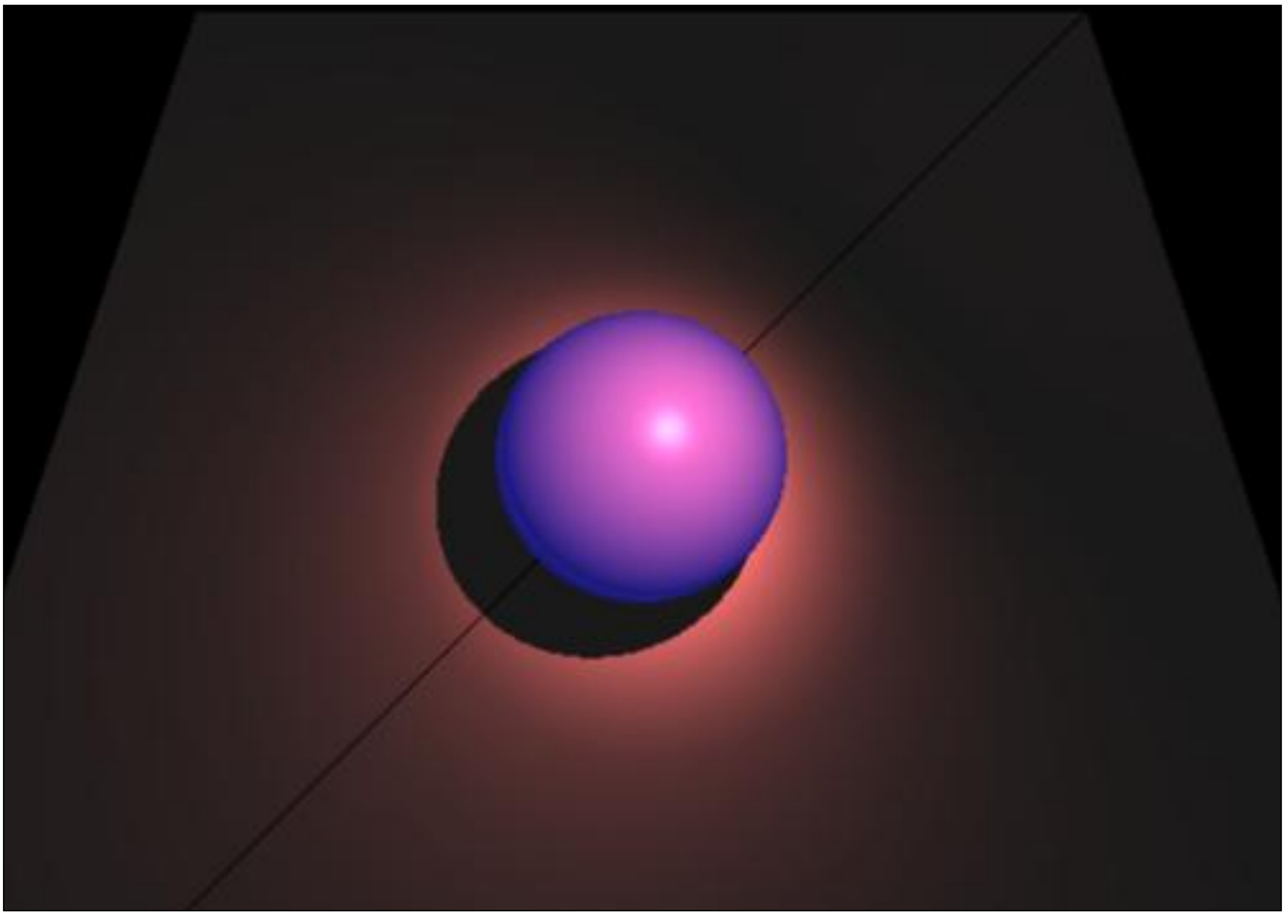
Improving Ray Tracing

Improving ray tracing render quality with grid supersampling

Ephraim Hammer
ID: 900445

7 August 2017



Introduction

As we have seen in class the ray tracing program that we created was far from perfect, the more we zoom on the picture the more we see staircases-like edges, this is due to a digital processing of data points coupled with a limit in memory availability as seen in class.

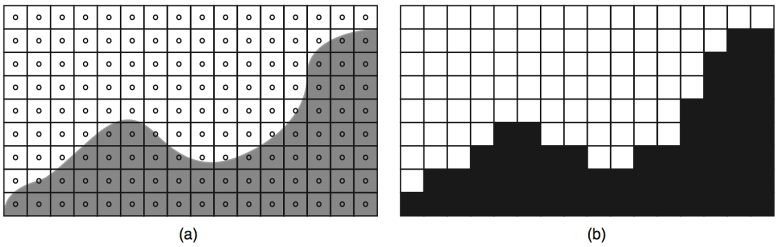
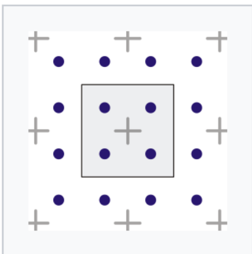


Figure 1: Illustration of jaggies caused by sampling
(a) Infinitely detailed curve
(b) Jagged sampled representation

So as to improve render quality I choose to implement a SuperSampling algorithm, more specifically a **Grid SuperSampling** algorithm.



Grid algorithm in uniform distribution

Grid Supersampling algorithm implementation

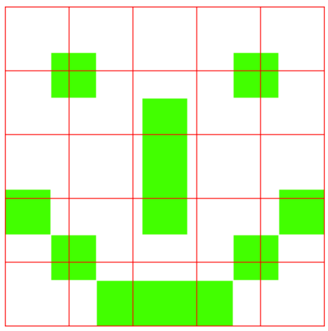
The method implemented by me is very simple it consists in taking into account multiple rays for each pixel and then averaging the result. For a very simple version think of it like this:

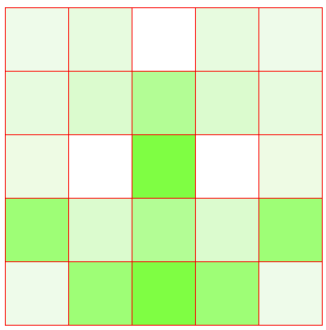
- We render first an image of size 1024x1024, one ray for each pixel (for example)
- After rendering, we scale the image to 512x512 (each 4 pixels are averaged into one) and you can notice that the edges are smoother. This way you have effectively used 4 rays for each pixel in the final image of 512x512 size.

Pixels averaging

To average pixels we take the supersampling pixels and divide the sum of their colours by the square of our chosen grid size.

- **Example:**

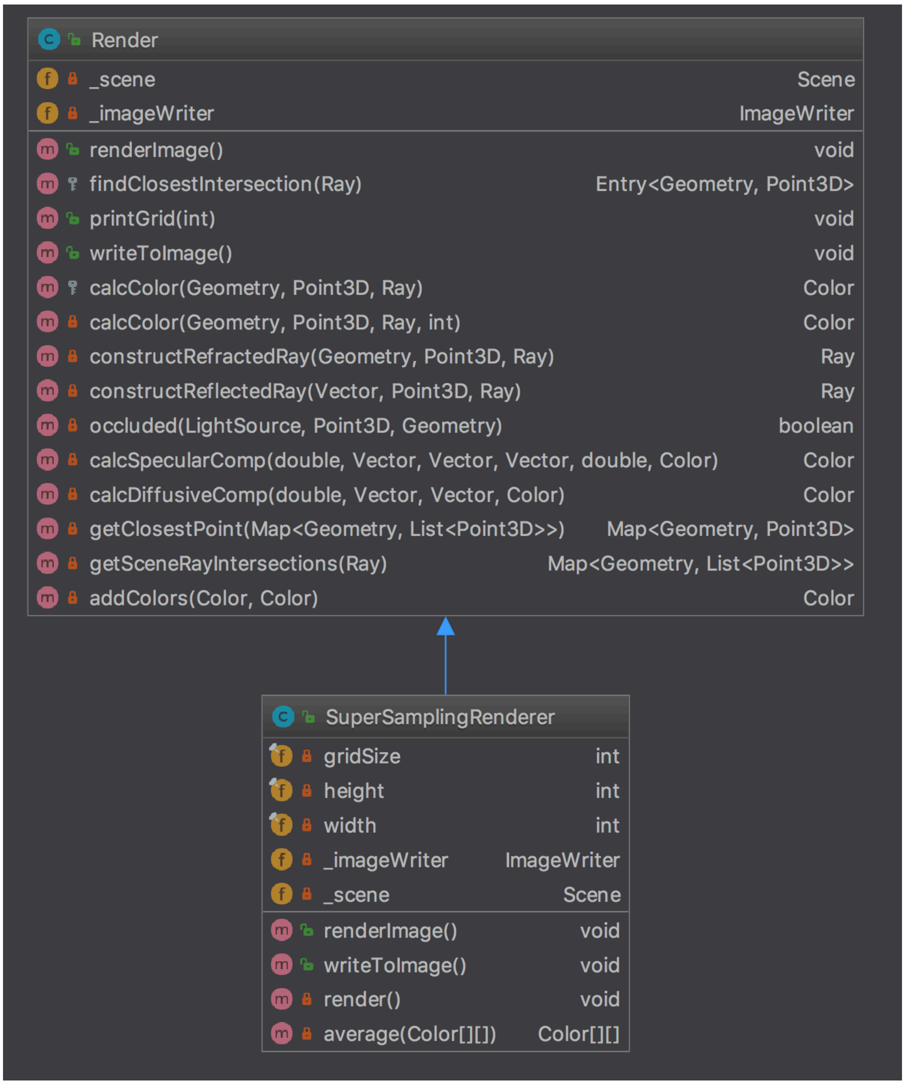




Averaged pixels result

Code

I cerated a *SuperSamplingRenderer Class* that extended the *Renderer Class.*



In this class I overrode the *renderImage()* method so that when called it calls the *render()* method.

The *render()* method. first multiplies the size of the image by the chosen grid size and than traces all the rays trough each pixel and computes for each his colour.

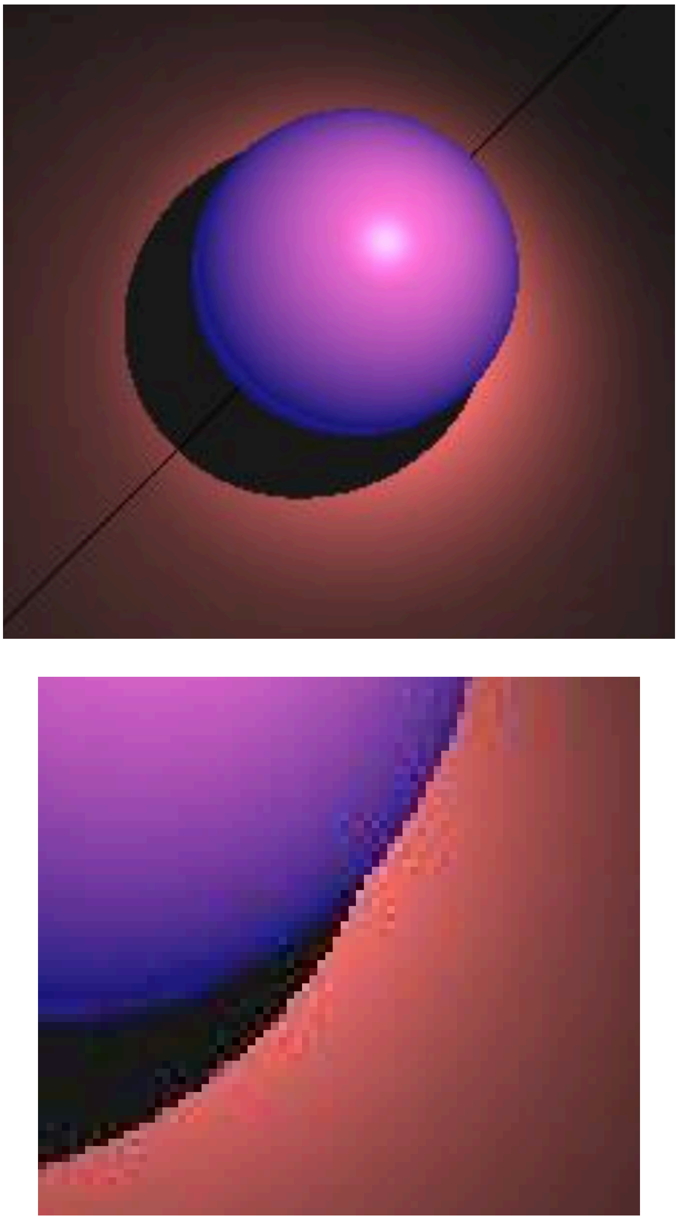
```
private void render()
{
    // initialize the array of blocks of rays
    final Ray[][] rays = new Ray[this.width * this.gridSize][this.height * this.gridSize];

    // Create an array of generated  colors
    Color[][] superPixelColors = new Color [this.width*this.gridSize][this.height*this.gridSize];

    for (int y = 0; y < this.height*this.gridSize; ++y) {

        for (int x = 0; x < this.width*this.gridSize; ++x) {

                    rays[x][y] = _scene.get_camera().constructRayThroughPixel(
                            _imageWriter.getNx(),
                            _imageWriter.getNy(),
                            x,
                            y,
                            _scene.get_screenDistance(),
                            width,
                            height
                    );

                    Map.Entry<Geometry, Point3D> entry = findClosestIntersection(rays[x][y]);

                    if (entry == null){
                        superPixelColors[x][y] =  _scene.get_background();
                    } else {
                        superPixelColors[x][y]= calcColor(
                                entry.getKey(),
                                entry.getValue(),
                                rays[x][y]);
            }
        }
    }

    Color [][] colors;
    colors = average(superPixelColors);

    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            _imageWriter.writePixel(j, i, colors[j][i]);
        }
    }
}
```

It then call the *average()* that returns an array with the original image size and the averaged pixels.

