דוח פרויקט - אלעד פישר(213924624) ונריה מחפוד(213919616)

מידע כללי על הפרויקט:

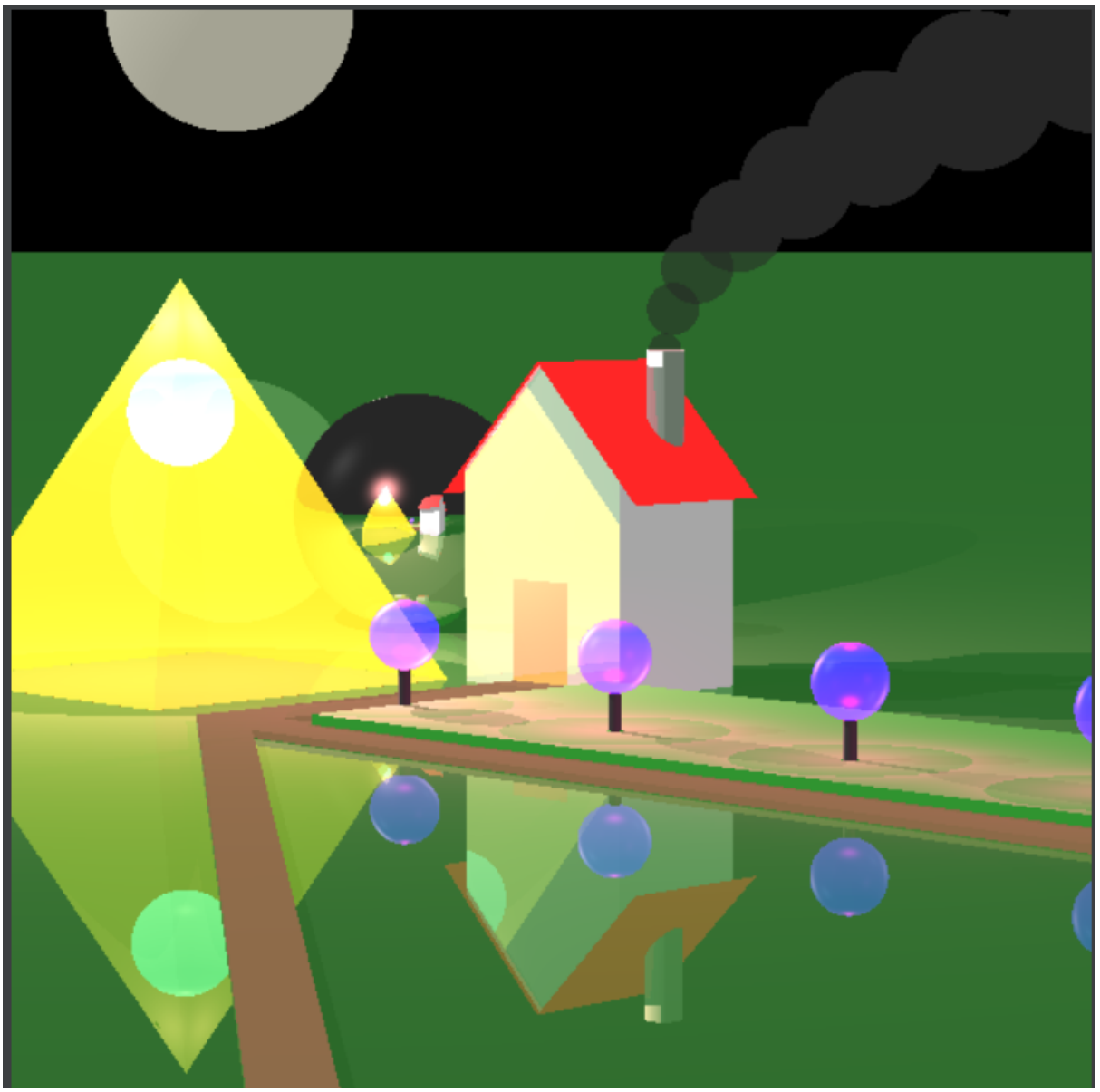
- יש גטרים וסטרים לכל השדות שיש כולל תיאור של הפרמטר, הסטרים הם בשיטה של chaining method מתי שביקשו(בעיקר במצלמה ורנדרר).
- השתמשנו בעיקרון של ניצול של קוד ושיטות בזה שלקחנו את הקוד שעובד מהמצגות של המרצים(בעיקר מצגת 7) וגם לקחנו את הפונקציה שמוצאת את הנקודות חיתוך של גליל סופי ואינסופי מזוג אחר בקבוצה(דרזנר ורפאל גולדטיין) על מנת להשתמש בהם בתמונה הסופית, וכדי לממש את העיקרון שבקוד שניסו בדכ יש פחות שגיאות.
- השתדלנו בנוסחאות להשתמש בשמות שיש במצגות על מנת שמי שייראה את הנוסחה יוכל בקלות לראות מה כל פרמטר, בשאר הפרוייקט ואיפה שראינו שהשמות לא טובים, בחרנו בשמות משמעותיים.
- הוספנו הערות לפי פורמט java Doc איפה שצריך(לפני כל מחלקה, פונקציה, שדה), וכמו כן הוספנו הערות בכל מקום שראינו שהקוד לא ברור מעצמו.
- הוספנו מחלקת Box שהיא מקבלת נקודת התחלה ושלוש ווקטורים, אחד לכל כיוון, כדי ליצור תיבה תלת ממדית, או 3 ערכים בשביל ליצור תיבה שהצלעות מקבילות לצירים.
- בונוסים כלליים שעשינו לאורך הפרויקט:
- תרגיל 2:מימוש נורמל לגליל סופי
- תרגיל 3:חיתוך עם מצולע
- תרגיל 4:טרנספורמציית מצלמה(הוספנו את זה רק בתרגיל 7, אך הבונוס לא מוגבל בזמן)
- תרגיל 6:סיום תוך שבוע
- תרגיל 7:כל הבונוסים:תמונה עם מעל 10 אובייקטים, סיבוב התמונה וצילום מכמה זוויות, מימוש בעיית ההצללה בדרך השניה.
- מיני פרוייקט 1:מימשנו שני שיפורי תמונה:עומק שדה, אנטי אלייסינג(החלקת עקומות).

