Avigail Tenenbaum 5520

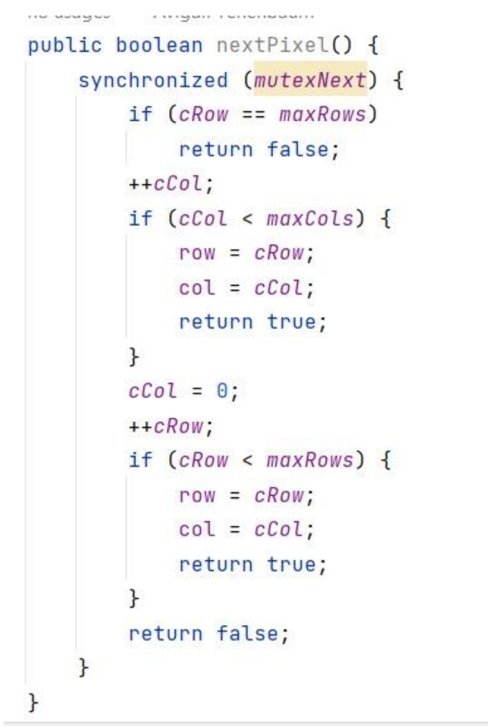
Noa Landman 7160

Introductory software engineering mini project reportIntroduction:

In this part of the report our goal is to show the improvements we made to our project focusing on the efficiency of the ray tracing process.
We did so by adding two components, first, we added multi-threading to the ray tracing process, secondly, we added an improved Anti-Aliasing method Called Adaptive Anti-Aliasing. We will elaborate on both of them shortly.

Threads:

up to this point we rendered our image using a single thread, to improve CPU utilization for a faster rendering time , we added multi-threading to our render image function, our main concern though is that the process of multi-threading has to be done safely so we do not have two threads tracing and coloring the same pixel, to do so we added a class Pixel.java {@credit Author Dan Zilbershtein} that is responsible on designating the next pixel to be rendered, while securing the separate threads from causing errors.
The main function in the class is the nextPixel() method:

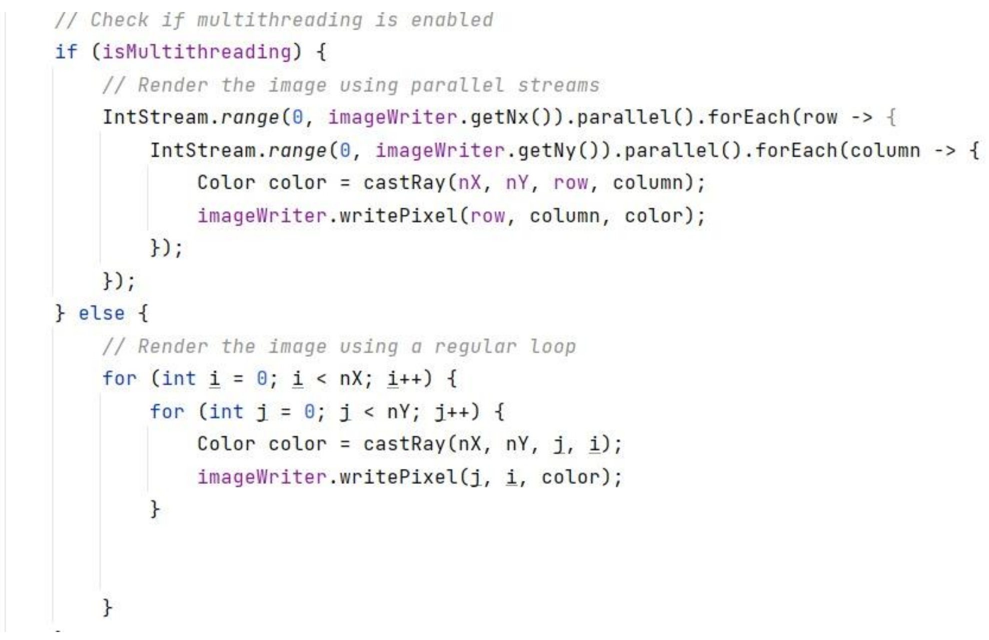


```java
public boolean nextPixel() {
    synchronized (mutexNext) {
        if (cRow == maxRows)
            return false;
        ++cCol;
        if (cCol < maxCols) {
            row = cRow;
            col = cCol;
            return true;
        }
        cCol = 0;
        ++cRow;
        if (cRow < maxRows) {
            row = cRow;
            col = cCol;
            return true;
        }
        return false;
    }
}
```

which moves us to next pixel to render, but is done synchronized ,

assuring a correct ray tracing process.

