architracer

a render of a model of an architectural model

CS 488 A5 Demo Document
Zhehong (Jack) Zhou
20390479
z43zhou

Contents

Objective 1: Adaptive antialiasing (AAA)

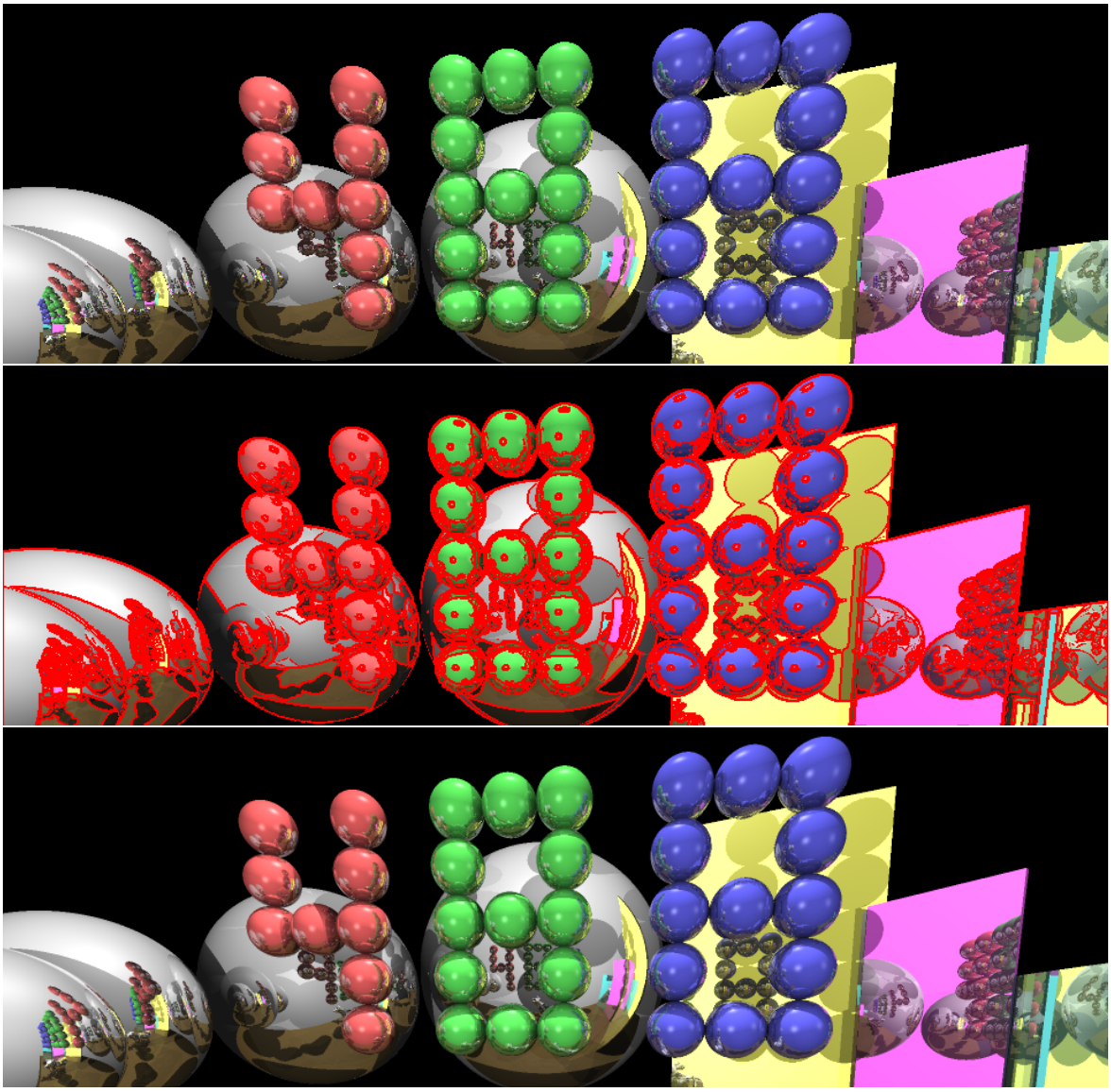

Figure 1: Edge detection using a threshold value on any of the colours, showing no antialiasing (top), edge detection (middle), and 2x AAA (bottom).

The Sobel-Feldman operator is used for edge detection. We perform adaptive antialiasing by only resampling the points where any one of the three colour transitions are over some threshold. This results in a massive speed-up with minimal visual trade-offs.

Table 1: Timing comparisons of AAA vs. raw SSAA for modified scene files on an i5-5257U with 2c/4t, with multithreading enabled. Times taken are single trials.

File	AA samples	AA type	Time taken (s)	Speed-up ratio
`nonhier.lua`	3x	AAA	1.995	2.79
		SSAA	5.557	1.00
`sample.lua`	2x	AAA	97.78	1.60
		SSAA	156.26	1.00

