Computer Graphics (CSCI-GA 2270-001) - Assignment 3

Avadesh Meduri (N10537558)

October 20, 2020

1 Common Information

The code was written in Ubuntu 18.04, with a cmake version 3.10.2 and C++ version 7.5.0. I have implemented all the algorithms in this assignment using the template functions provided in the github repo and have only used standard C++ libraries. I have commented the code so as to explain what I am trying to do in each block of the function. To create the GIF attached with the submission I use ffmpeg to merge the frames.

2 Answers to Exercises

1. Exercise 1
   (a) The following equation is used to scale the y direction of the grid : 2*scene.camera.focal_length*tan (scene.camera.field_of_view/2.0). Then x is scaled using the aspect ratio.
   (b) The implementation of the perspective camera is inside the render_scene function.
2. Exercise 2
   (a) The implementation of the shadow ray shooting and verification is inside the "is_light_visible function". The function template was modified to accept intersection variable hit (The point where the ray hits the object from which the shadow ray is to be cast) instead of the ray variable. I have written comments in the function to try to explain how the function works.
   (b) The figure generated when the epsilon offset is set to 0 for checking if a point falls within a shadow region is shown in Fig 1.
3. Exercise 3
   (a) The equation of the direction of the refracted ray is shown below

$$K = ri * D + (ri * (N.D) - \sqrt{1 - ri^2 * (1 - (N.D)^2)})N \qquad (1)$$

   where D is the direction of the incident ray, N is the normal at the point of intersection, ri is the refractive index and K is the direction of the refracted ray. The above equation can be derived

