דו"ח סופי מיני פרויקט 1

מטרת מיני פרויקט 1: שיפור התמונה על ידי התמקדות באזור מסויים של התמונה וטשטוש של השאר.

Depth of field

הרעיון: במקום לשלוח קרן 1 לכל פיקסל בview plane, נשלח הרבה קרניים שיטשטשו חלק מהתמונה

המימוש: הוספנו למצלמה פרמטר בשם depthOfField שמגדיר view plane וירטואלי במרחק של depthOfField מהview plane המקורי.
בזמן רינדור התמונה, במקום שתצא קרן רק ממיקום המצלמה לכיוון הפיקסל, נוציא הרבה קרניים כאשר כל קרן תתחיל מסביבה קרובה למיקום המצלמה ותעבור דרך הview plane הווירטואלי בנקודה שבה הקרן המקורית חתכה את הנ"ל.
הוספנו גם פרמטר בשם aperture שמגדיר את המרחק המקסימלי של הקרניים הנוספות שיוצאות, ממיקום המצלמה.
בנוסף, קבענו פרמטר בשם n שקובע את מספר הקרניים שיצאו מאזור המצלמה – מספר הקרניים שיצאו הוא n^2. (בעצם נוצר סביב מיקום המצלמה מעין view plane שמחולק לפיקסלים לפי aperture*2/n וכל פעם **יוצאת** קרן מפיקסל אחר).
להלן הקוד שממש את הנ"ל:

```
8 usages   Yisrael jacob
public Camera renderImageWithDepthOfField() {
    if (imageWriter == null)
        throw new MissingResourceException("You need to enter a image writer", ImageWriter.class.getName(), "");
    if (rayTracer == null)
        throw new MissingResourceException("You need to enter a ray tracer", RayTracerBase.class.getName(), "");

    int nX = this.imageWriter.getNx();
    int nY = this.imageWriter.getNy();
    if(!splitToThreads) {
        for (int row = 0; row < nY; row++) {
            for (int col = 0; col < nX; col++) {
                Ray myRay = constructRay(nX, nY, col, row);
                List<Ray> myRays = constructRaysGridFromCamera(n, myRay);
                Color myColor = new Color( r: 0,  g: 0,  b: 0);
                for (Ray ray : myRays) { // we pass in the list myRays and for each ray we found his color
                    myColor = myColor.add(rayTracer.traceRay(ray)); // we add the color of each ray to myColor
                }
                imageWriter.writePixel(col, row, myColor.reduce(myRays.size())); // we reduce myColor with the size of my list (number o
            }
        }
    }
```

2 usages Yisrael jacob

```java
public List<Ray> constructRaysGridFromCamera(int n, Ray ray) { // we construct a square around the circle of the camera, the size n=2*r
    // we launch a ray (we choose a random _focusPoint in the pixel)  from each pixel of the grid, and
    // we select only the ray IN the circle of the camera

    List<Ray> myRays = new LinkedList<>(); //the list of all the rays

    double t0 = depthOfField + distance; // distance from the central focusPoint of the camera to the focus focusPoint
    double t = t0 / (vTo.dotProduct(ray.getDir())); // distance from the focusPoint on the aperture grid to the focus focusPoint ( foun
    Point focusPoint = ray.getPoint(t); // we found the focus focusPoint

    double pixelSize = alignZero( number: (aperture * 2) / n); // the size of each pixel

    // we construct a ray from each pixel of the grid, and we select only the rays in the circle
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            Ray tmpRay = constructRayFromPixel(n, n, j, i, pixelSize, focusPoint);
            // we check if each ray is in the circle of the camera
            if (tmpRay.getP0().equals(p0)) {// if the ray is from the camera center
                myRays.add(tmpRay); // we add the ray to the list myRays
            } else if (tmpRay.getP0().subtract(p0).dotProduct(tmpRay.getP0().subtract(p0)) <= aperture * aperture) {
                // if the distance with the center (squared) is <= the square of the radius -> the ray is in the circle of the camera
                myRays.add(tmpRay); // we add the ray to the list myRays
            }
        }
    }
    return myRays; // we return  the list of all my rays in the circle
}
```

1 usage Yisrael jacob

```java
private Ray constructRayFromPixel(int nX, int nY, double j, double i, double pixelSize, Point focusPoint) {

    Point pIJ = p0;

    Random r = new Random(); // we want a random point for each pixel for more precision

    double xJ = ((j + r.nextDouble() / (r.nextBoolean() ? 2 : -2)) - ((nX - 1) / 2d)) * pixelSize;
    double yI = -((i + r.nextDouble() / (r.nextBoolean() ? 2 : -2)) - ((nY - 1) / 2d)) * pixelSize;

    if (xJ != 0) {
        pIJ = pIJ.add(vRight.scale(xJ));
    }
    if (yI != 0) {
        pIJ = pIJ.add(vUp.scale(yI));
    }

    Vector vIJ = focusPoint.subtract(pIJ);

    return new Ray(pIJ, vIJ); // return a new ray from a pixel
}
```

סצנה למיני פרויקט 1:

יצרנו סצנה של חדר המורכב מ6 משטחים(polygon). בתוך החדר 9 שולחנות + שולחן של מורה כאשר כל אחד מהם מורכב מ:

1. 2 משטחים (polygon) בשביל הפלטה של השולחן שרחוקים אחד מהשני כמרחק עובי השולחן.
2. 4 מלבנים בשביל למלא את עובי השולחן מכל צד.
3. 4 רגליים כאשר כל רגל מורכבת מ4 מלבנים ל4 הצדדים.

