מיני פרויקט 1- שיפור תמונה

מגישות: אביה אלפסי 315435768
סימא פיליפס 211358338

הבעיה:

קצוות חדים- קצוות הגופים בתמונה אינם חלקים. התמונות נראות מאד מפוקסלות. בעיה זו נוצרת כתוצאה מכך שקבענו את הצבע בפיקסל להיות הצבע שמתקבל מהקרן שנשלחת מהמצלמה דרך מרכז הפיקסל לאובייקטים בסצנה. בקצוות הגופים ישנו מצב שחלק מהפיקסל מכסה גוף מסוים וחלק ממנו לא.

הפתרון:

שימוש ב- Super Sampling-

להחליף כל קרן שנשלחת בהפצה של קרניים. במקום לשלוח דרך כל פיקסל קרן אחת בלבד, נשלח הרבה קרניים דרך חלקים שונים בפיקסל. נחשב את הממוצע המשוקלל של הצבעים שהתקבלו מכל הקרניים שנוצרו לצבע אחד, והוא יהיה הצבע של הפיקסל. ניתן לשלוח את הקרניים על ידי הזזה רנדומלית של הקרן המקורית שעברה דרך מרכז הפיקסל, אך אנחנו בחרנו לעשות Super Sampling על ידי Grid- חילקנו כל פיקסל להרבה "מיני-פיקסלים" בדיוק כמו שחילקנו בזמנו את המסך לרשת של פיקסלים. בעינינו צורה זו היתה יותר סימטרית ומדויקת.

כדי לאפשר הדלקה וכיבוי של תכונת ה-Super Sampling הוספנו במחלקה Render משתנה בוליאני. כמו כן, הוספנו משתנה קבוע ששומר את מספר הקרניים שיועברו דרך הפיקסל. (במקרה ששמור 9 יהיו 9 בשניה קרניים, כלומר 81):

```
private boolean superSampling;
private static final int NUM_OF_SAMPLE_RAYS = 9; //81 rays through pixel for super sampling
```

כתבנו פונקציה חדשה במחלקה Camera ששמה ConstructRaysThroughPixel. הפונקציה מקבלת ארגומנטים שונים וביניהם מספר הקרניים הרצויים וכך יודעת כמה קרניים ליצור. עבור כל פיקסל במסך היא קוראת לפונקציית עזר בעלת שם זהה שמחזירה את רשימת הקרניים שעוברים דרך אותו פיקסל. הפונקציה מחזירה רשימה של כל הקרניים בסצנה.

```
public List<Ray> constructRaysThroughPixel(int nX, int nY, int j, int i, double screenDistance,
                                           double screenWidth, double screenHeight, int num_of_sample_rays)
{
    //The distance between the screen and the camera cannot be 0
    if (isZero(screenDistance))
    {
        throw new IllegalArgumentException("distance cannot be 0");
    }

    List<Ray> sample_rays = new ArrayList<>();

    double Ry = screenHeight/nY; //The number of pixels on the y axis
    double Rx = screenWidth/nX;  //The number of pixels on the x axis
    double yi =  ((i - nY/2d)*Ry); //distance of original pixel from (0,0) on Y axis
    double xj=   ((j - nX/2d)*Rx); //distance of original pixel from (0,0) on x axis
    double pixel_Ry = Ry/num_of_sample_rays; //The height of each grid block we divided the parcel into
    double pixel_Rx = Rx/num_of_sample_rays; //The width of each grid block we divided the parcel into

    for (int row = 0; row < num_of_sample_rays; ++row) {//foreach place in the pixel grid
        for (int column = 0; column < num_of_sample_rays; ++column) {
            sample_rays.add(constructRaysThroughPixel(pixel_Ry,pixel_Rx,yi, xj, row, column,screenDistance));//add the ray
        }
    }
    sample_rays.add(constructRayThroughPixel(nX, nY, j, i, screenDistance, screenWidth, screenHeight));//add the center screen ray
    return sample_rays;
}
```

פונקציית העזר מחלקת את הפיקסל שקיבלה ל-Grid ועבור כל משבצת קוראת לפונקציה הרגילה ConstructRayThroughPixel ושולחת קרן דרכו:

```
private Ray constructRaysThroughPixel(double Ry,double Rx, double yi, double xj, int j, int i, double screenDistance)
{
    Point3D Pc = _p0.add(_vTo.scale(screenDistance)); //the center of the screen point

    double y_sample_i =  (i *Ry + Ry/2d); //The pixel starting point on the y axis
    double x_sample_j=   (j *Rx + Rx/2d); //The pixel starting point on the x axis

    Point3D Pij = Pc; //The point at the pixel through which a beam is fired
    //Moving the point through which a beam is fired on the x axis
    if (!Util.isZero( number: x_sample_j + xj))
    {
        Pij = Pij.add(_vRight.scale(x_sample_j + xj));
    }
    //Moving the point through which a beam is fired on the y axis
    if (!Util.isZero( number: y_sample_i + yi))
    {
        Pij = Pij.add(_vUp.scale(-y_sample_i -yi ));
    }
    Vector Vij = Pij.subtract(_p0);
    return new Ray(_p0,Vij);//create the ray throw the point we calculate here
}
```