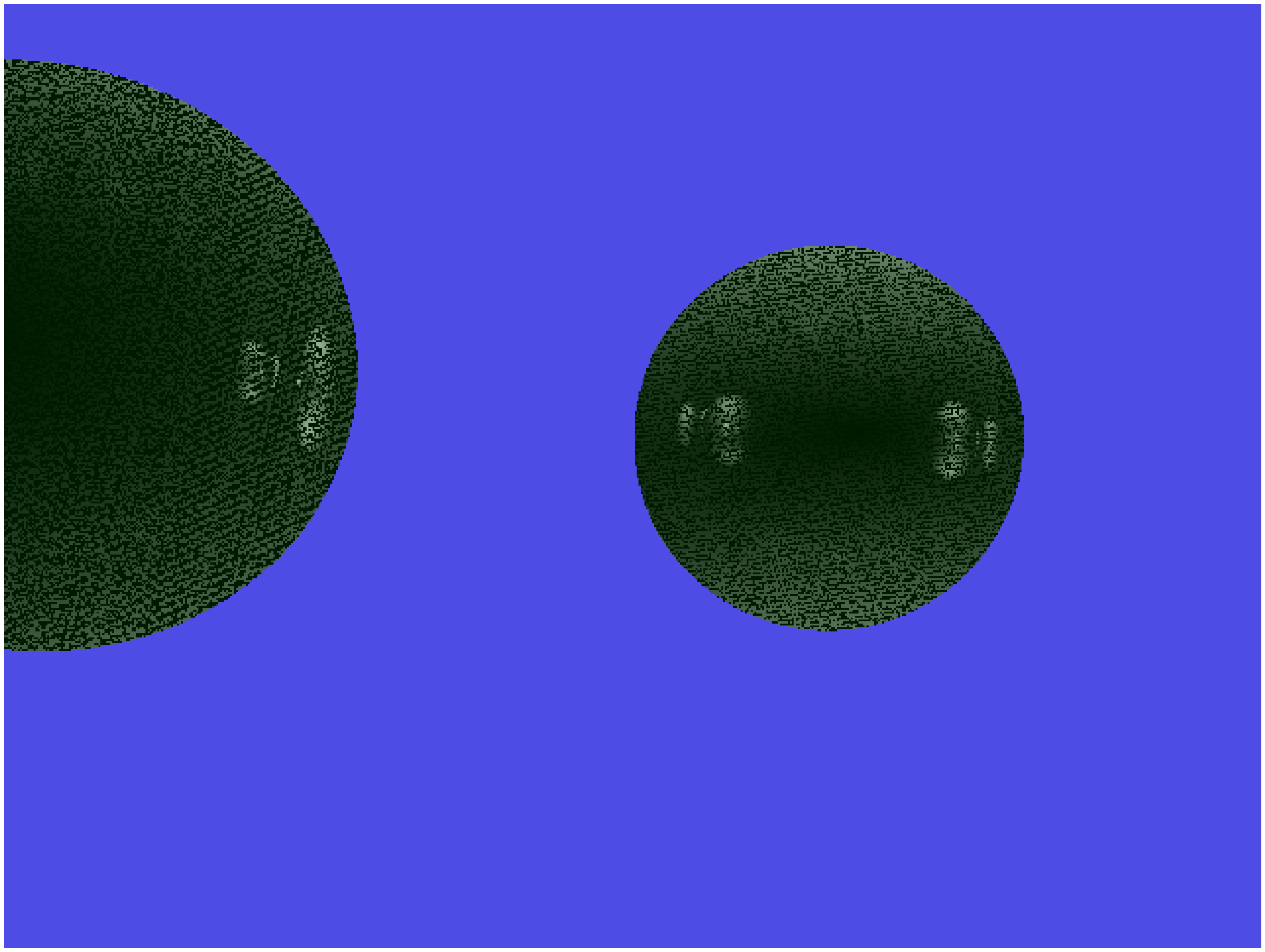

Figure 1: Figure generated with no epsilon for shading effect. Other effects like reflection and refraction are also implemented

as follows

$$\begin{aligned}
K &= sin(\theta_r)\hat{A} - cos(\theta_r)N \\
K &= sin(\theta_r)((D + Ncos(\theta_i))/sin(\theta_i)) - cos(\theta_r)N \\
K &= ri * (D + Ncos(\theta_i)) - cos(\theta_r)N \\
K &= ri * (D + N(N.D)) - (\sqrt{1 - ri^2 sin^2\theta_r})N \\
K &= ri * (D + N(N.D)) - (\sqrt{1 - ri^2(1 - cos^2\theta_i)})N \\
K &= ri * (D + N(N.D)) - (\sqrt{1 - ri^2(1 - (N.D)^2)})N
\end{aligned}$$

where $\hat{A}$ is the vector perpendicular to N and is given by $(D + Ncos(\theta_i))/sin(\theta_i)$. Here D and N are assumed to be unit vectors. And D is considered to be in the direction from the camera to the incident surface.

(b) The implementation of the reflection and refraction features is in the ray_color function. Fig 2 shows the rendered image with the two effects. The phong's constant had to be reduced from the value in the json to make the specular effects more prominent and match the desired image.

4. Exercise 4

   (a) The implementation of the depth of field effect is in the "render_scene function". The std::rand function is used to generate a float between the range of the camera lens radius. This is then used to alter the ray origin from the camera origin to create the depth of field effect. 5 rays are cast per pixel and the average is taken to generate the depth of field effect. Figure 3 shows the rendered image with depth of field.

   Increasing the number of rays per pixel increase the time of computation. At the same time, the objects away from the focal plane looked more out of focus with more rays. Similarly, when the lens radius increased, the images looked more blurry when away from the focal plane.

   (b) The rendered image when the right most sphere is moved near the focal plane to bring the sphere into focus is shown in figure 4

5. Exercise 5
   To create an animation, 10 images/frames were rendered by moving the right most sphere from the given position in the json towards the focal frame so that it comes into focus. After which, the sphere is brought back from the focal frame to the original location. The frames of the animation are shown in figure 5. The final gif formed by running these frames in a sequence is attached along with this submission.