בנוסף, בתוך החדר ישנם כסא ליד כל שולחן כאשר כל אחד מהם מורכב מ:

1. מלבן שמייצג את מושב הכסא
2. מלבן שמייצג את משענת הכסא
3. 4 רגליים כמו בשולחנות

על השולחן של המורה ועל השולחן האמצעי בכיתה מונחים כדור על כל אחד מהם.

בקיר הימני ישנה מראה שמראה את החלק הקידמי של הכיתה (כמובן שזה תלוי במיקום וכיוון המצלמה).

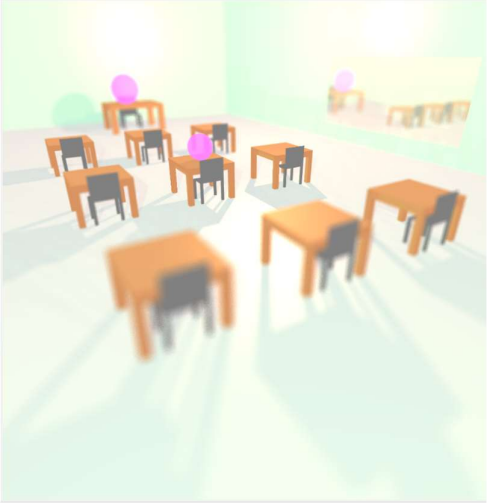
כמו כן בסצנה שישה מקורות אור:

- Directional light שלא משפיע על התמונה כי החדר אטום עם קירות, תקרה ורצפה
- Point light שנמצא בערך באמצע החדר קרוב לתקרה
- Spot light שנמצא גם באמצע החדר וכיוונו לכיוון הימני קידמי של הכיתה וכלפי מטה
- 3Xspot light שמצאים ברצפה צמוד לקיר האחורי ומכוונים כלפי קדימה

ניתן לראות את ההבדלים לאחר השיפור depth of field בכך שמרכז התמונה ברורה והשאר מטושטשת

תמונה לפני השיפור

תמונה לאחר השיפור



דו"ח סופי מיני פרויקט 2

שיפור ביצועים - Boundary Volume Hierarchy(BVH):

השיפור: יצרנו עץ בינארי שמחזיק תיבות כך ש: השורש מחזיק תיבה שמכילה את כל הגופים בסצנה וכל בן מחזיק תיבה קטנה יותר שמכילה פחות גופים העלים של העץ מחזיקים תיבה שמכילה גוף אחד בלבד.

העץ מאפשר לנו בצורה מהירה לזהות אילו אובייקטים נמצאים בתחום מסוים של התצוגה. במקום לבדוק את כל האובייקטים בצורה ישירה, אנו יכולים לבדוק רק את הקופסאות בעץ ולגלות אילו קופסאות מתכנסות עם התחום המבוקש. זה מאפשר לנו לחסוך הרבה זמן ומשאבים בתהליך הצגת התמונה או הסרטון, מאחר ואנחנו פועלים רק עם חלק מהאובייקטים הנמצאים בסצנה.

המימוש: במחלקה Intersectable הגדרנו מחלקה פנימית בשם BoundingBox ששומרת את הקואורדינטות המינימליים והמקסימליים של התיבה שנוצרת סביב הגוף/גופים.

```
public static class BoundingBox {
    16 usages
    public Point minimums;  //Borders of box
    16 usages
    public Point maximums;  //Box borders

    4 usages   Yisrael jacob
    public BoundingBox(Point mins, Point maxs) {
        minimums = mins;
        maximums = maxs;
    }

}
```

הוספנו מתודה שבודקת האם התיבה אכן נחתכת על פי אלגוריתם של חיתוך קרן עם תיבה:

```java
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.getCoordinate().getX() - p0.getCoordinate().getX()) / dir.getCoordinate().getX();
    double tMax = (box.maximums.getCoordinate().getX() - p0.getCoordinate().getX()) / dir.getCoordinate().getX();
    if (tMin > tMax) {
        double temp = tMin;
        tMin = tMax;
        tMax = temp;
    }
    double tyMin = (box.minimums.getCoordinate().getY() - p0.getCoordinate().getY()) / dir.getCoordinate().getY();
    double tyMax = (box.maximums.getCoordinate().getY() - p0.getCoordinate().getY()) / dir.getCoordinate().getY();
    if (tyMin > tyMax) {    //Swapping if the bottom is larget than the top
        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.getCoordinate().getZ() - p0.getCoordinate().getZ()) / dir.getCoordinate().getZ();
    double tzMax = (box.maximums.getCoordinate().getZ() - p0.getCoordinate().getZ()) / dir.getCoordinate().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;
}
```