Objective 2: Reflections

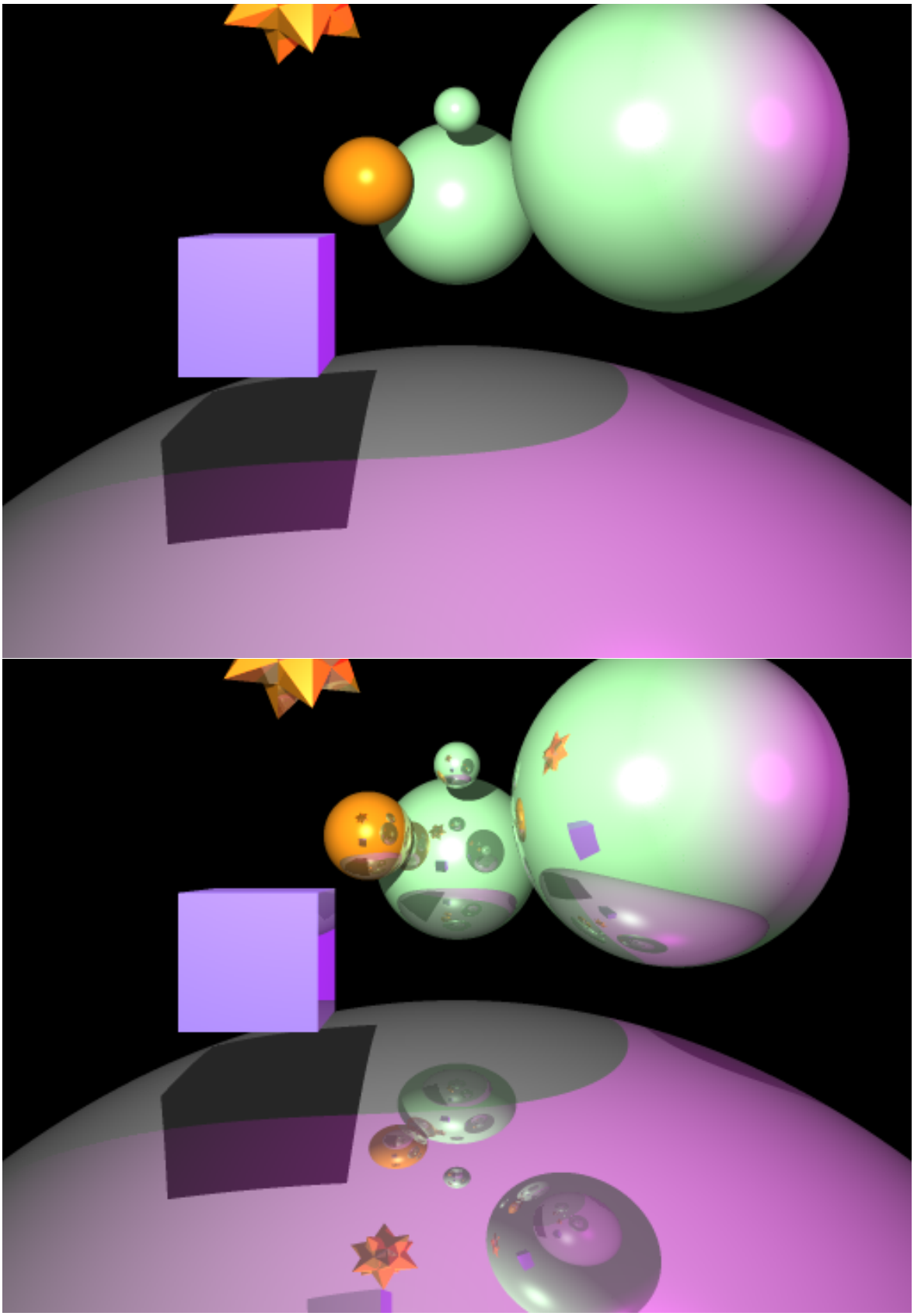

Figure 2: Comparison between renders with no reflections (top) and multiplicative reflections turned on, all materials using reflectivity 0.6 (bottom).

Objective 3: Refractions

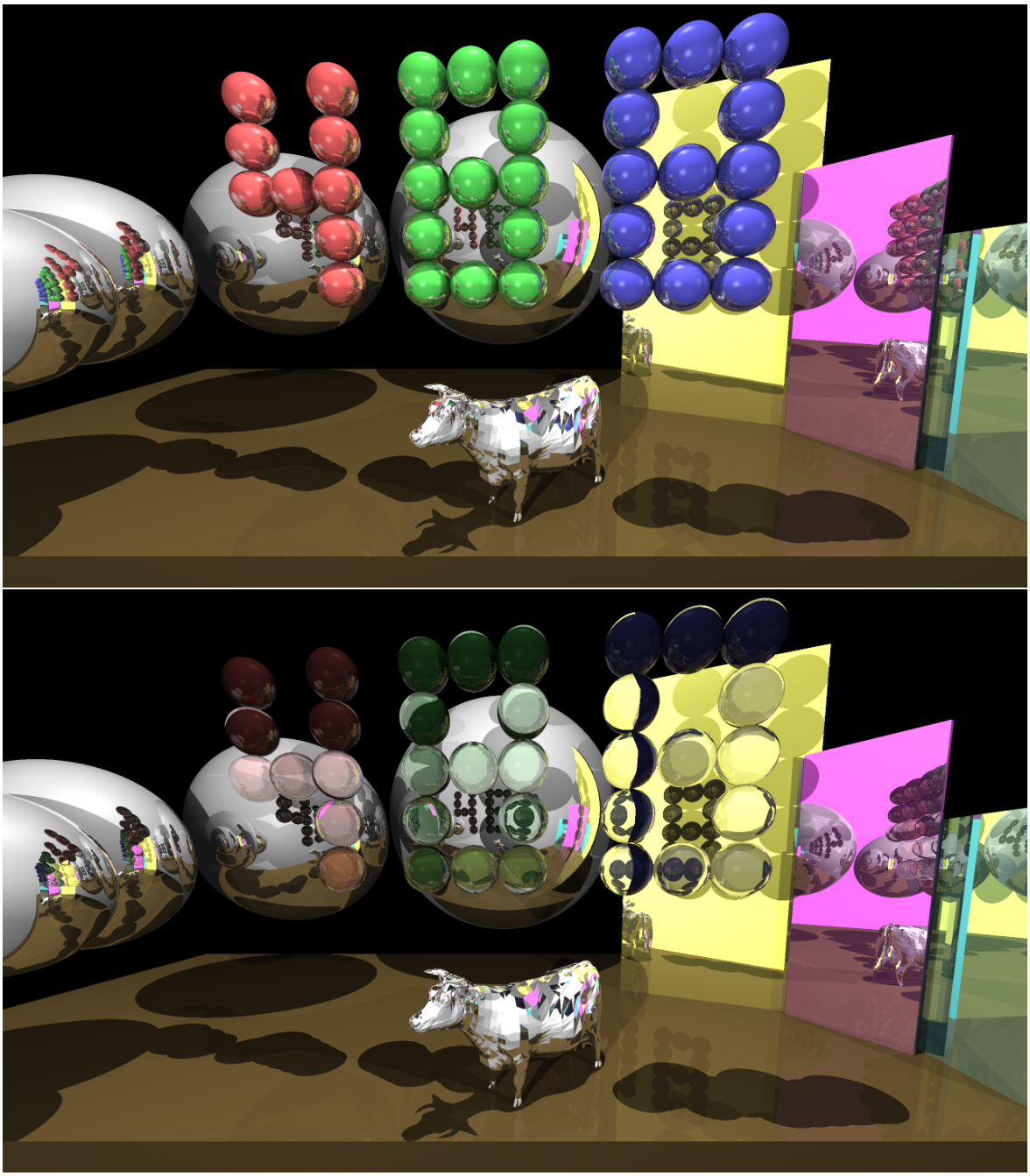

Figure 3: Comparison between renders with no refractions (top) and multiplicative refractions turned on for the spheres comprising the "488", using refractivity 0.9 and refractive index 1.2 (bottom).