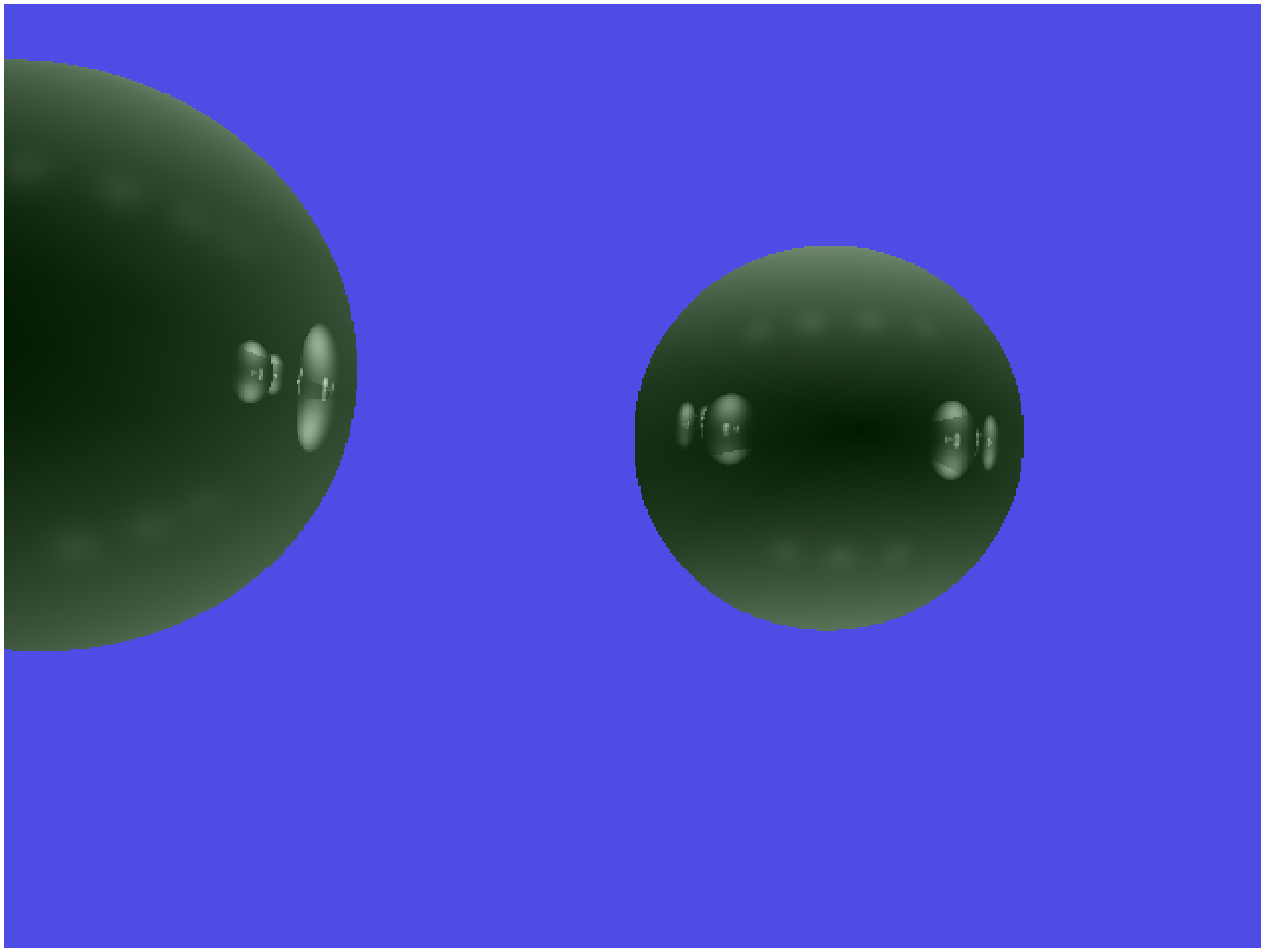

Figure 2: Figure generated with shading and refraction. The shadow effect is also active while rendering this image

