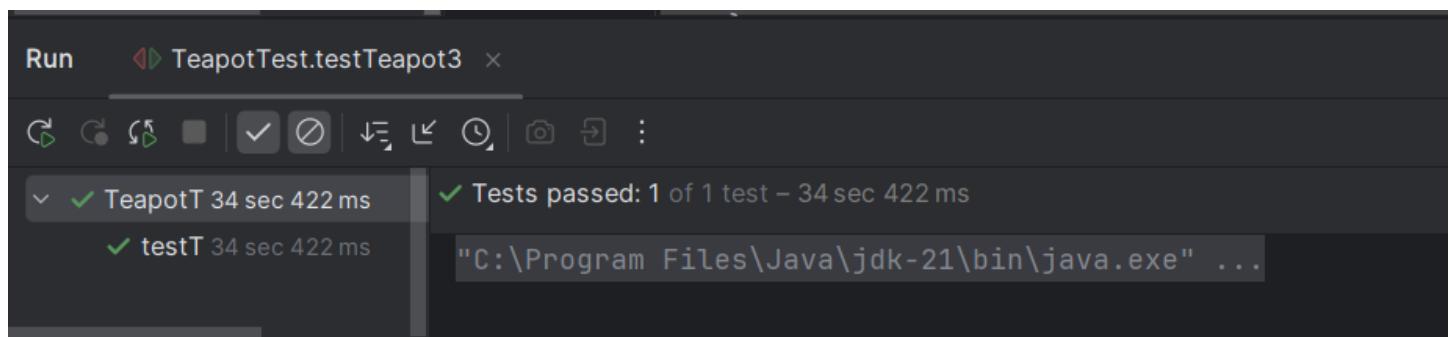
✓ testTeapot (8 min 13 sec) "C:\Program Files\Java\jdk-21\bin\java.exe" ...