Objective 4: Phong shading

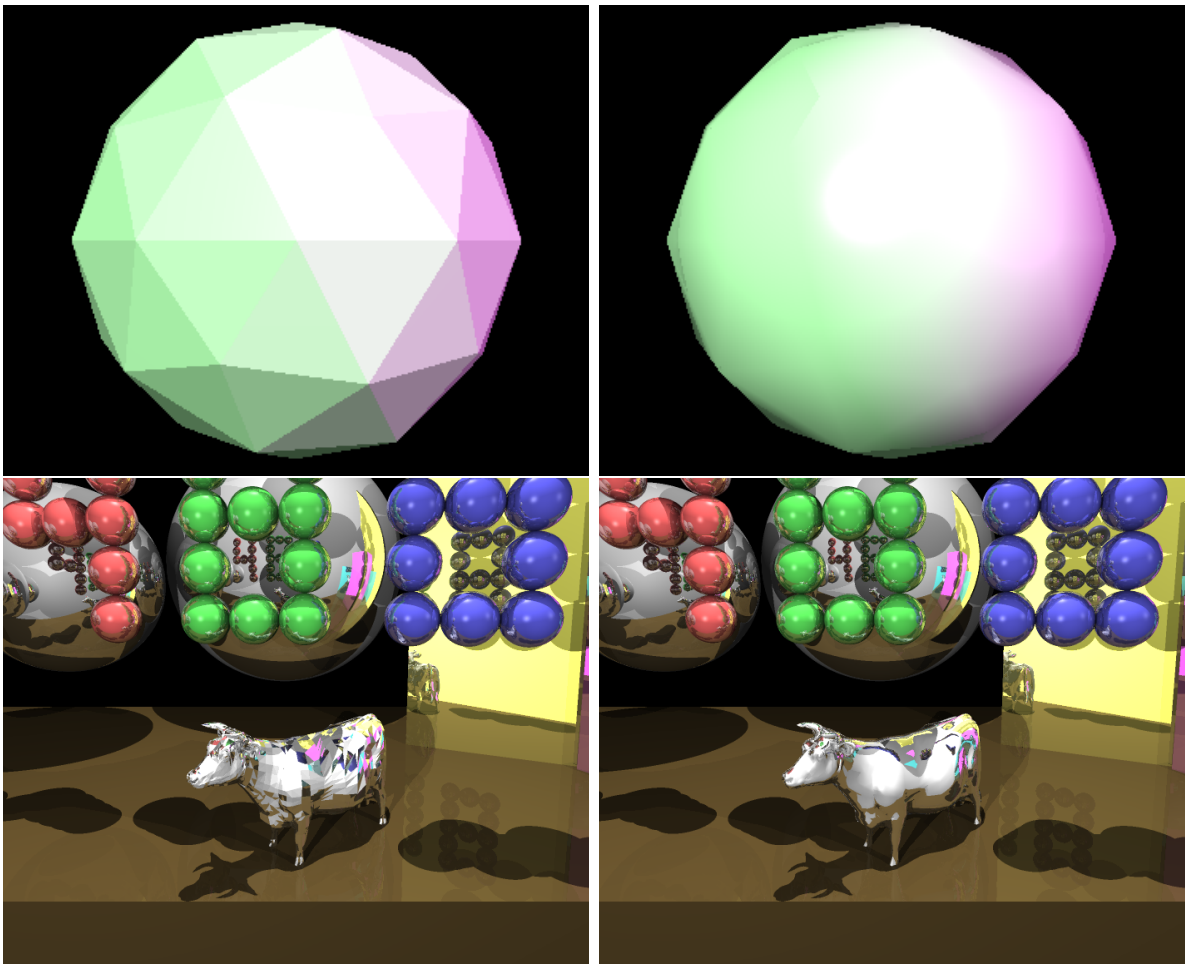

Figure 4: Comparison between renders using face-wise triangle normals (left two) and with Phong shading (right two).

Phong shading is implemented by performing Barycentric interpolation of the vertex normals at the collision point. Note that this requires the modelling software to export the `obj` file using smooth normals rather than face normals as the vertex normals. If faces are specified without normals, regular triangle normals are used.