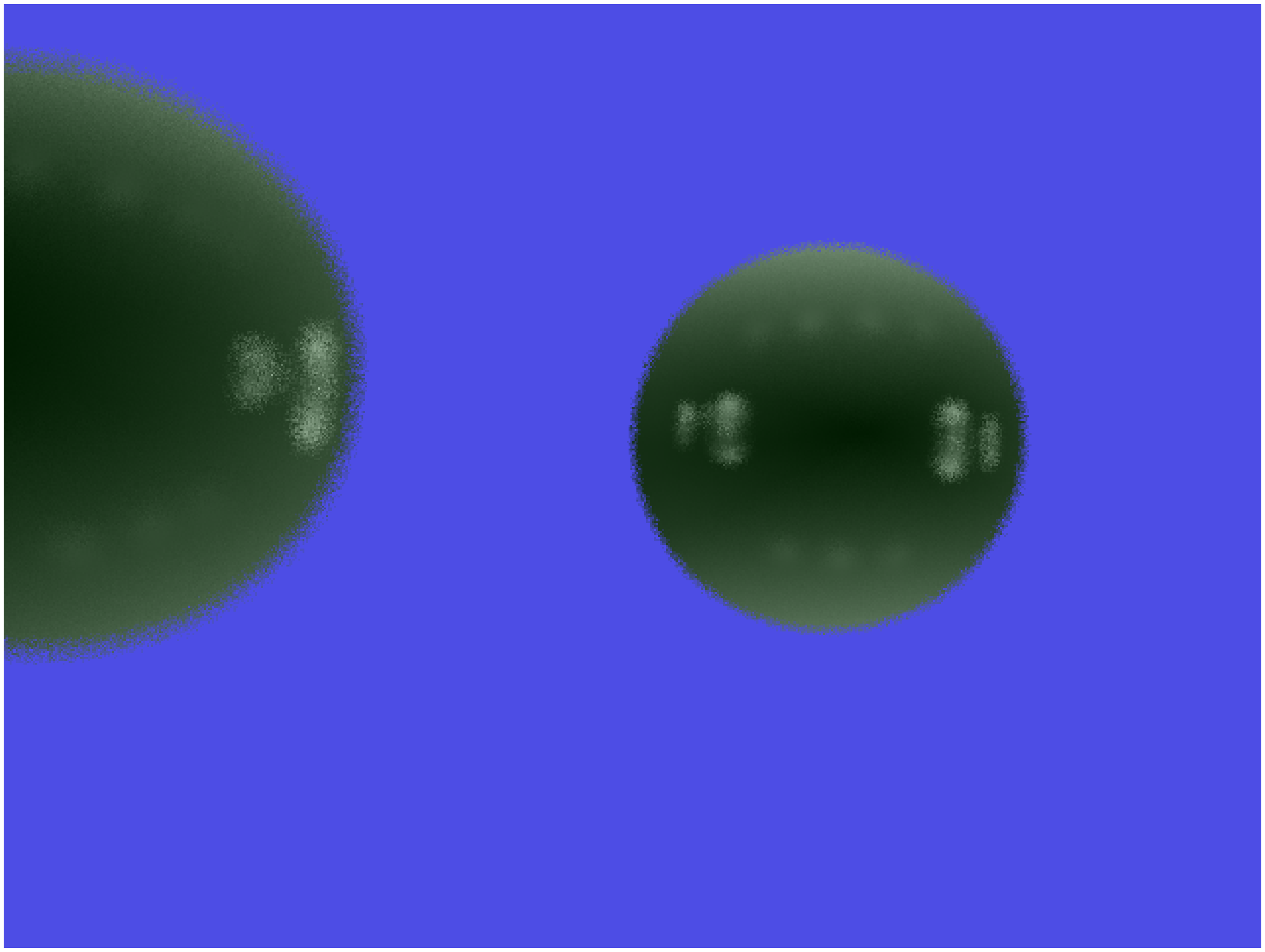

Figure 3: Figure generated with depth of field effect. Other effects are also implemented while rendering this image.