במחלקה Render הגדרנו את הפונקציה superSampling. במקרה ש המשתנה הבוליאני superSampling הוא בעל ערך אמת, הפונקציה RenderImage תשתמש בו:

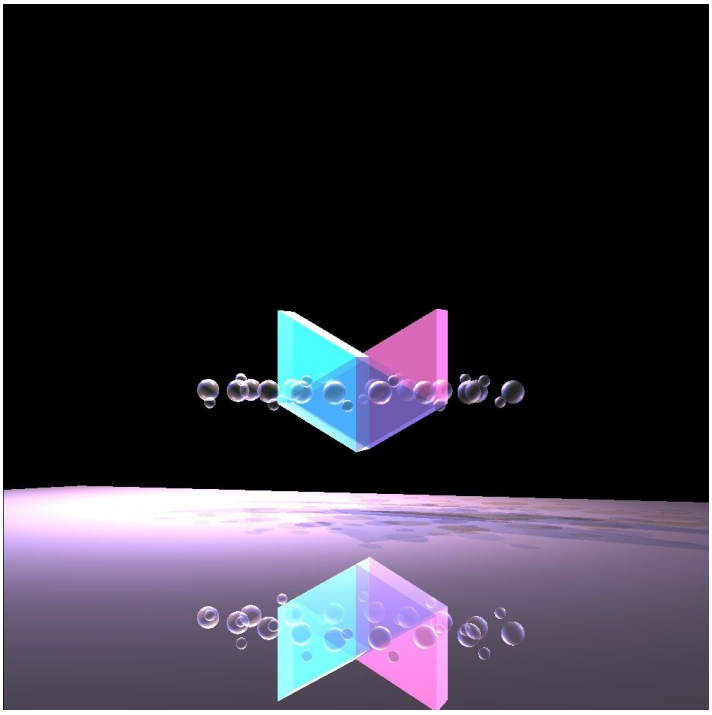
```
private Color superSampling(Camera camera, double distance, int width, int height, int nx, int ny, Pixel pixel) {
    List<Ray> rays = camera.constructRaysThroughPixel(nx, ny, pixel.col, pixel.row, distance, width, height, NUM_OF_SAMPLE_RAYS);  //calculate rays throw this pixel
    return calcColor(rays);  //calculate the color of the pixel
}
```

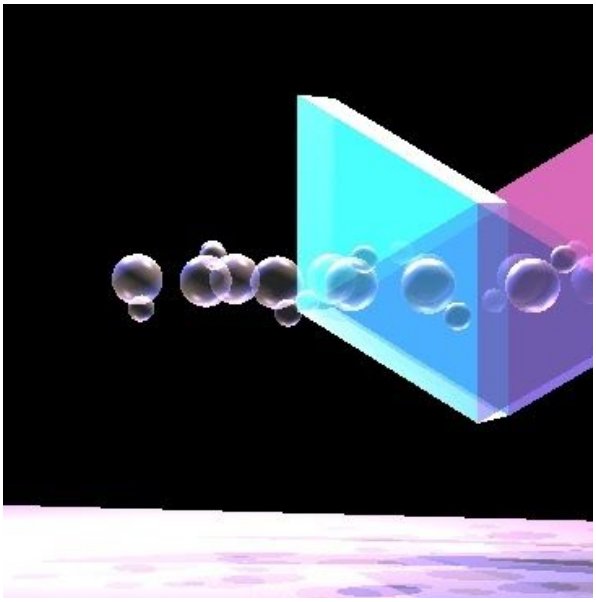
היה צורך בפונקציה שתחשב את הצבע שהוא הממוצע המשוקלל של אוסף הקרניים העוברים דרך פיקסל. הוספנו פונקציה בשם CalcColor למחלקה Render, שמקבלת רשימת קרניים:

```
private Color calcColor(List<Ray> superSamplingRays){
    Color color = Color.BLACK;
    for(Ray ray: superSamplingRays){    //Calculate the color for each of the rays
        color = color.add(calcColor(ray));  //sum up the color scheme
    }
    color = color.reduce(superSamplingRays.size());   //Reduces the result color by the number of rays in the list (average calculation)
    return color;
}
```

תוצאה סופית:

התמונה ללא שיפור:





התמונה לאחר שיפור: