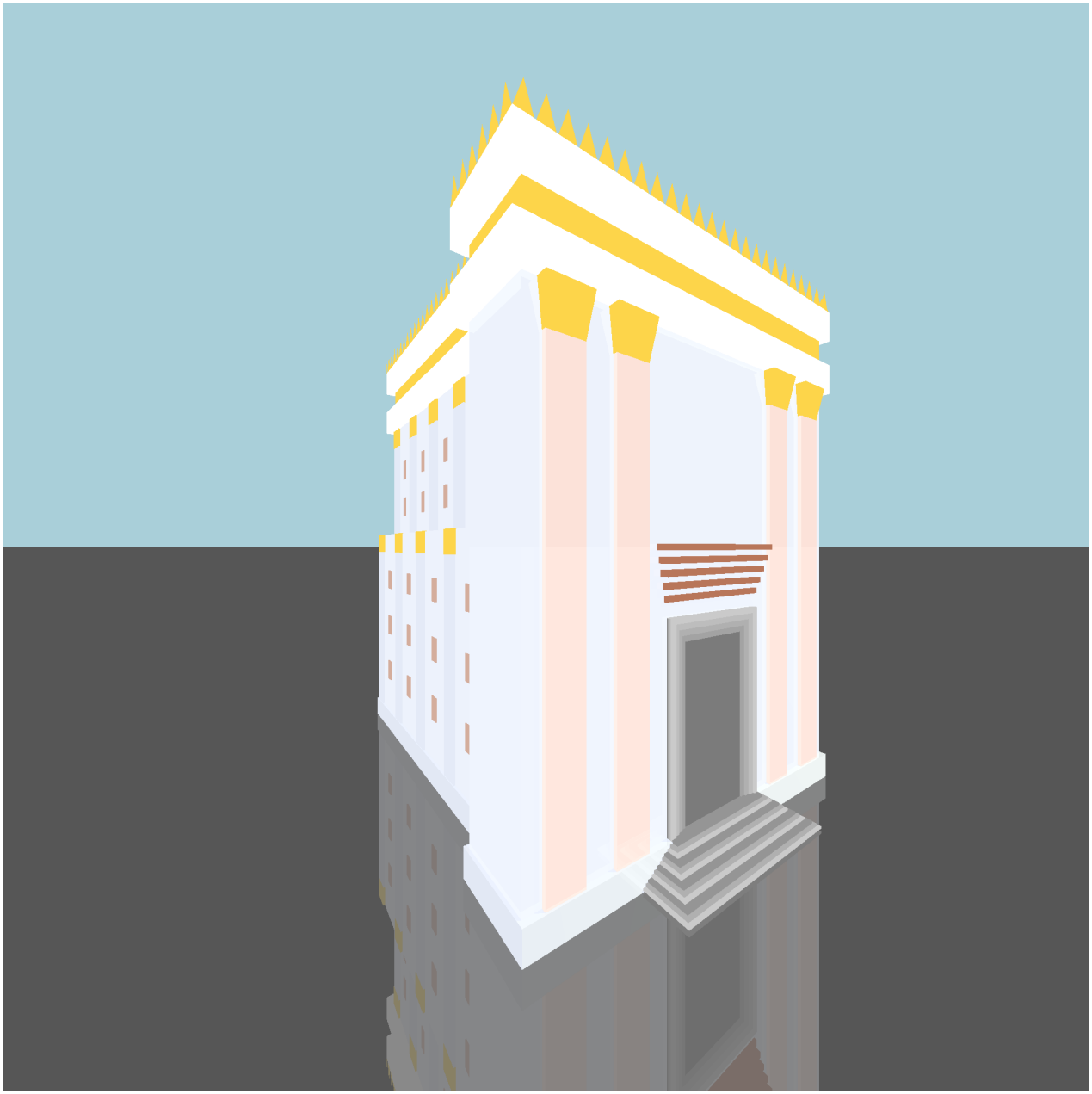
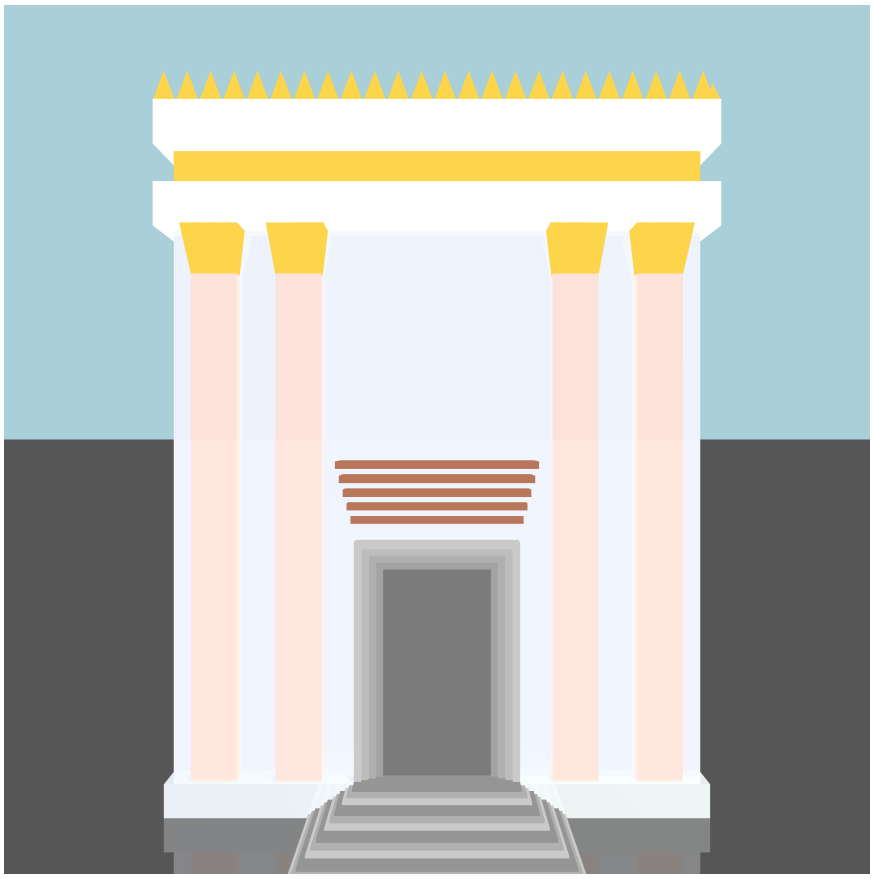
דו"ח מעבדה