Objective 5: Primitives

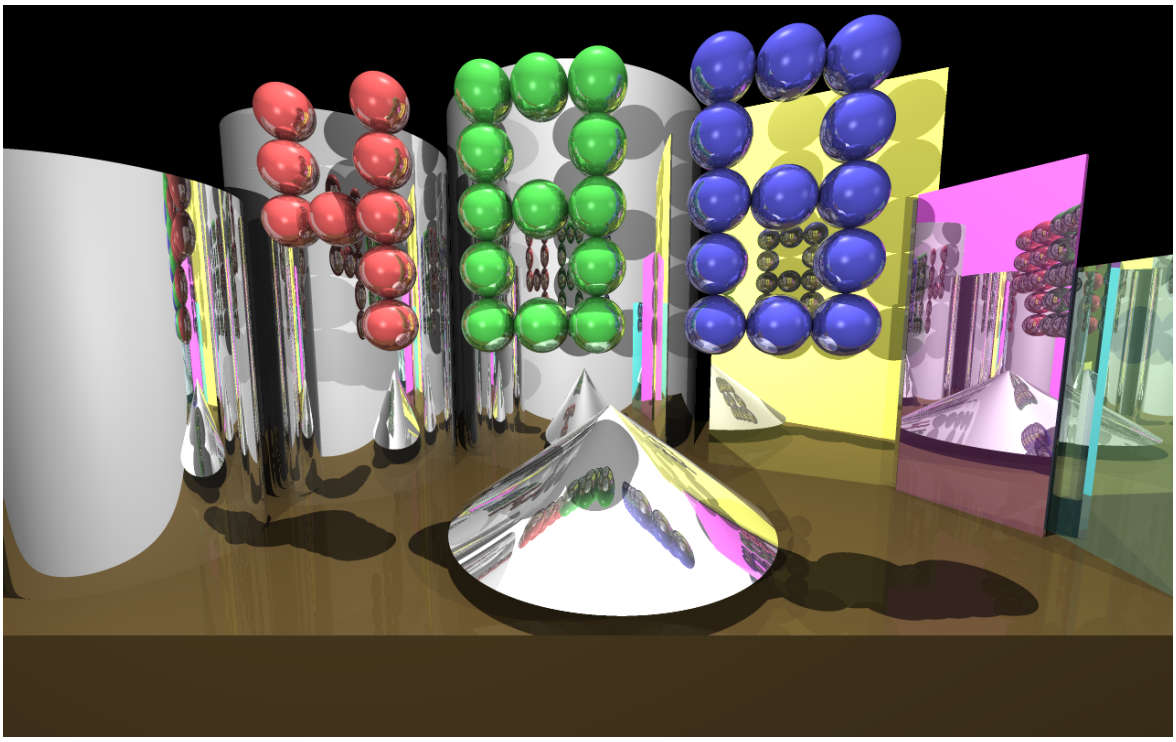

Figure 5: Sample render modified to use cylinder and cone primitives.

Objective 6: Texture mapping

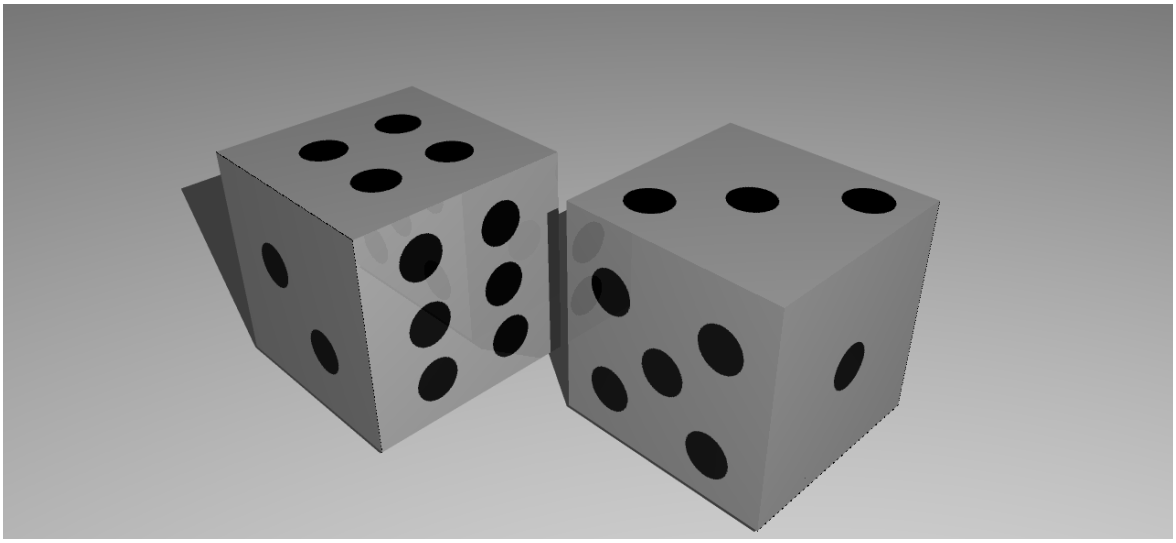

Figure 6: Texture-mapped dice with 0.1 reflectivity.

Texture maps are done using vertex texture values included in the `obj` file. They are generated by Blender using its UV-mapping system, which allows for precisely-mapped textures.

Objective 7: Bump mapping

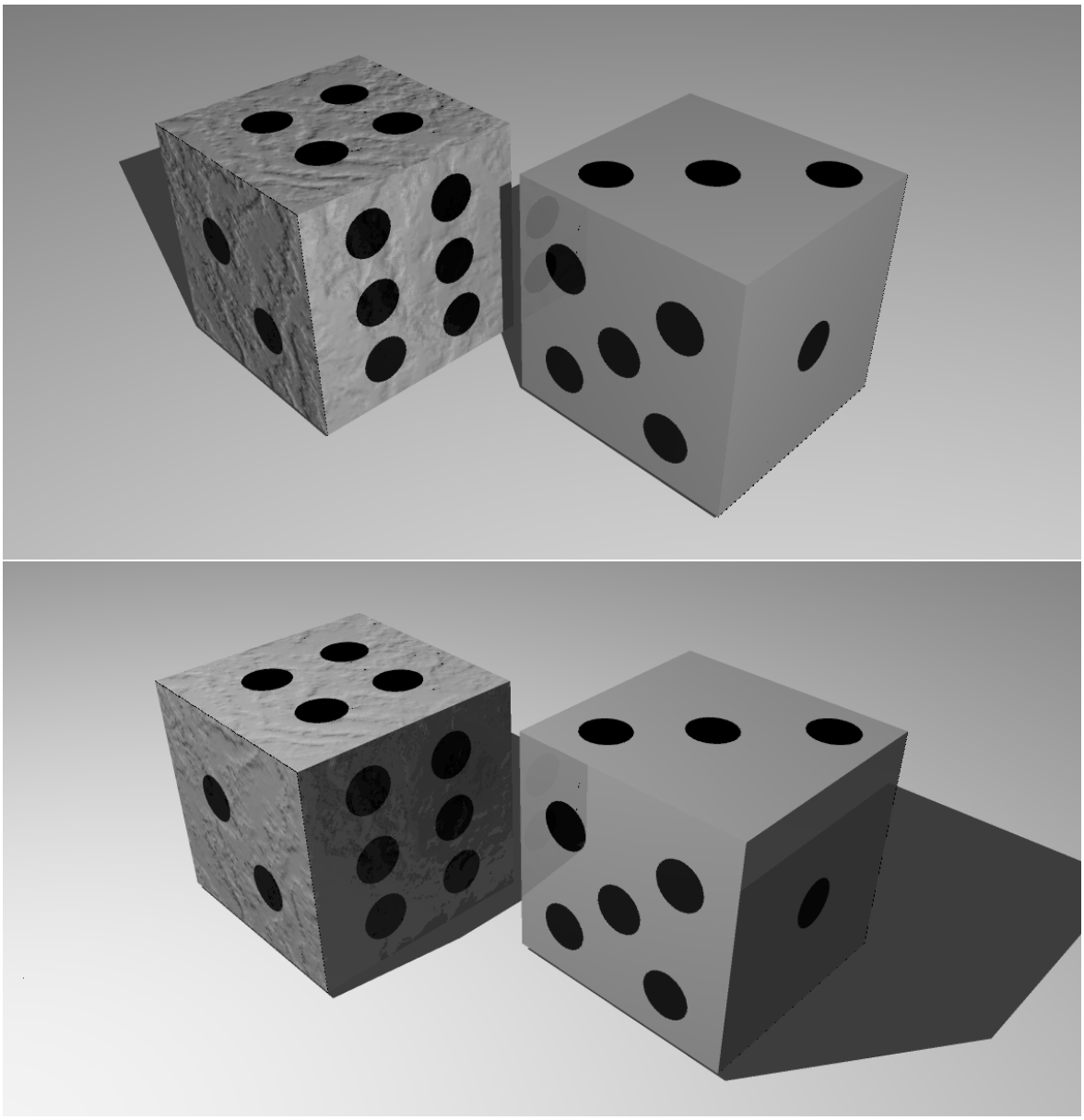

Figure 7: Bump map applied to the left die with lighting in two locations to show texture.

Bump mapping is applied using a height map. We use some pre-defined delta value (in this case, 0.001) and compute a changed normal vector. We then calculate the transformation between the original normal and the changed normal and apply it to the normal of the collision.

Objective 8 and 9: Photon mapping

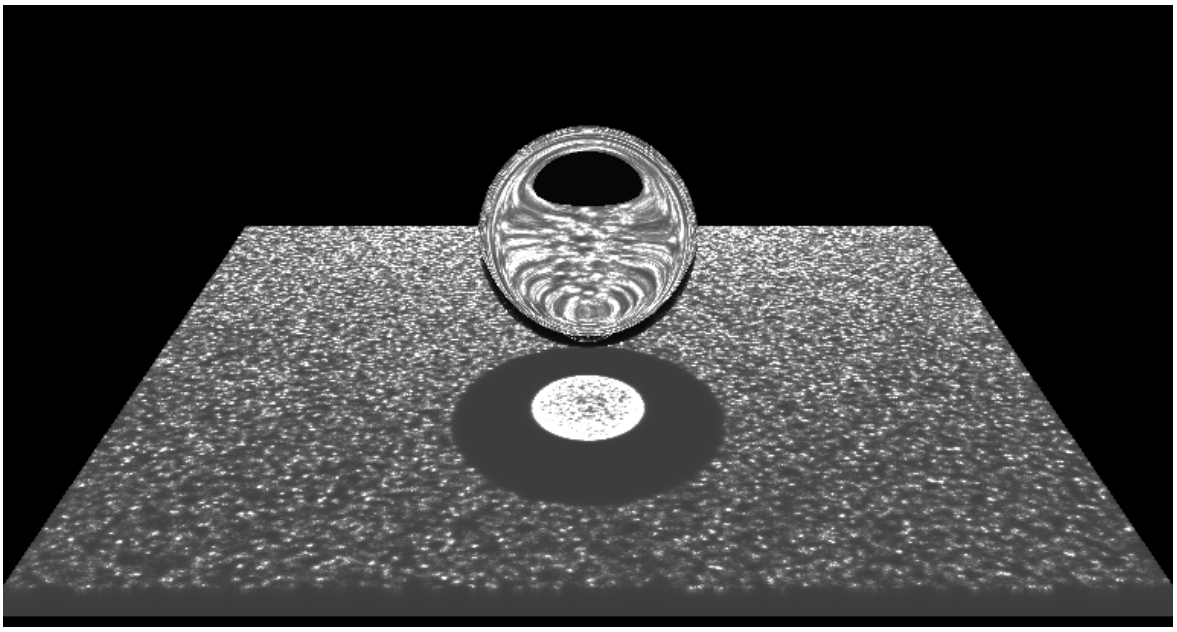

Figure 8: Photon mapping with $k = 1$ nearest neighbour at 100000 photons and modified lighting to show photon casting and caustics. Note that the final version does not allow photons that did not go through another surface to be absorbed.