הסבר על הבעיות והשיפורים של מיני פרוייקט 1:

הבעיה:

הקצוות של התמונה לפעמים חדות מאוד ואז התמונה נראית לא טבעית כמו שיש כאן דוגמא לתמונה לא טבעית:

חלק מהתמונה הסופית של "ירח" שנראה קצת מפוקסל בקצוות



התמונה המלאה, פה ניתן לראות שיש קצוות מאוד חדים בעיקר בשביל לכיוון המצלמה, הפירמידה נראית עם "מדרגות", ובכללי יש עוד מקומות כמו ההשתקפות דרך הכדור השחור, הגג של הבית וההצללה שיש פשוט קווים מפוקסלים ברמות.

התיקון:

אנחנו נשלח מספר מסויים של קרניים לכל פיקסל ולכן מה שיהיה בקצוות ייראה למעשה בצורה יותר עדינה, של שילוב הצבעים לפי כמה הפיקסל קרוב לקצה, וככה זה ייראה בצורה חלקה יותר עם הצבעים.

כדי לשלוח מספר קרניים לכל פיקסל יש כמה שיטות:

1. להשתמש בשיטת מונטה קרלו לחלק פיקסל לחלקים ולקבל נקודה אקראית בכל "תת פיקסל"
2. אקראי לחלוטין.
3. לעשות גריד של נקודות.

אנחנו בחרנו לעשות גריד של נקודות כי זה יחסית מדוייק ונותן חלוקה טובה יחסית בלי "שטחים מתים" שאין בהם קרן (הכללנו את הקצוות של הפיקסל כדי לקבל חלוקה מאוד מדוייקת).

בכללי אנחנו חושבים שעל פי העיקרון RDD האחריות של היצירה של הקרניים היא של המצלמה שמעלה ככה את האיכות, והרנדרר אחראי רק לרנדר את התמונה בפועל בצורה היבשה.
לכן שינינו את והוספנו את היצירה של הקרניים אצל המצלמה, וגם מימשנו את היצירה של הגריד אצל המצלמה כי זה משמש רק אותו וזה אחריות שלו.