Process finished with exit code 0

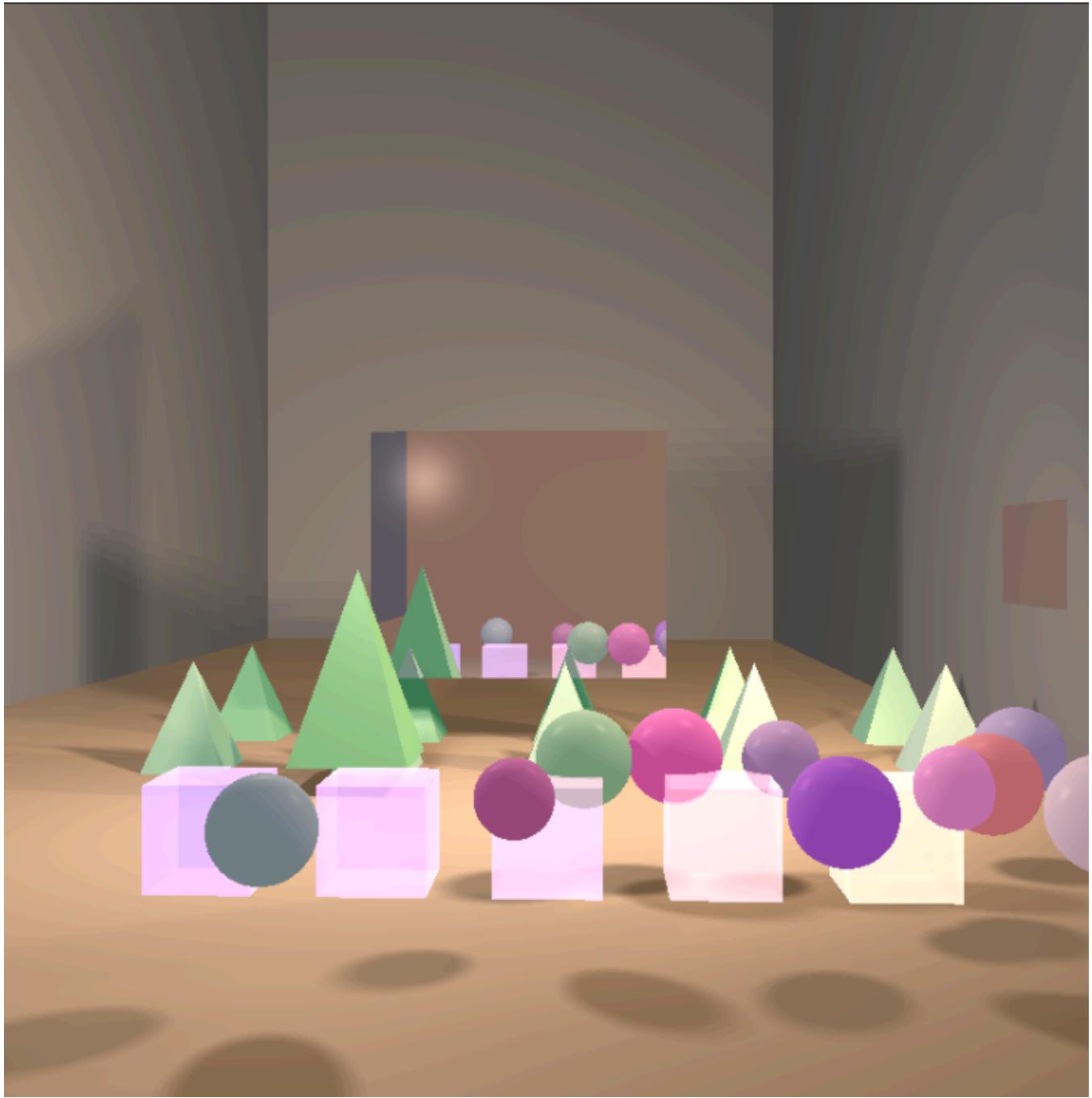
:CBR עם הצאת



:BVH עם הצאת H



תמונה נוספת:



בלי האצה:

A screenshot of a terminal window titled "Run" showing the output of a test run. The title bar says "Minip1.createSceneWithMultipleGeometriesSoftShadows". The terminal shows the following text:
Tests passed: 1 of 1 test – 10 min 17 sec
C:\Users\molev\.jdks\openjdk-21.0.2\bin\java.exe ...
Process finished with exit code 0

עם האצת תהיליכונים:

```
Run Minip2.createSceneWithMultipleGeometriesSoftShadows_...
Tests passed: 1 of 1 test – 4 min 3 sec
C:\Users\molev\.jdks\openjdk-21.0.2\bin\java.exe ...
100.0%
Process finished with exit code 0
```

עם האצת BVH

```
Run Minip2.createSceneWithMultipleGeometriesSoftShadows_...
Tests passed: 1 of 1 test – 2 min 45 sec
C:\Users\molev\.jdks\openjdk-21.0.2\bin\java.exe ...
100.0%
Process finished with exit code 0
```

עם האצת BVH ידני

```
Run Minip2.createSceneWithBVHManual
Tests passed: 1 of 1 test – 37 sec 661 ms
C:\Users\molev\.jdks\openjdk-21.0.2\bin\java.exe ...
100.0%
Process finished with exit code 0
```

עם האצת BVH + תהליכיונים מרובים:

```
Run Minip2.createSceneWithMultipleGeometriesSoftShadows_...
Tests passed: 1 of 1 test – 42 sec 746 ms
C:\Users\molev\.jdks\openjdk-21.0.2\bin\java.exe ...
100.0%
Process finished with exit code 0
```

עם האצת CBT + תהליכיונים מרובים:

Run Minip2.createSceneWithMultipleGeometriesSoftShadows_... X

Minip2 (renderer) 50 sec 117 ms

createSceneWithMul 50 sec 117 ms

Tests passed: 1 of 1 test – 50 sec 117 ms

C:\Users\molev\.jdks\openjdk-21.0.2\bin\java.exe ...

100.0%

Process finished with exit code 0

בונוסים:

שלב 3 - בונוס - חיתוך עם מצולע - לציון בלבד | grading only



הוגש על ידי:

אפרת ישן ורוחמה בריקר