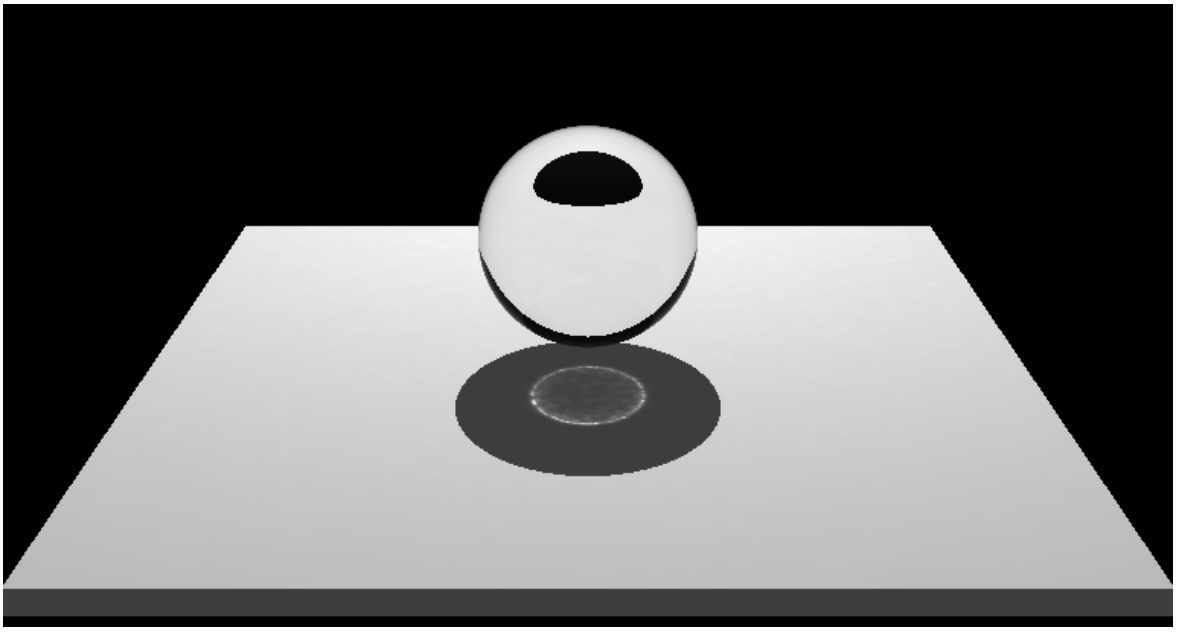

Figure 9: Photon mapping with $k = 30$ nearest neighbours at 100000 photons and standard lighting to show photon gathering.

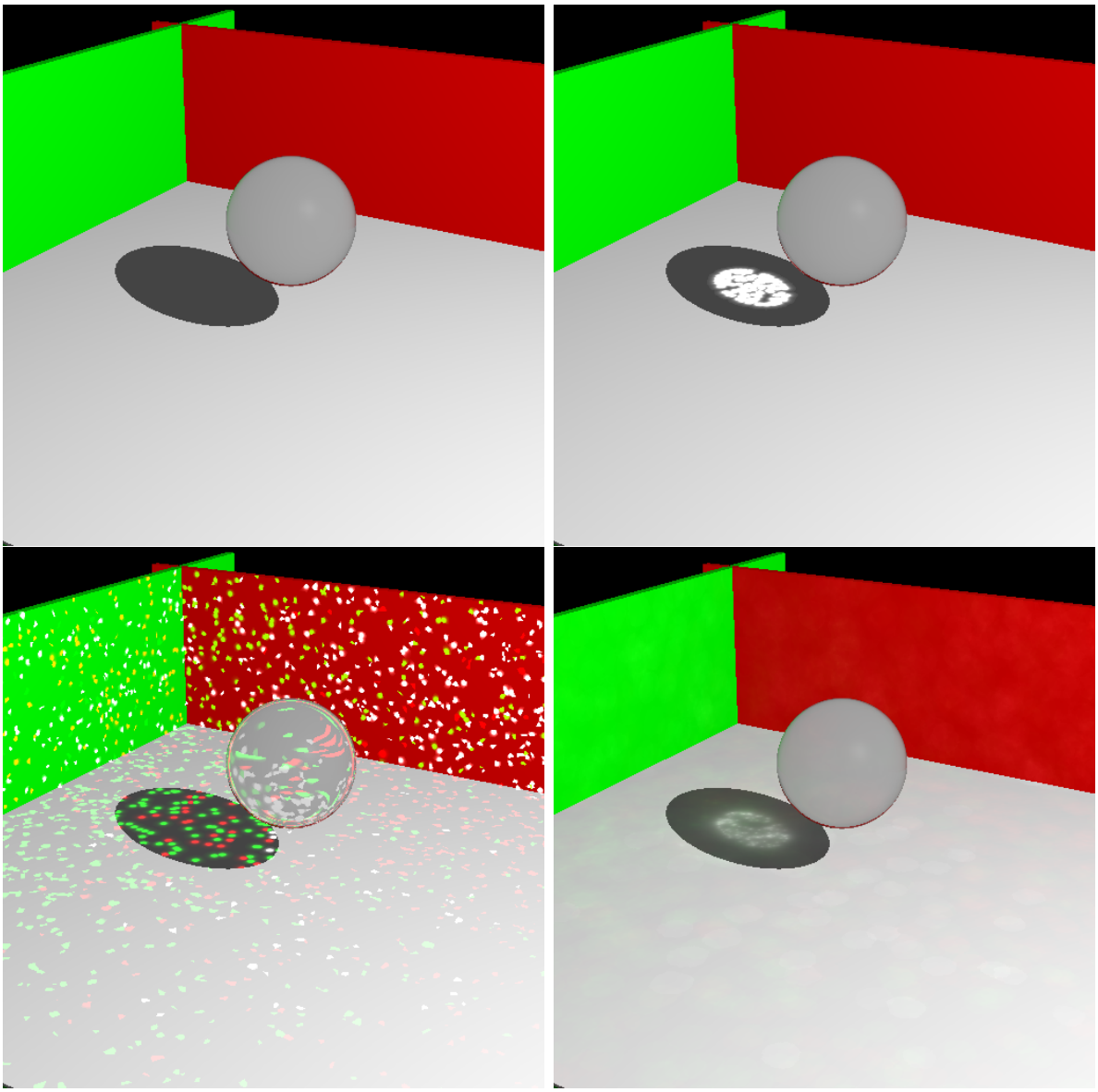

Figure 10: Each step of the photon-mapping process: before (top-left), caustics photons (top-right), global illumination photons (bottom-left), sampled and finished (bottom-right).

The *nanoflann* library was used for nearest-neighbour search. It uses kD-trees internally.

A fixed number of photons are cast from each point light source with a random direction, one set for global illumination and one set for caustics. Upon collision with a surface, a Russian roulette Monte Carlo simulation is used to determine how the photon travels. Because I chose to use these only for indirect lighting and caustics, I restricted the requirements for absorption as follows, where L is light, D is a diffuse surface, S is a specular surface, Rl is a reflective surface, Rr is a refractive surface, and + is at least 1:

- Indirect lighting (global illumination): L(D—S)+D.
- Caustics: L(Rl—Rr)+D.

The reason for this is that it reduces splotchiness overall, and I get to keep regular Phong lighting. If a photon fails to collide or is absorbed before meeting the requirements, it is treated as a colourless photon. A photon's reflective and refractive behaviour make use of the surface's material k_d, k_s, texture colour, reflectivity, and refractivity.