השינויים בקוד ובתצורה של הפרוייקט:

שינינו את הפונקציה `constructRaysThroughPixel` שתחזיר רשימה של קרניים עבור אותו פיקסל, ואת הרנדרר שינינו שידע לרנדרר רשימה של קרניים לכל פיקסל ולעשות מהם את הממוצע:
השינוי של הרנדרר:

```
for (int i = 0; i < nX; ++i)
    for (int j = 0; j < nY; ++j)
    {
        Color color = getAverageColor(_camera.constructRaysThroughPixel(nX, nY, i, j));
        _imageWriter.writePixel(i, j, color);
    }
```

בפעולת הרנדור זה יכתוב את הצבע הממוצע שמתקבל ע"י רשימת הקרניים.

```
/** helper function that calculate the average color o
private Color getAverageColor(List<Ray> rays) {
    Color c = Color.BLACK;

    // for each ray in rays we check the color and add
    for (Ray r : rays) {
        c = c.add(_rayTracerBase.traceRay(r));
    }


    return c.reduce(rays.size());
}
```

חישוב הממוצע של הצבע של הקרניים.
השינוי שעשינו לתהליך יצירת הקרניים במצלמה:

```
List<Ray> rays = null;

if (raysSampling != 1)
    rays = createRaySampling(Pij, ratioY, ratioX);

else
    rays = List.of(new Ray(p0, Pij.subtract(p0)));
```

כאן אם רוצים לירות יותר מקרן אחת לפיקסל אנחנו למעשה קוראים לפונקציה שתיצור את הרשימה עבור הפיקסל המסויים ושולחים גם את הגובה והרוחב של הפיקסל.

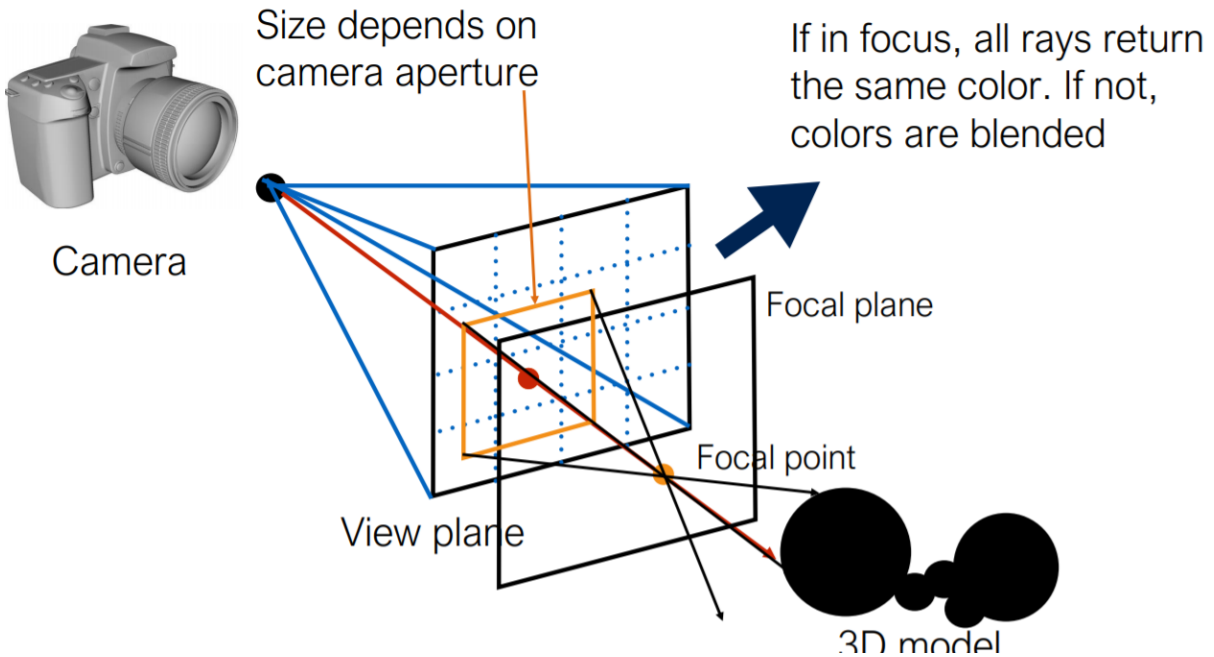
תהליך יצירת הקרניים לכל פיקסל:

```
    //find the 4 corners of the pixel
    Point3D ru = pixel.add(vUp.scale(pixelHeight / 2)).add(vRight.scale(pixelWidth / 2));
    Point3D rd = pixel.add(vUp.scale(-pixelHeight / 2)).add(vRight.scale(pixelWidth / 2));
    Point3D lu = pixel.add(vUp.scale(pixelHeight / 2)).add(vRight.scale(-pixelWidth / 2));
    Point3D ld = pixel.add(vUp.scale(-pixelHeight / 2)).add(vRight.scale(-pixelWidth / 2));


    List<Ray> res = new LinkedList<>();

    List<Point3D> pointsInPixel = generatePointsInPixel(raysSampling, ru, rd, lu, ld);

    //create all the rays that intersect the VB through the points
    for (Point3D p : pointsInPixel)
    {
        res.add(new Ray(point3D, p.subtract(point3D)));
    }

    return res;
}
```