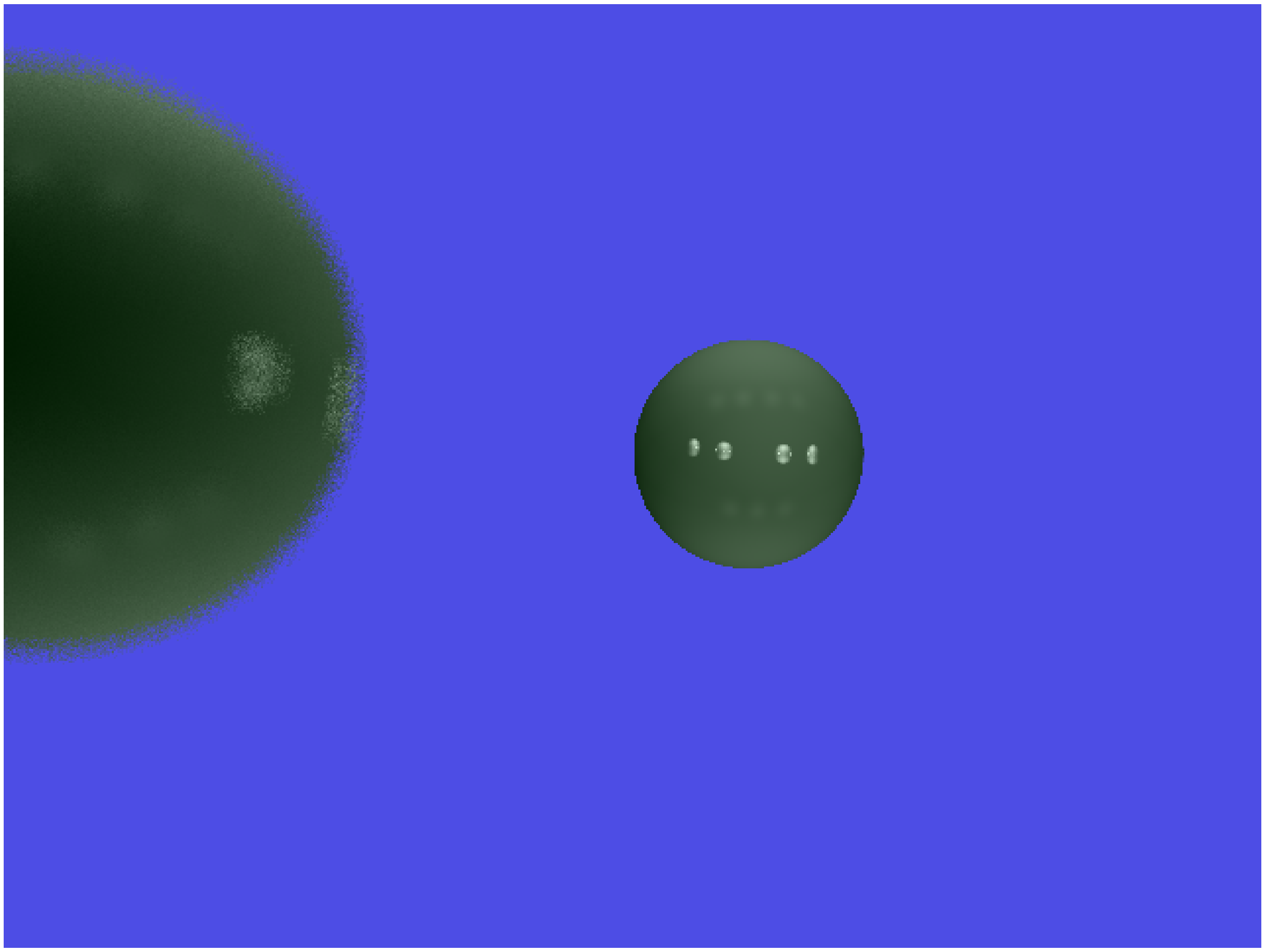

Figure 4: Rendered image with right sphere in focus with depth of field.