Objective 10: Architectural scene

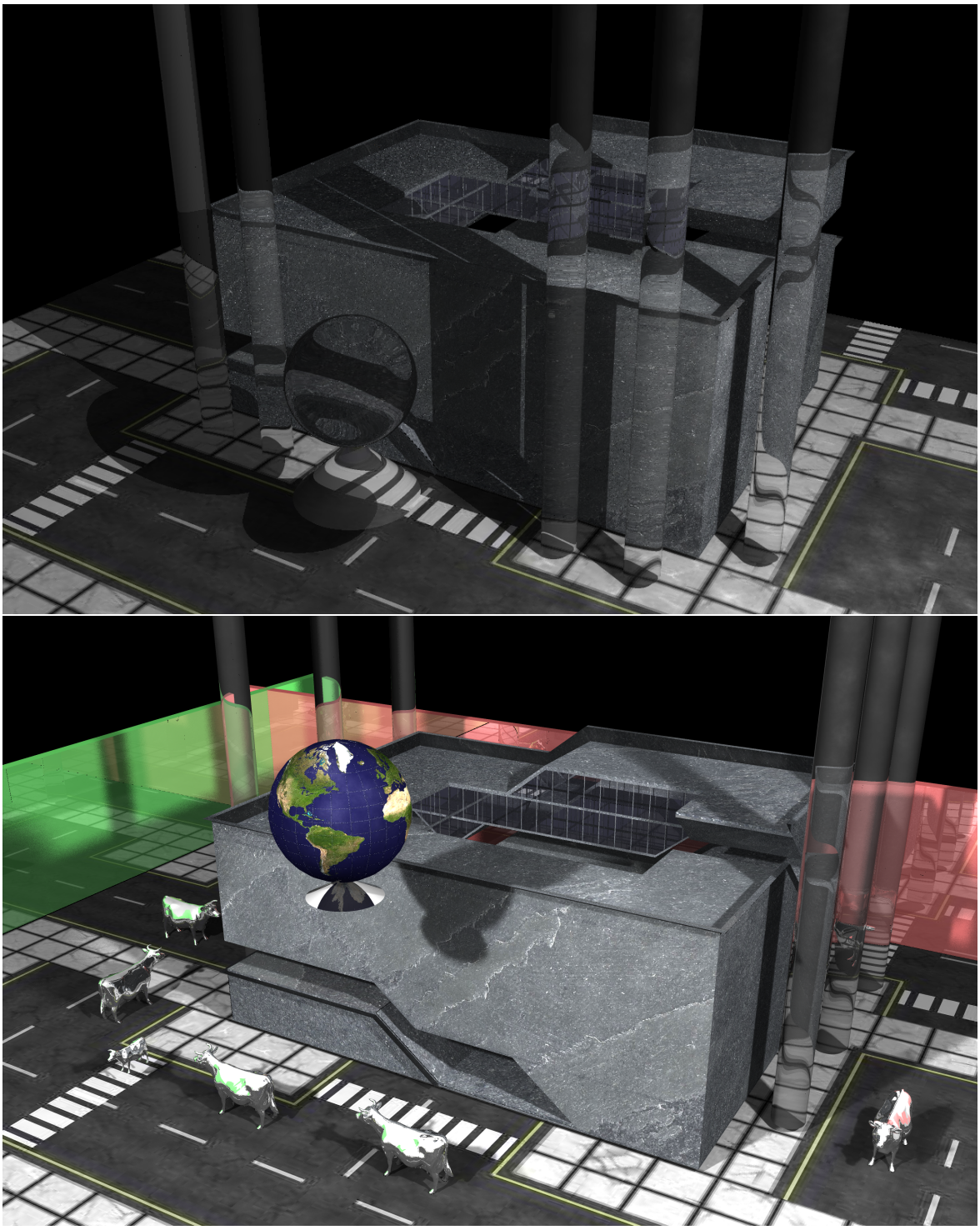

Figure 11: Early render to determine scene contents (above) and final composition (below).