בעיה שנייה:

התמונה כולה בפוקוס, ולכן זה לא נראה מציאותי, כמו כן לפקס רק את מה שאני רוצה שיהיה בולט כמו העצם המרכזי בתמונה זה מאוד חשוב לתמונות עתירות פרטים וצורות.

התיקון: נעשה עוד מישור שכל מה שעליו יהיה בפוקוס, אנחנו נגידר את המישור בתור המרחק בינו לבין הview plane, כך שמה שעל המישור החדש יהיה בפוקוס.
אנחנו נגדיר גם "צמצם" של המצלמה שלנו כך שכמה שהוא יהיה יותר גדול, מה שלא יהיה בפוקוס יהיה יותר מטושטש.



המחשה של הרעיון עם הצמצם והמרחק.

השינויים בקוד:

הרנדור יעבוד באותה דרך, כי אני אחזיר רשימה של קרניים.
למצלמה יתווספו השדות הרלונטיים של הנתונים שהגדרתי לעיל.

הקוד של החישוב קרניים דרך פיקסל ישתנו ככה:

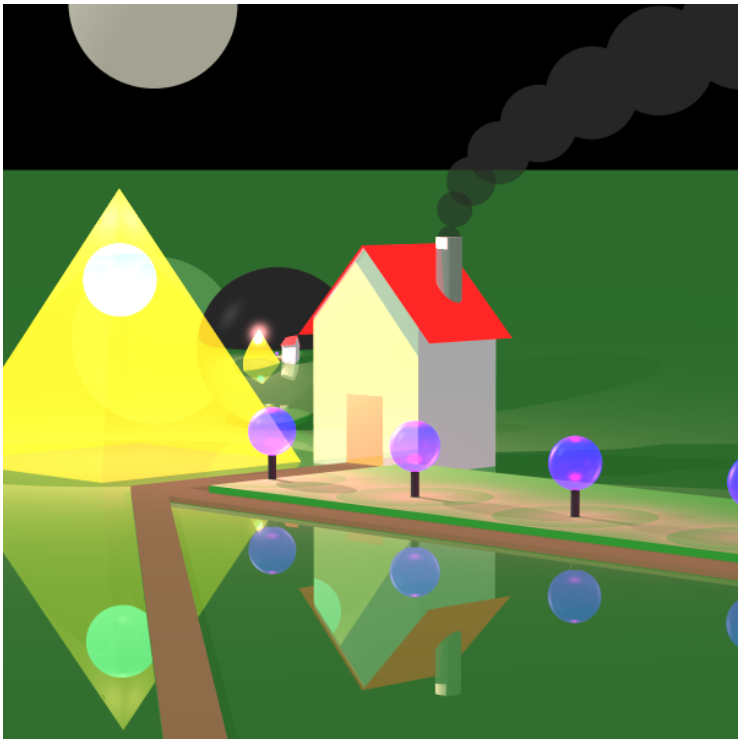
```java
private List<Ray> createDepthOfFieldRays(Point3D pij, Ray ray) {

    //calc the focal point
    Point3D focalPoint = focalPlane.findIntersections(ray).get(0);

    //calc the edges of the aperture
    Point3D ru = pij.add(vUp.scale(apertureSize / 2)).add(vRight.scale(apertureSize / 2));
    Point3D rd = pij.add(vUp.scale(-apertureSize / 2)).add(vRight.scale(apertureSize / 2));
    Point3D lu = pij.add(vUp.scale(apertureSize / 2)).add(vRight.scale(-apertureSize / 2));
    Point3D ld = pij.add(vUp.scale(-apertureSize / 2)).add(vRight.scale(-apertureSize / 2));

    //get points on the aperture
    List<Point3D> gridPoints = generatePointsInPixel(samplingDepth, ru, rd, lu, ld);

    /** a beam of rays that go through the pixel and intersect the focal point. */
    List<Ray> r = new LinkedList<Ray>();
    r.add(ray);

    //create all the focused rays
    for (Point3D p : gridPoints)
    {
        r.add(new Ray(p, focalPoint.subtract(p)));
    }

    return r;
```

החישוב של הקרני עומק שדה יתבצע ע"י קבלה של הקרן(זה חשוב על מנת לתמוך גם בעומק שדה וגם בהחלקת קצוות) והנקודה שהיא חותכת את הview plane (על מנת לחשוך בביצועים).