"עלו ההר והבאתם עץ ובנו הבית וארצה בו ואכבד אמר ה'." ~ ספר חגי, פרק א, פסוק ח'



הילה בוזנח-212127435
חנה לאה סילברברג -211416250

תוכן עניינים



Antialiasing

החלקת עקומות - בגלל שעד היום השתמשנו במרכז הפיקסל בלבד לחישוב הצבע שלו, דבר זה יצר לנו קצוות משוננים.

איך נשפר? נעביר כמה קרניים לכל פיקסל, ונעשה ממוצע של הצבעים המוחזרים מהם.

הדלקה וכיבוי של המתודה נעשה ע"י בחירת פונקצייה לרינדור (בחירת renderImageSuperSampling לעומת renderImage)

נשתמש באלגוריתם Super-sampling, אלגוריתם זה ישלח קרניים למקומות רנדומליים בפיקסל, ונרנדר את התמונה לפי ממוצע הצבעים שקרניים אלו יחזירו.

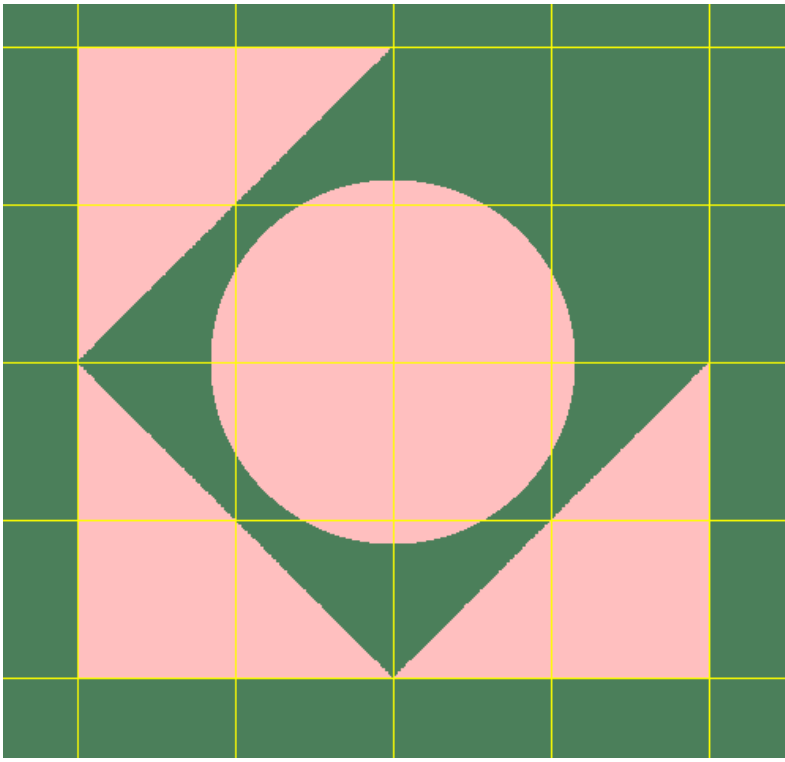
הפונקציה castBeamSuperSampling תשלח קבוצת קרניים (beam) למרכז הפיקסל, ותחשב את הממוצע. וconstructBeamSuperSampling שתיצור קבוצת קרניים רנדומלית בהן נשתמש.

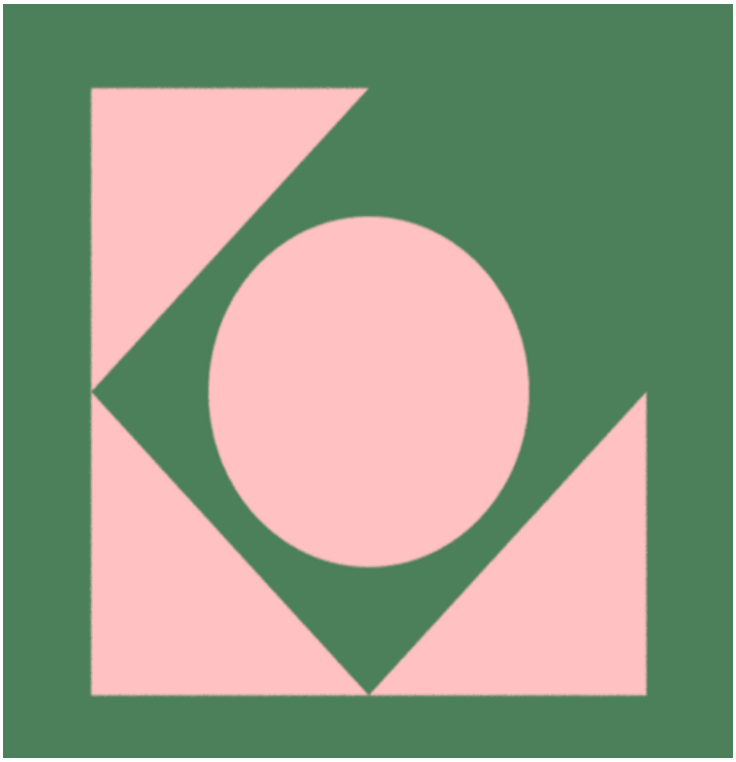
```
private List<Ray> constructBeamSuperSampling(int nX, int nY, int j, int i) {
    List<Ray> beam = new LinkedList<>();
    beam.add(constructRay(nX, nY, j, i));
    double ry = height / nY;
    double rx = width / nX;
    double yScale = alignZero( number: (j - nX / 2d) * rx + rx / 2d);
    double xScale = alignZero( number: (i - nY / 2d) * ry + ry / 2d);
    Point pixelCenter = p0.add(vTo.scale(distance)); // center
    if (!isZero(yScale))
        pixelCenter = pixelCenter.add(vRight.scale(yScale));
    if (!isZero(xScale))
        pixelCenter = pixelCenter.add(vUp.scale( scaleFactor: -1 * xScale));
    Random rand = new Random();
    // create rays randomly around the center ray
    for (int c = 0; c < nSS; c++) {
        // move randomly in the pixel
        double dxfactor = rand.nextBoolean() ? rand.nextDouble() : -1 *
                rand.nextDouble();
        double dyfactor = rand.nextBoolean() ? rand.nextDouble() : -1 *
                rand.nextDouble();
        double dx = rx * dxfactor;
        double dy = ry * dyfactor;
        Point randomPoint = pixelCenter;
        if (!isZero(dx))
            randomPoint = randomPoint.add(vRight.scale(dx));
        if (!isZero(dy))
            randomPoint = randomPoint.add(vUp.scale( scaleFactor: -1 * dy));
        beam.add(new Ray(p0, randomPoint.subtract(p0)));
    }
```

```
1 usage   Administrator
private Color castBeamSuperSampling(int j, int i) {
    List<Ray> beam = constructBeamSuperSampling(imageWriter.getNx(), imageWriter.getNy(), j, i);
    Color color = Color.BLACK;
    // calculates the average colour of rays traced
    for (Ray ray : beam) {
        color = color.add(rayTracer.traceRay(ray));
    }
    return color.reduce(nSS);
}
```

לפני ואחרי(ימין לשמאל):

ניתן לראות כי השיפור בהחלט "מחליק" את הקצוות המשוננים.