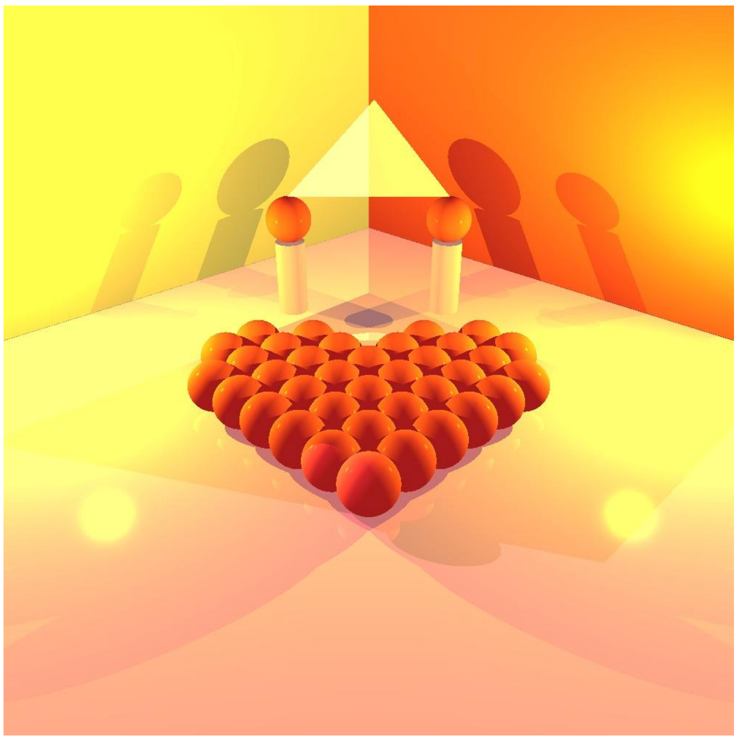
We then adjusted our renderImage() function to support multi-threading:



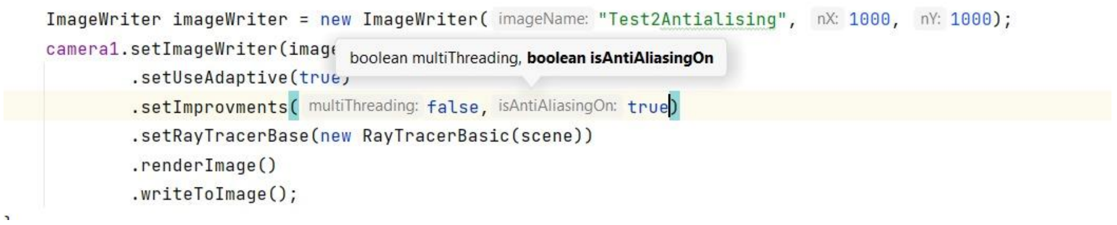
```java
// Check if multithreading is enabled
if (isMultithreading) {
    // Render the image using parallel streams
    IntStream.range(0, imageWriter.getNx()).parallel().forEach(row -> {
        IntStream.range(0, imageWriter.getNy()).parallel().forEach(column -> {
            Color color = castRay(nX, nY, row, column);
            imageWriter.writePixel(row, column, color);
        });
    });
} else {
    // Render the image using a regular loop
    for (int i = 0; i < nX; i++) {
        for (int j = 0; j < nY; j++) {
            Color color = castRay(nX, nY, j, i);
            imageWriter.writePixel(j, i, color);
        }


    }
```

Runtime improvement using threads:

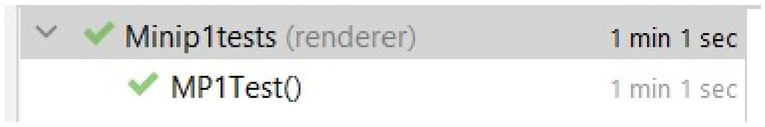
endered this image using 1000x1000 pixels and the adaptive anti aliasing method with recursion level of 4 .