בנוסף, הוספנו במחלקה Geometries, מימוש של יצירת עץ של תיבות ומתודה שמחזירה את הגופים שהקרן הרלוונטית פוגעת בתיבות שהגופים הנ"ל מוכלים בתוכה.

```java
public static Geometries buildBVH(Geometries geometries) {
    if (geometries.items.isEmpty()) {
        return null;
    }
    return recursiveBuildBVH(geometries, start: 0, end: geometries.items.size() - 1);
}
3 usages  Yisrael jacob
private static Geometries recursiveBuildBVH(Geometries geometries, int start, int end) {
    if (start == end) {
        // Create a leaf node
        Geometries myGeo = new Geometries(geometries.box, left: null, right: null, geometries.items,
                geometries.items.get(0));
        myGeo.setBvhIsOn(true);
        return myGeo;
    }

    // Find the axis with the longest extent
    int longestAxis = findLongestAxis(geometries, end: end - start);

    // Sort geometries based on the longest axis
    geometries.items.sort(Comparator.comparingDouble(a -> getCentroid(a, longestAxis)));

    int mid = (start + end)/ 2;


    // Recursively build left and right subtrees
    Geometries left = recursiveBuildBVH(new Geometries(geometries.items.subList(0, mid - start + 1)), start, mid);
    Geometries right = recursiveBuildBVH(new Geometries(geometries.items.subList(mid - start + 1, end - start + 1)), start: mid + 1, end);

    // Calculate the bounding box for the current node
    BoundingBox boundingBox = calculateBoundingBox(geometries.items, end: end - start);
    double minx = boundingBox.minimums.getCoordinate().getX() -0.01;
    double miny = boundingBox.minimums.getCoordinate().getY()-0.01;
    double minz = boundingBox.minimums.getCoordinate().getZ()-0.01;
    double maxx = boundingBox.maximums.getCoordinate().getX()+0.01;
    double maxy = boundingBox.maximums.getCoordinate().getY()+0.01;
    double maxz = boundingBox.maximums.getCoordinate().getZ()+0.01;

    // Create an internal node
    Geometries myGeo = new Geometries(boundingBox, left, right, geometries.items, new Polygon(
            new Point(minx, miny, minz),
            new Point(minx, miny, maxz),
            new Point(maxx, maxy, maxz),
            new Point(maxx, maxy, minz)));
    myGeo.setBvhIsOn(true);
    myGeo.intersectable.setBvhIsOn(true);
    return myGeo;
}
```

```java
@Override
    public List<GeoPoint> findGeoIntersections(Ray ray) {
    if (box == null)
        return findGeoIntersectionsHelper(ray);
    List<GeoPoint> geometries1 = new LinkedList<>();
    if (intersectable.isIntersectingBoundingBox(ray)) {
        if (isLeaf()) {
            // Perform intersection test with individual geometries at the leaf node
            return findGeoIntersectionsHelper(ray);
        } else {
            // Intersect with left and right child nodes recursively
            List<GeoPoint> leftIntersect = left.findGeoIntersections(ray);
            List<GeoPoint> rightIntersect = right.findGeoIntersections(ray);

            if (leftIntersect != null)
                geometries1.addAll(leftIntersect);
            if(rightIntersect != null)
                geometries1.addAll(rightIntersect);
        }
    }
    if (geometries1.size() == 0)
        return null;

    // If the ray does not intersect the bounding box of the node, return false
    return geometries1;
}
```

תוצאות שיפור הBVH:

```
without BVH took 21802 mile seconds

with BVH took 14979 mile seconds
```

שיפור ביצועים – Multithreading

השיפור: חלוקת חישוב צבע הפיקסלים לתהליכונים כך שכמה פיקסלים יחושבו במקביל.

המימוש: באמצעות stream שמקבל את מספר הפיקסלים ומחלק אותם לתהליכונים.

```
//rendering image with using of threads
IntStream.range(0, nY).parallel().forEach(row ->
        IntStream.range(0, nX).parallel().forEach(col -> {
            Ray myRay = constructRay(nX, nY, col, row);
            List<Ray> myRays = constructRaysGridFromCamera(n, myRay);
            Color myColor = new Color( r: 0,  g: 0,  b: 0);
            for (Ray ray : myRays) { // we pass in the list myRays and for each ray we found his color
                myColor = myColor.add(rayTracer.traceRay(ray)); // we add the color of each ray to myColor
            }
            imageWriter.writePixel(col, row, myColor.reduce(myRays.size())); // we reduce myColor with the size of my list (num
        })
);
```

תוצאות שיפור הMultithreading:

```
without multithreading took 17860 mile seconds

with multithreading took 8245 mile seconds
```