Adaptive SuperSampling

לעיתים קרובות, ובייחוד עם השיפור הקודם , שיפור ה -antialiasing (ובמיוחד אצלינו בתמונה הסופית), נשלח קרניים רבות לאזורים בהם יש צבע אחיד, ויקרה שנשלח קרניים רבות לחשב אותו צבע שוב ושוב (סוג של needless complexity).

איך נשפר? נחלק את הפיקסל (דגימה) ל 4 חלקים קטנים יותר, נשווה בין הצבעים של החלקים, אם הצבעים אותו דבר - נצא, אין מה להמשיך בשליחת קרניים לאזורים בעלי צבע אחיד, אחרת - נפצל וכך נציג דיוק רב יותר בצבע.

מימשנו את השיפור באמצעות 3 פונקציות:

דומה לרינדור הרגיל - renderImageAdaptiveSuperSampling

קיבוץ רנדומלי של קבוצת קרניים למרכז ושולחת לפונקציית חישוב-castBeamAdaptiveSuperSampling ממוצע

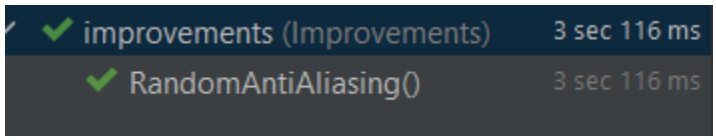
```
/**
 * Casts a ray of beams using adaptive super sampling
 * @param j col
 * @param i row
 * @return average colour of pixel
 */
2 usages  Administrator
private Color castBeamAdaptiveSuperSampling(int j, int i) {
    Ray center = constructRay(imageWriter.getNx(), imageWriter.getNy(), j, i);
    Color centerColor = rayTracer.traceRay(center);
    return calcAdaptiveSuperSampling(imageWriter.getNx(), imageWriter.getNy(), j, i, maxLevelAdaptiveSS, centerColor);
}
```

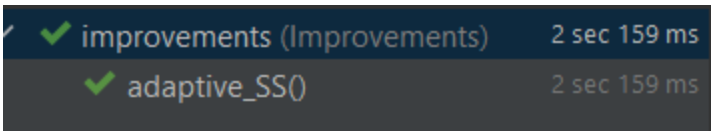
חישוב ממוצע ושליחת הקרניים - CalcAdaptiveSuperSampling

```
2 usages  Administrator
private Color calcAdaptiveSuperSampling(int nX, int nY, int j, int i, int maxLevel, Color centerColor) {
    if (maxLevel <= 0) {
        return centerColor;
    }
    Color color = centerColor;
    // divide pixel into 4 mini-pixels
    Ray[] beam = new Ray[]{constructRay( Nx: 2 * nX,  Ny: 2 * nY,  j: 2 * j,  i: 2 * i),
            constructRay( Nx: 2 * nX,  Ny: 2 * nY,  j: 2 * j,  i: 2 * i + 1),
            constructRay( Nx: 2 * nX,  Ny: 2 * nY,  j: 2 * j + 1,  i: 2 * i),
            constructRay( Nx: 2 * nX,  Ny: 2 * nY,  j: 2 * j + 1,  i: 2 * i + 1)};
    for (int ray = 0; ray < 4; ray++) {
        Color currentColor = rayTracer.traceRay(beam[ray]);
        if (!currentColor.equals(centerColor))
            currentColor = calcAdaptiveSuperSampling( nX: 2 * nX,  nY: 2 * nY,
                    j: 2 * j + ray / 2,  i: 2 * i + ray % 2, (maxLevel - 1), currentColor);
        color = color.add(currentColor);
    }
    return color.reduce( k: 5);
}
```

תוצאות שיפור Adaptive Supersampling :

לפני ואחרי (מימין לשמאל):





Boundary Volume Hierarchy

השיפור: לכל צורה, ניצור אזור תוחם, שמרני, כך שנוכל מהר מאוד לבדוק האם הקרן חותכת אותו או לא. וכך לא נכנס לכל צורה את החיתוכים שלה במקרה שאין צורך (בשאין חיתוכים). ונבדוק רק במקרה שהאזור התוחם נחתך.