השינוי בקבלת הקרניים לכל פיקסל יתבטא בזה שניצור רשימה של קרני עומק שדה עבור כל קרן שיש בכללי.

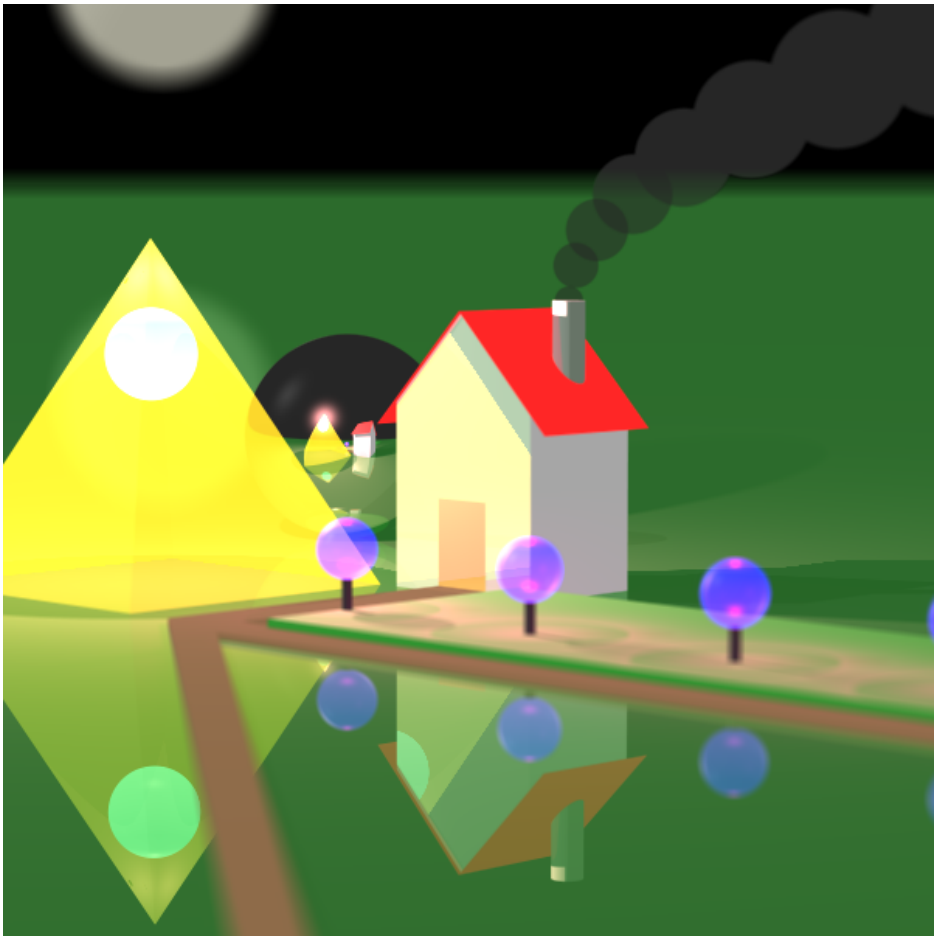
```java
    //if there is focus generate and return the focused rays
    if (0 != focalPlaneDist && samplingDepth != 1)
    {
        List<Ray> DOFrays = new LinkedList<>();
        focalPlane = new Plane(vTo, getCenterPoint().add(vTo.scale(focalPlaneDist)));

        //calc and add all the DOF rays
        for (Ray r: rays)
        {
            DOFrays.addAll(createDepthOfFieldRays(Pij,r));
        }

        //return all the DOF rays
        return DOFrays;
    }


    //if there isn't focus return the rays that calculated before no matter if the rays
    else
        return rays;
}
```

תמונה סופית:



תמונה סופית עם anti aliasing.

תמונה סופית עם עומק שדה:



תמונה סופית בלי שום שיפור: