

Introduction to Software Engineering

Mini Project – Report

Spring 2023

Introduction :

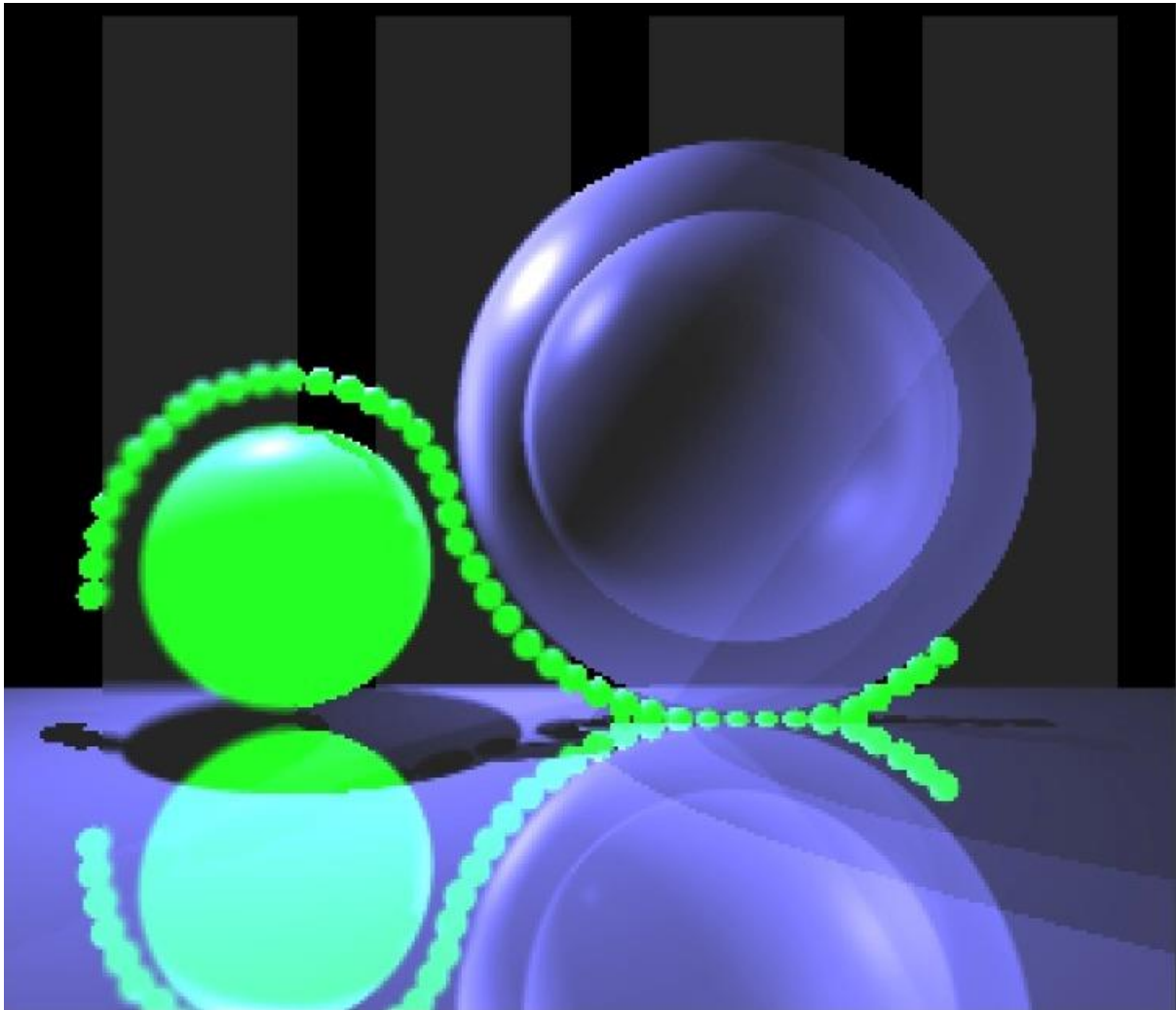
In this report, we aim to showcase the methodologies employed in crafting our picture while highlighting the enhancements made to our project. The task at hand was to construct a visually captivating scene comprising various geometries, along with the incorporation of multiple light sources to emphasize shading effects throughout the composition.

Our approach to creating the picture involved the seamless integration of a diverse functionalities. This entailed the skillful utilization of reflective surfaces, exemplified by the floor, which adds a captivating visual element. Additionally, transparent surfaces, represented by the spheres, were strategically employed to introduce an intriguing interplay of light and transparency within the scene.

Furthermore, we paid careful attention to the positioning and configuration of the light sources. By incorporating multiple light sources, we achieved the desired effect of cross shadows, enhancing the overall realism and depth of the scene. These shadows, cast by the interaction of light with the various geometries, contribute to the immersive and visually engaging nature of the final picture.

In summary, our diligent thought process, meticulous integration of functionalities, and judicious use of reflective and transparent surfaces, alongside the strategic placement of light sources, have culminated in the creation of a scene that showcases the full potential and advancements of our ray tracing project.

Our picture:



In the scene, there are several spheres with different positions, radii, and materials. The spheres are used to create a chain-like structure. The spheres have red emission color and are made of a material with diffuse and specular coefficients set to 0.8, shininess of 30.

The scene also includes two larger spheres with black emission color, overlapping each other. These spheres have different radii and are made of a material similar to the chain spheres.

The scene has a black background and ambient light with a white color and intensity of 0.15.

The camera is positioned at coordinates (0, 0, -1000) and is looking in the positive z-direction. The view plane size is set to 1000x1000, and the view plane distance is set to 1000.

That means that when we look at the picture, the spheres are behind the polygons.

The floor surface is reflective, creating reflections of the spheres. Additionally, the spheres cast shadows resulting from a light source positioned in the northwest direction.

The largest sphere in the scene features transparency, adding an extra dimension to the composition.

Bonus

We added a **rotate camera** feature to our project, allowing users to change the camera's orientation. It enables them to explore the virtual environment from different angles, enhancing immersion and interaction. By rotating the camera, users gain a fresh perspective, inspect objects, and examine details from various viewpoints. This feature adds flexibility, control, and engagement to the user experience, making it valuable for gaming, architectural visualization, and other interactive applications.

```
* Rotates the camera around the axes with the given angles  
*  
* @param x angles to rotate around the x axis  
* @param y angles to rotate around the y axis  
* @param z angles to rotate around the z axis  
* @return the current camera  
*/  
1 usage  👤 Tehila Benezra  
public Camera rotate(double x, double y, double z) {  
    vTo = vTo.rotateX(x).rotateY(y).rotateZ(z);  
    vUp = vUp.rotateX(x).rotateY(y).rotateZ(z);  
    vRight = vTo.crossProduct(vUp);  
  
    return this;  
}
```

Improvements:

Now, after the scene construction is complete, let's examine the improvement we added to our project.

Anti-Aliasing:

If we look carefully at the scene, it can be observed that the edges of the image are too sharp and create a sliced and unrealistic appearance to the eye.

This occurs because we sample the center of the pixels to determine their color and then assign the same color to the entire pixel. However, this assumption doesn't hold true in all cases. The edges of the spheres might only partially cover a pixel, yet the entire pixel is inaccurately colored with the sphere's color. As a result, the boundaries between different elements in the scene appear abrupt and less visually convincing.

To solve this issue, we added Anti-Aliasing functionality to our camera object:

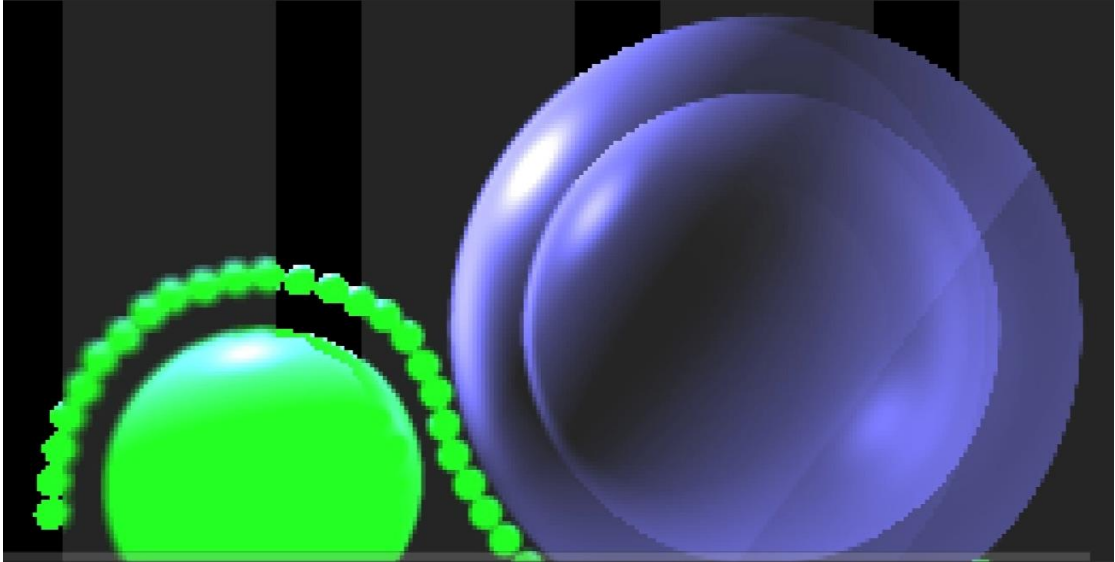
```
/**
 * Renders the image using the configured camera parameters.
 *
 * @return The camera instance.
 * @throws MissingResourceException if any camera data is missing.
 */
20 usages  Tehila Benezra
public Camera renderImage() {
    // Check if any camera data is missing
    if (p0 == null || vRight == null
        || vUp == null || vTo == null || distance == 0
        || width == 0 || height == 0 || centerPoint == null
        || imageWriter == null || rayTracer == null) {
        throw new MissingResourceException("Missing camera data", Camera.class.getName(), null);
    }
    Pixel.initialize(imageWriter.getNy(), imageWriter.getNx(), interval: 1);

    if (!adaptive) {
        // Render image using multi-threading and regular anti-aliasing
        while (threadsCount-- > 0) {
            new Thread() -> {
                for (Pixel pixel = new Pixel(); pixel.nextPixel(); Pixel.pixelDone())
                    imageWriter.writePixel(pixel.col, pixel.row, rayTracer.
                        TraceRays(constructRays(imageWriter.getNx(),
                            imageWriter.getNy(), pixel.col, pixel.row,
                            antiAliasing)));
            }.start();
        }
        Pixel.waitForFinish();
    } else {
        // Render image using multi-threading and adaptive super-sampling
        while (threadsCount-- > 0) {
            new Thread() -> {
                for (Pixel pixel = new Pixel(); pixel.nextPixel(); Pixel.pixelDone())
                    imageWriter.writePixel(pixel.col, pixel.row, AdaptiveSuperSampling(imageWriter.getNx(),
                        imageWriter.getNy(), pixel.col, pixel.row, antiAliasing));
            }.start();
        }
        Pixel.waitForFinish();
    }
    return this;
}
```

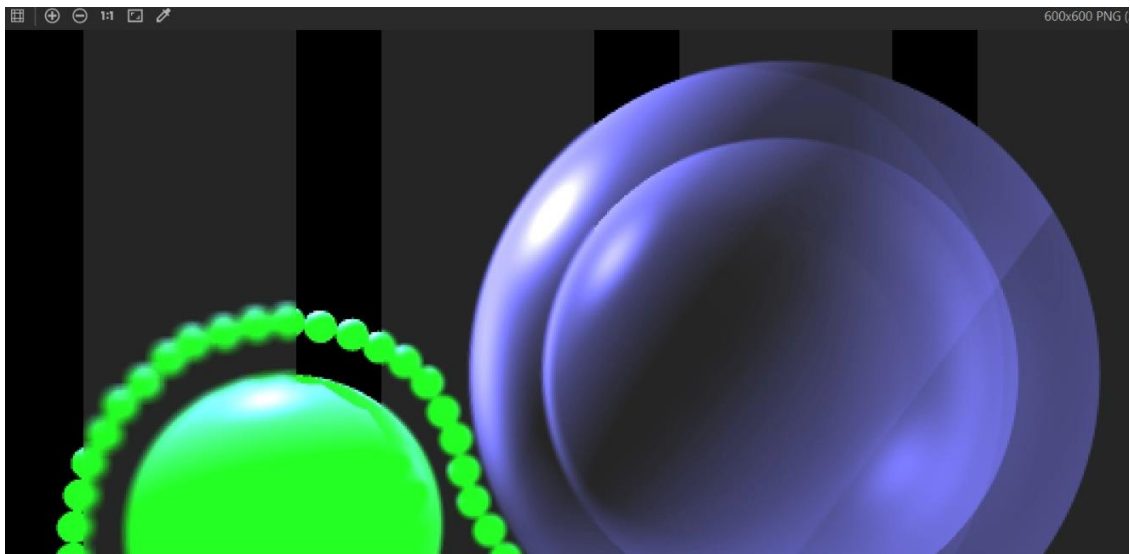
Tehila Ben-Ezra
Esther Malka Nusbacher

If we enlarge both the photos and focus on a small area, we can see the "mixed color" effect on the pixels.

No Anti-Aliasing:



With Anti-Aliasing:



Optimization:

Adaptive Super-sampling:

We optimized our image improvements by addressing the problem of excessive ray tracing per pixel. To minimize unnecessary calculations, we introduced a technique where we sample the color of pixel edges. If the color is consistent across all edges, we avoid further recursion and return that color. Only when colors differ, do we proceed with the recursive process on sub-pixels. In the end, we return the average color from the four sub-pixels. These code changes significantly reduce computational overhead and enhance the efficiency of our ray tracing process.

```
/**
 * This method performs adaptive super sampling to determine the color of a pixel.
 * It uses individual rays to check the color and averages between them.
 * If necessary, it continues to send beams of rays in recursion.
 *
 * @param nX      Pixel length
 * @param nY      Pixel width
 * @param j       The position of the pixel relative to the y-axis
 * @param i       The position of the pixel relative to the x-axis
 * @param numOfRays The amount of rays sent
 * @return        The color of the pixel
 */
private Color AdaptiveSuperSampling(int nX, int nY, int j, int i, int numOfRays) {
    Vector Vright = vRight;
    Vector Vup = vUp;
    Point cameraLoc = this.getP0();
    int numOfRaysInRowCol = (int) Math.floor(Math.sqrt(numOfRays));

    // If only one ray is used, directly trace the ray through the pixel
    if (numOfRaysInRowCol == 1) {
        return rayTracer.TraceRay(constructRayThroughPixel(nX, nY, j, i));
    }

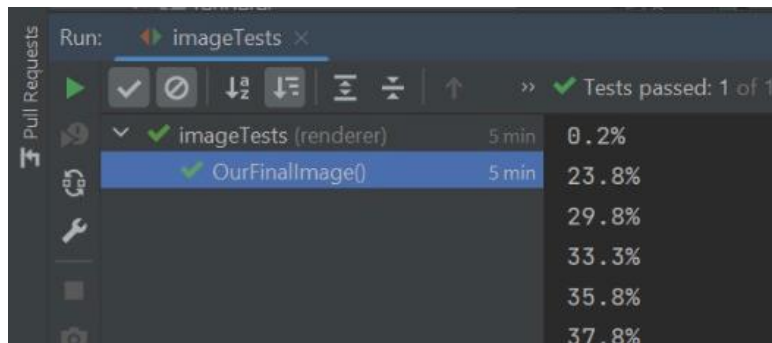
    Point pIJ = getCenterOfPixel(nX, nY, j, i);

    // Calculate the ratios of pixel width and height
    double rY = alignZero(number: height / nY);
    double rX = alignZero(number: width / nX);

    double PRy = rY / numOfRaysInRowCol;
    double PRx = rX / numOfRaysInRowCol;

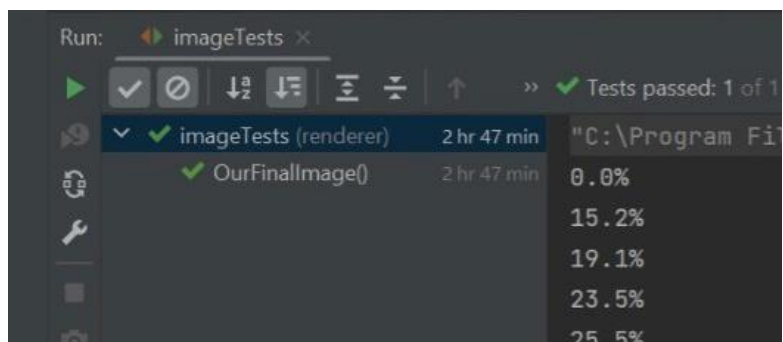
    // Perform recursive adaptive super sampling
    return rayTracer.AdaptiveSuperSamplingRec(pIJ, rX, rY, PRx, PRy, cameraLoc, Vright, Vup, prePoints: null);
}
```

With Adaptive Super-sampling:



Run: imageTests x		
✓	imageTests (renderer)	5 min 0.2%
✓	OurFinalImage()	5 min 23.8%
		29.8%
		33.3%
		35.8%
		37.8%

Without Adaptive Super-sampling:



Run: imageTests x		
✓	imageTests (renderer)	2 hr 47 min "C:\Program Fi"
✓	OurFinalImage()	2 hr 47 min 0.0%
		15.2%
		19.1%
		23.5%
		25.5%

Conclusion:

In conclusion of the project, we have successfully implemented a multitude of principles learned during the introductory course, including essential concepts such as DRY (Don't Repeat Yourself) and KISS (Keep It Simple, Stupid), among others. We dedicated a substantial amount of time and thoughtful consideration to the project, which resulted in the acquisition of new knowledge and, most importantly, a tremendous sense of satisfaction. By adhering to these principles, we have elevated the professionalism of our work and achieved a higher level of proficiency in our project execution.