על מנת לממש , ניצור מחלקה פנימית ל - intersectable בשם Bounding box , שתייצג את הקופסא התוחמת.

```
/**
 * class representing boundary box
 */
4 usages   Administrator
public class BoundingBox {
    7 usages
    public Point minimums;  //Borders of box
    7 usages
    public Point maximums;  //Box borders

    3 usages   Administrator
    public BoundingBox(Point mins, Point maxs) {
        minimums = mins;
        maximums = maxs;
    }

}
/**
```

ונוסיף שדות מתאימים בIntersectable , :

כדי להפעיל ולכבות את השיפור - bvhIsOn

```
11 usages
protected boolean bvhIsOn = true;  //a field to turn on and off the bvh

19 usages
public BoundingBox box; //Boundary box

/**
```

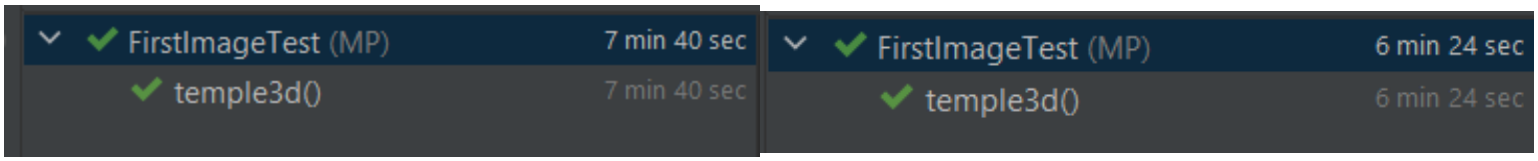
בנוסף, נוסיף פונקציה אבסטרקטית - createBoundingBox , שכל צורה גיאומטרית תדרש לממש כך שתהיה לה קופסא תוחמת מתאימה.

ולבסוף - פונקציה שתחשב לנו את החיתוך של הקופסא, באמצעותה נדע אם להמשיך לבדוק חיתוך או לא.

```
1 usage   Administrator
public boolean isIntersectingBoundingBox(Ray ray) {
    if (!bvhIsOn || box == null)    //Intersect as usual
        return true;
    Vector dir = ray.getDir();
    Point p0 = ray.getP0();
    double tMin = (box.minimums.getX() - p0.getX()) / dir.getX();
    double tMax = (box.maximums.getX() - p0.getX()) / dir.getX();
    if (tMin > tMax) {
        double temp = tMin;
        tMin = tMax;
        tMax = temp;
    }
    double tyMin = (box.minimums.getY() - p0.getY()) / dir.getY();
    double tyMax = (box.maximums.getY() - p0.getY()) / dir.getY();
    if (tyMin > tyMax) {
        double temp = tyMin;
        tyMin = tyMax;
        tyMax = temp;
    }
    if ((tMin > tyMax) || (tyMin > tMax))
        return false;
    if (tyMin > tMin)
        tMin = tyMin;
    if (tyMax < tMax)
        tMax = tyMax;
    double tzMin = (box.minimums.getZ() - p0.getZ()) / dir.getZ();
    double tzMax = (box.maximums.getZ() - p0.getZ()) / dir.getZ();
    if (tzMin > tzMax) {
        double temp = tzMin;
        tzMin = tzMax;
        tzMax = temp;
    }
    if ((tMin > tzMax) || (tzMin > tMax))
        return false;
    if (tzMin > tMin)
        tMin = tzMin;
    if (tzMax < tMax)
        tMax = tzMax;
    return true;
}
```

תוצאות שיפור הBVH:

לפני ואחרי (מימין לשמאל):



הערה: בדיקה זו נעשה עם multithreading , ללא multithreading ההבדל בין הזמנים היה זהה,

תודות וקרדיטים:

קוד של BVH:

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-accelerationstructure/bounding-volume-hierarchy-BVH-part1

קוד של CalcAdaptive - שירה כהן.

מצגות הקורס - Dr. Elishai Ezra Tsur Dan Zilberstein.

בדיקת ערכי גבול ושקילות - ד"ר אליעזר גנסבורגר

בנוסף, תודה לד"ר אליעזר גנסבורגר על כל ההשקעה שלו בקורס, זכינו ללמוד המון ממנו על כתיבת קוד נקי ויעיל, ושיפור קוד, מעריכות מאוד!

בהמשך: שיפורים לקוד ולתאורה, ואור ניבט מן החלונות, ובע"ה בית מקדש אמיתי:)