With no multi-threading – runtime:

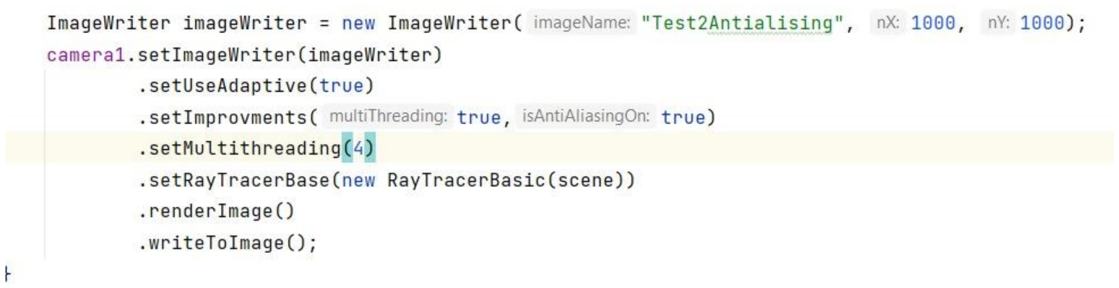


```
ImageWriter imageWriter = new ImageWriter( imageName: "Test2Antialising",  nX: 1000,  nY: 1000);
camera1.setImageWriter(image  boolean multiThreading, boolean isAntiAliasingOn
        .setUseAdaptive(true)
        .setImprovments( multiThreading: false,  isAntiAliasingOn: true)
        .setRayTracerBase(new RayTracerBasic(scene))
        .renderImage()
        .writeToImage();
```

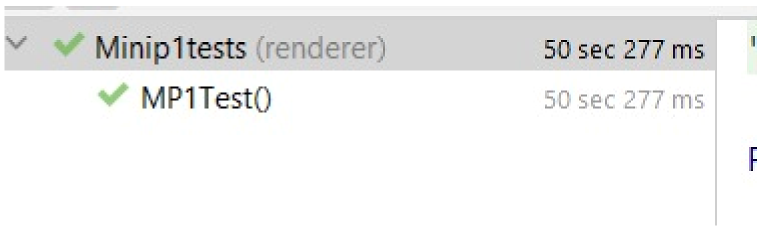


```
Minip1tests (renderer)    1 min 1 sec
    MP1Test()             1 min 1 sec
```

With multi-threading – using four threads:



```
ImageWriter imageWriter = new ImageWriter( imageName: "Test2Antialising",  nX: 1000,  nY: 1000);
camera1.setImageWriter(imageWriter)
        .setUseAdaptive(true)
        .setImprovments( multiThreading: true,  isAntiAliasingOn: true)
        .setMultithreading(4)
        .setRayTracerBase(new RayTracerBasic(scene))
        .renderImage()
        .writeToImage();
```



```
Minip1tests (renderer)    50 sec 277 ms
    MP1Test()             50 sec 277 ms
```

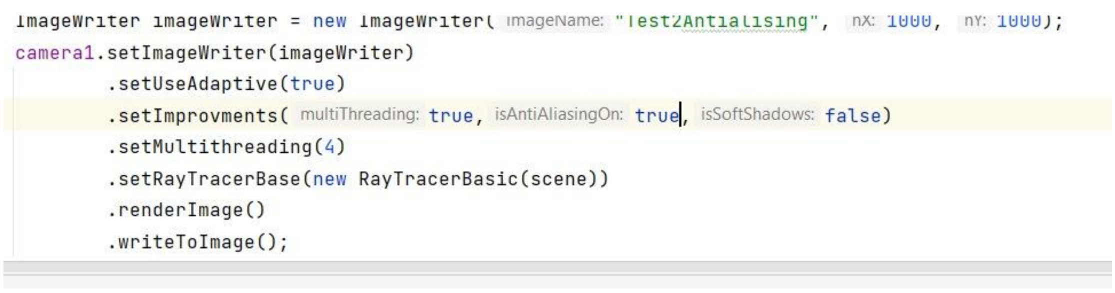
Choosing to use multi-threading cut our runtime by 11 seconds!

Choosing The Improvement:

The decision of what improvement to use we designed to be extremely easy.

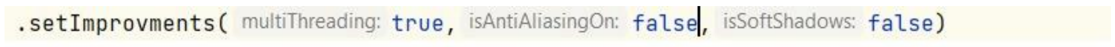
we will show how changing a single line in the test case will render the same image with different methods, requiring no changes to the actual image.

This code will render the image using the adaptive method:



```
ImageWriter imageWriter = new ImageWriter( imageName: "Test2Antialising",  nX: 1000,  nY: 1000);
camera1.setImageWriter(imageWriter)
        .setUseAdaptive(true)
        .setImprovments( multiThreading: true,  isAntiAliasingOn: true,  isSoftShadows: false)
        .setMultithreading(4)
        .setRayTracerBase(new RayTracerBasic(scene))
        .renderImage()
        .writeToImage();
```

By switching one line:

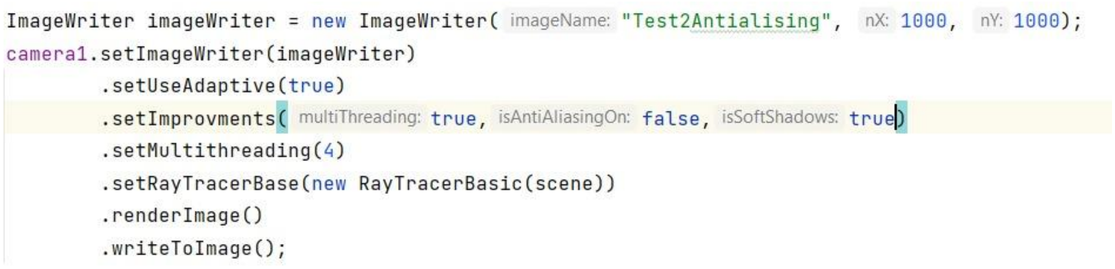


```
.setImprovments( multiThreading: true,  isAntiAliasingOn: false,  isSoftShadows: false)
```

Now the image will be rendered without any improvements.

If we want to use SoftShadows improvement for example instead,

We will turn off the Anti-Aliasing and instead turn on the softShadows:



```
ImageWriter imageWriter = new ImageWriter( imageName: "Test2Antialising",  nX: 1000,  nY: 1000);
camera1.setImageWriter(imageWriter)
        .setUseAdaptive(true)
        .setImprovments( multiThreading: true,  isAntiAliasingOn: false,  isSoftShadows: true)
        .setMultithreading(4)
        .setRayTracerBase(new RayTracerBasic(scene))
        .renderImage()
        .writeToImage();
```

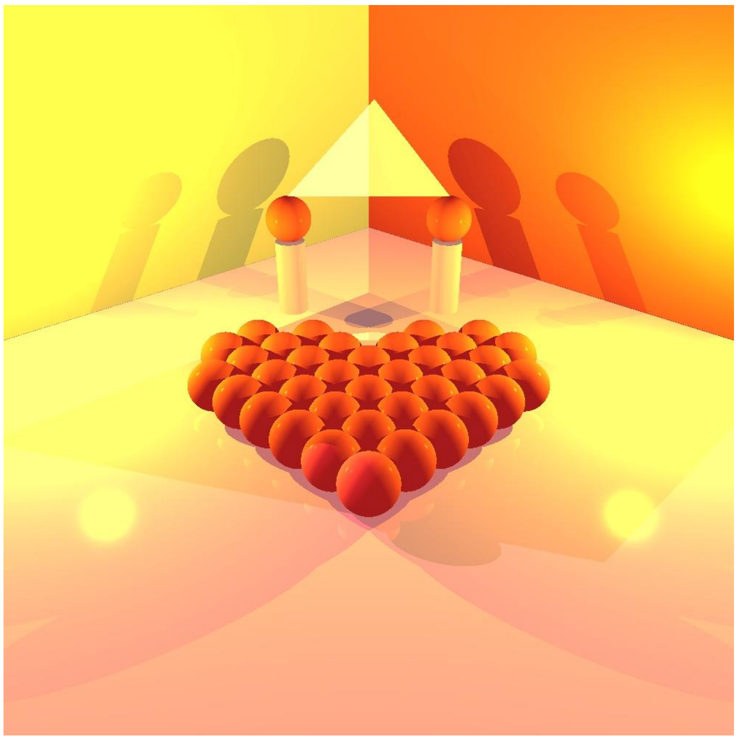
and so forth, allowing a very swift change between the options.

Introduction:

In this part of the report our goal is to demonstrate the methodologies we used to create our picture, and to show the two improvements we added to our project/picture.

The requirement was to create a scene that includes multiple geometries, with a few light sources added to the scene, which emphasizes the shading affects across the scene.

Our thought process in creating the picture was to integrate all the functionalities.



In our scene there is a room, in the center of the room there are 36 spheres placed in the shape of a heart.

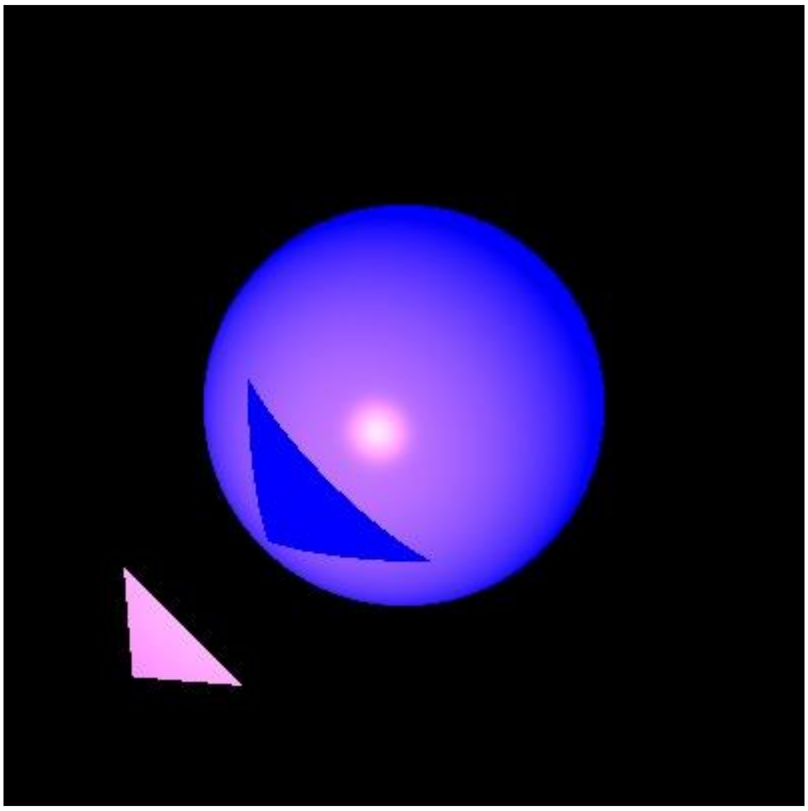
Behind the heart are two cylinders with 2 spheres above which are figures holding a triangle.

The scene has 3 different light sources

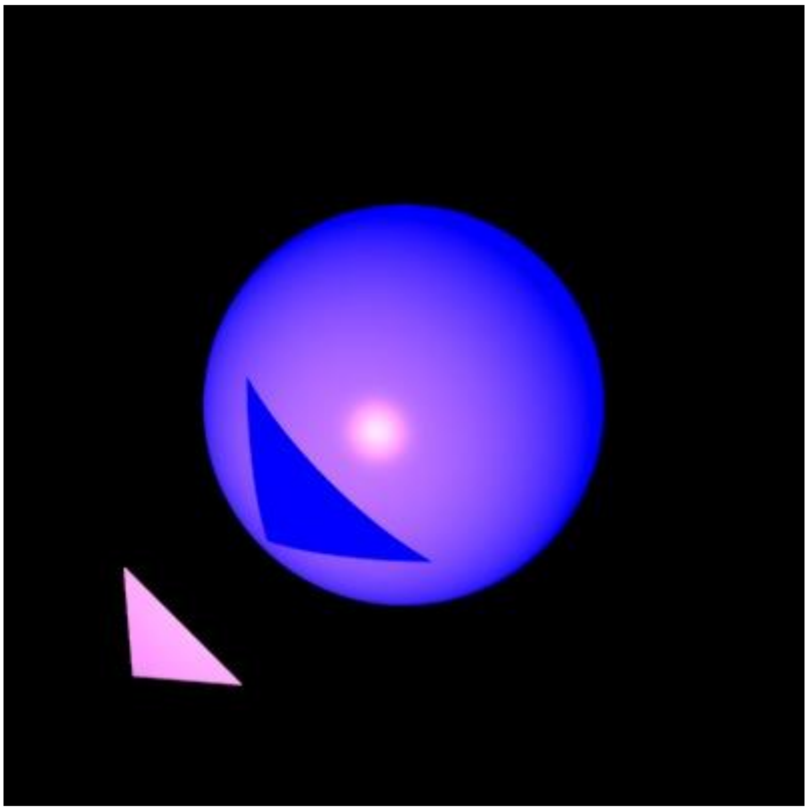
Improvements:

Anti-Aliasing:

Before:



After:



You can see that the edges of the triangle are not smooth but jagged.

This happens because we sample the center of the pixels to get its color,

And we continue to paint the whole pixel with the same color, and it's not

By chance all the time, the edge of the table may cover only part of it

the pixel, but the whole pixel is incorrectly colored in the color of the tables.

To solve this problem, we added Anti-Aliasing functionality to our camera.

Soft Shadow:

Another issue we noticed is that if we take a closer look at the shadows in

the scene, we notice that they are very "hard", meaning, that we have an

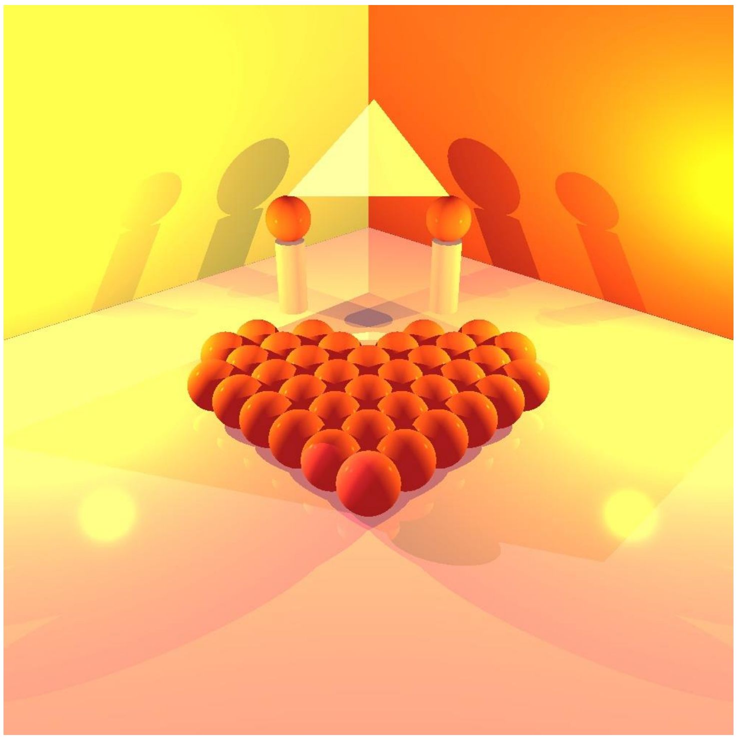
area that is shadowed completely and then right next to it we have an

unshaded area with a very clear line between them, which does not

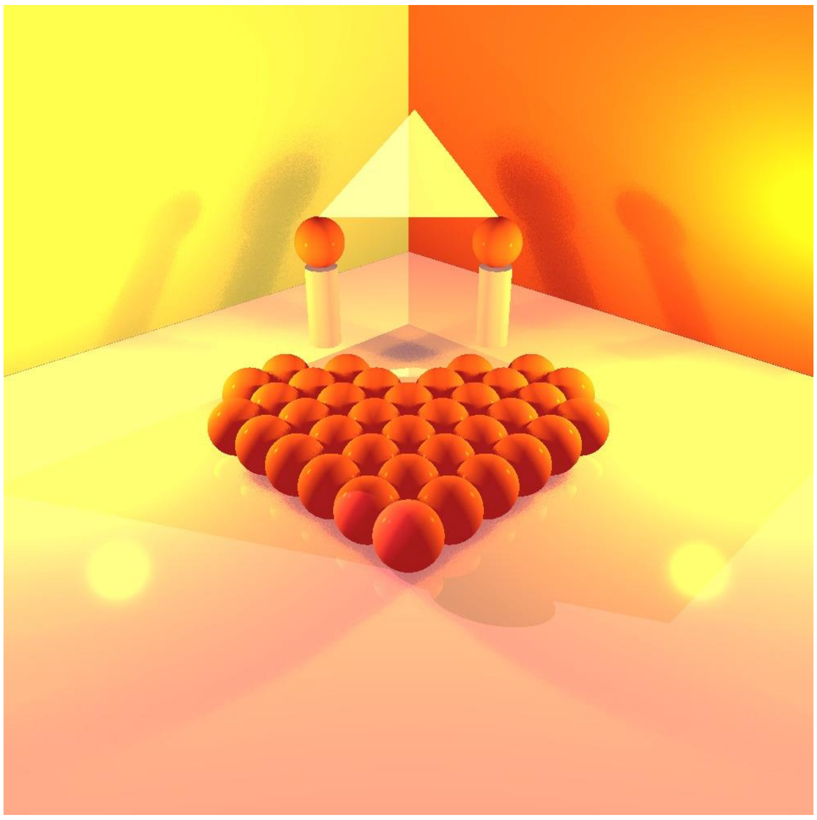
resemble shadow in the real world were the shadow fades slowly until the

area is completely unshaded.

Before:



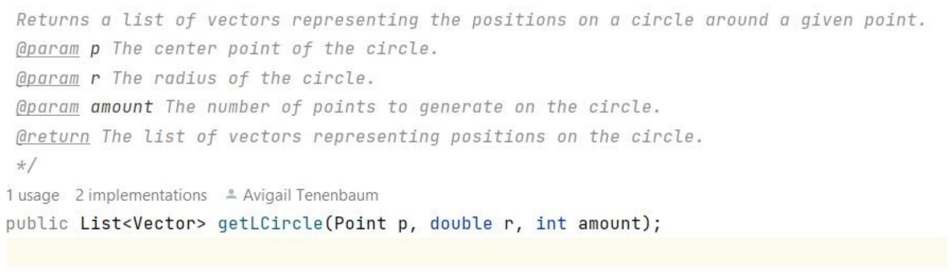
After:



To solve this , instead of casting one ray from the intersection point towards the light source to see if the point is shaded we cast n*m rays towards a "radius" around the light source , we calculate the shade by combining the results of all the rays , the closer the rays are in direction to the ray from the intersection point to the light source all the rays will intersect the shadowing object and the point will be fully shaded, but further away from the center of the shadowing object towards the edges , some of the rays will miss the shadowing object adding some light to the color of the point , after adding all the ray colors we will get a lighter shade giving us the fading affect as we wanted.

We implemented this in our code by adding a function in the light sources classes:

credit to Dina)@)



```java
Returns a list of vectors representing the positions on a circle around a given point.
@param p The center point of the circle.
@param r The radius of the circle.
@param amount The number of points to generate on the circle.
@return The list of vectors representing positions on the circle.
*/
1 usage   2 implementations   Avigail Tenenbaum
public List<Vector> getLCircle(Point p, double r, int amount);
```

Results:

Finally, we present our full picture using the improvements

Additional credits:

To Eliezer Gainsburger for all the help.

to chatgpt for the documentation.

To the folks at GitHub for helping with the tests