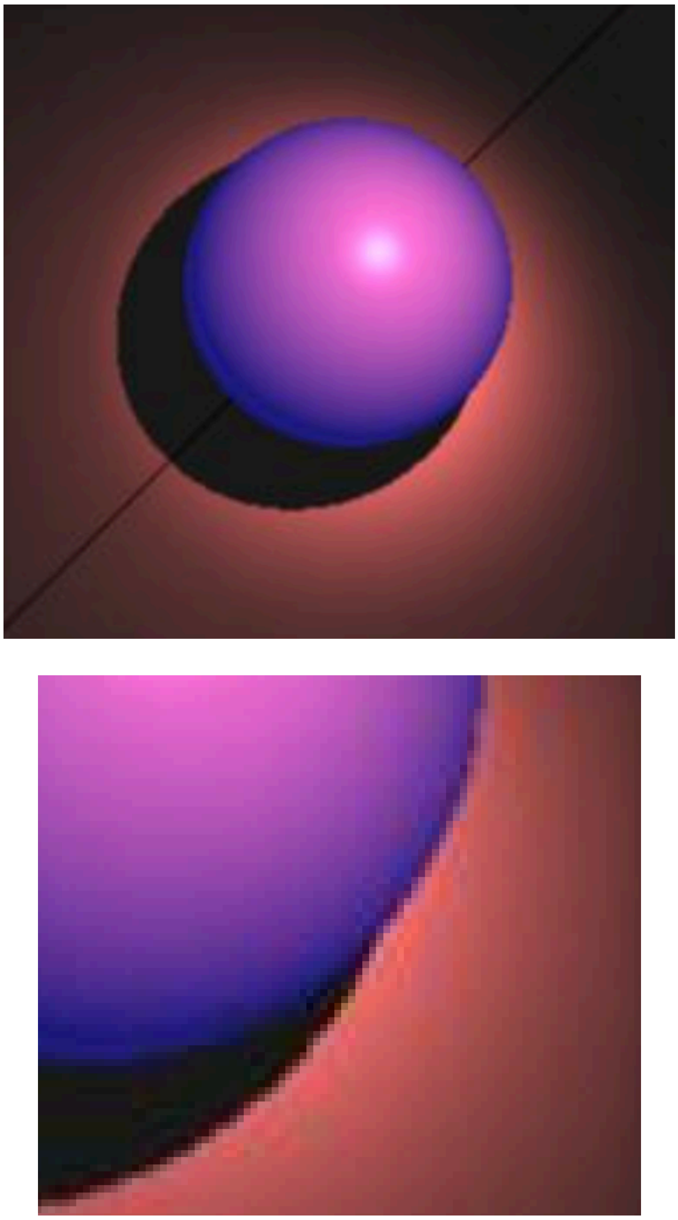
```java
private Color[][] average(final Color[][] pixels)
{
    final int gridSize = this.gridSize;
    final int gridSizeSquared = gridSize * gridSize;

    final Color[][] result = new Color[width][height];

    for (int i = 0; i < height;i++) {
        for (int j = 0; j < width; j++) {

            int sumRed = 0;
            int sumGreen = 0;
            int sumBlue = 0;

            // inner loops for superpixel
            for (int k = i; k < gridSize+i; k++) {
                for (int l = j; l < gridSize+j; l++) {
                    sumRed += pixels[k][l].getRed();
                    sumGreen += pixels[k][l].getGreen();
                    sumBlue += pixels[k][l].getBlue();
                }
            }

            // Average
            sumRed = sumRed / gridSizeSquared;
            sumGreen  = sumGreen /  gridSizeSquared;
            sumBlue = sumBlue  / gridSizeSquared;

            result[i][j] = new Color(sumRed,sumGreen,sumBlue);

        }

    }
    return result;
}
```

Result

Images speak louder that words so I will illustrate my grid supersampling implementation with two comparative picture with and without a supersampling with a grid size of 2 meaning each pixels is multiplied by 4.



Without supersampling



With supersampling