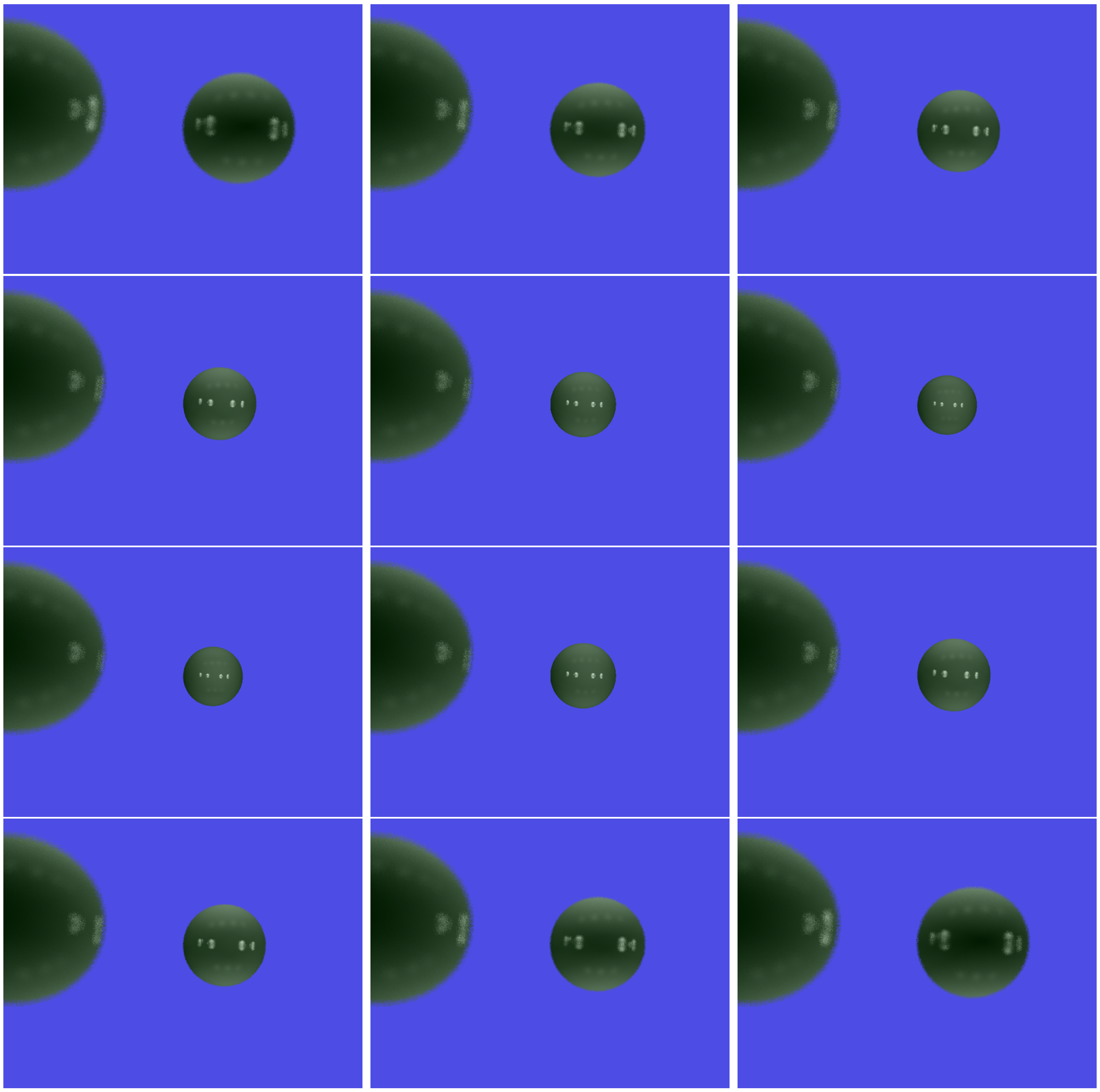

Figure 5: Rendered frames for creating a gif. The first frame is the top left followed by the one two its right. The last frame is the bottom right image