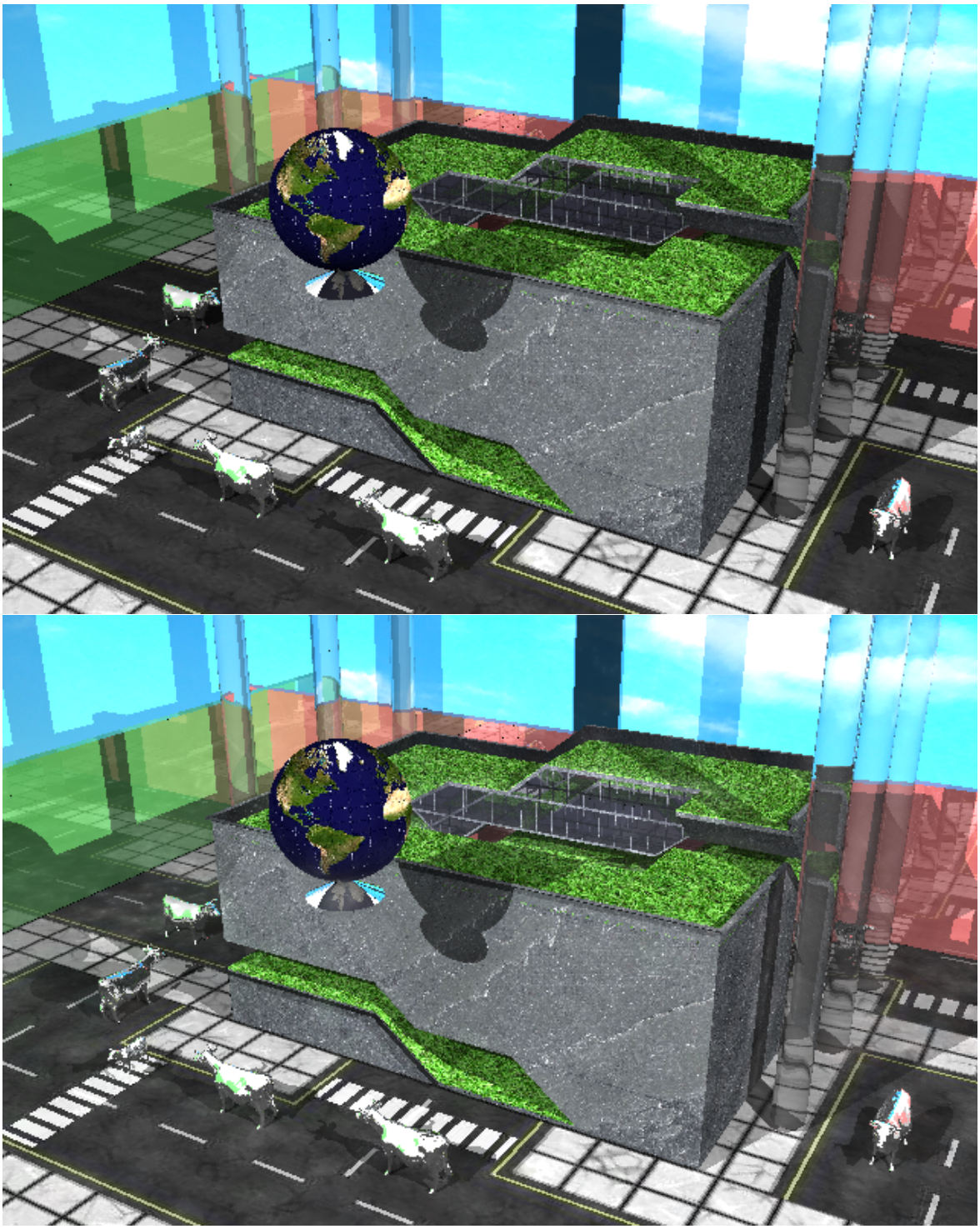

Figure 12: Early final render, comparing no photon mapping (left) and with photon mapping enabled (right). Note the bright caustic spots around the cows and on the grass.

Adaptive antialiasing was used at 2x for the scene. Reflections are enabled for the windows, cows, and coloured walls. Refractions are present for the pillars. Phong shading is used for the globe and cows. Both the cylinder and cone were used (cylindrical pillars, cone below globe). Building, globe, and ground are texture-mapped and bump-mapped. Photon mapping is used for both global illumination as well as some caustics. Soft shadows are present with 1 degree of jitter and 20 lighting rays per regular lighting ray.

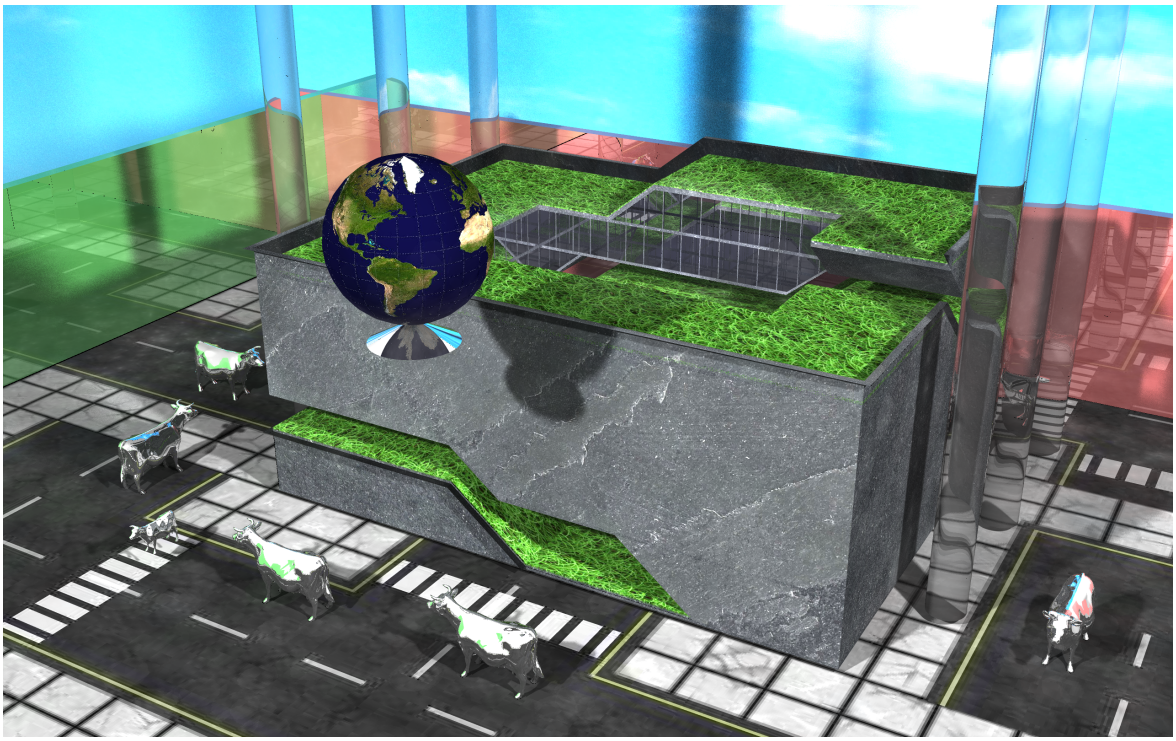

Figure 13: The final render. Took just over 547 minutes (9:07) on an i5-5257U processor multi-threaded with four threads, with 2x AAA and soft shadows enabled.

Bonus objective 1: Multithreading

Table 2: Timing comparisons vs. single-threaded for modified `simple-cows.lua` on an i5-5257U with 2c/4t. Times taken are average of four trials.

Threads	Time taken (s)	Speed-up ratio	Thread efficiency
1	29.495	1.0	100%
2	14.927	1.98	98.8%
3	14.171	2.08	69.4%
4	13.749	2.15	53.6%
8	13.960	2.11	26.4%
16	14.113	2.09	13.1%

We see that the threading efficiency occurs with two threads, which corresponds to the two physical cores in the CPU. The maximum speed-up is seen with four threads, which corresponds to the four hyperthreaded logical cores of the CPU. Beyond four threads, the time required to synchronize and task switch actually slow the performance down. Note that in the final implementation, the ray tracer will either use the C++-detected number of logical cores as threads, if present, or fall back to four.

Bonus objective 2: Soft shadows

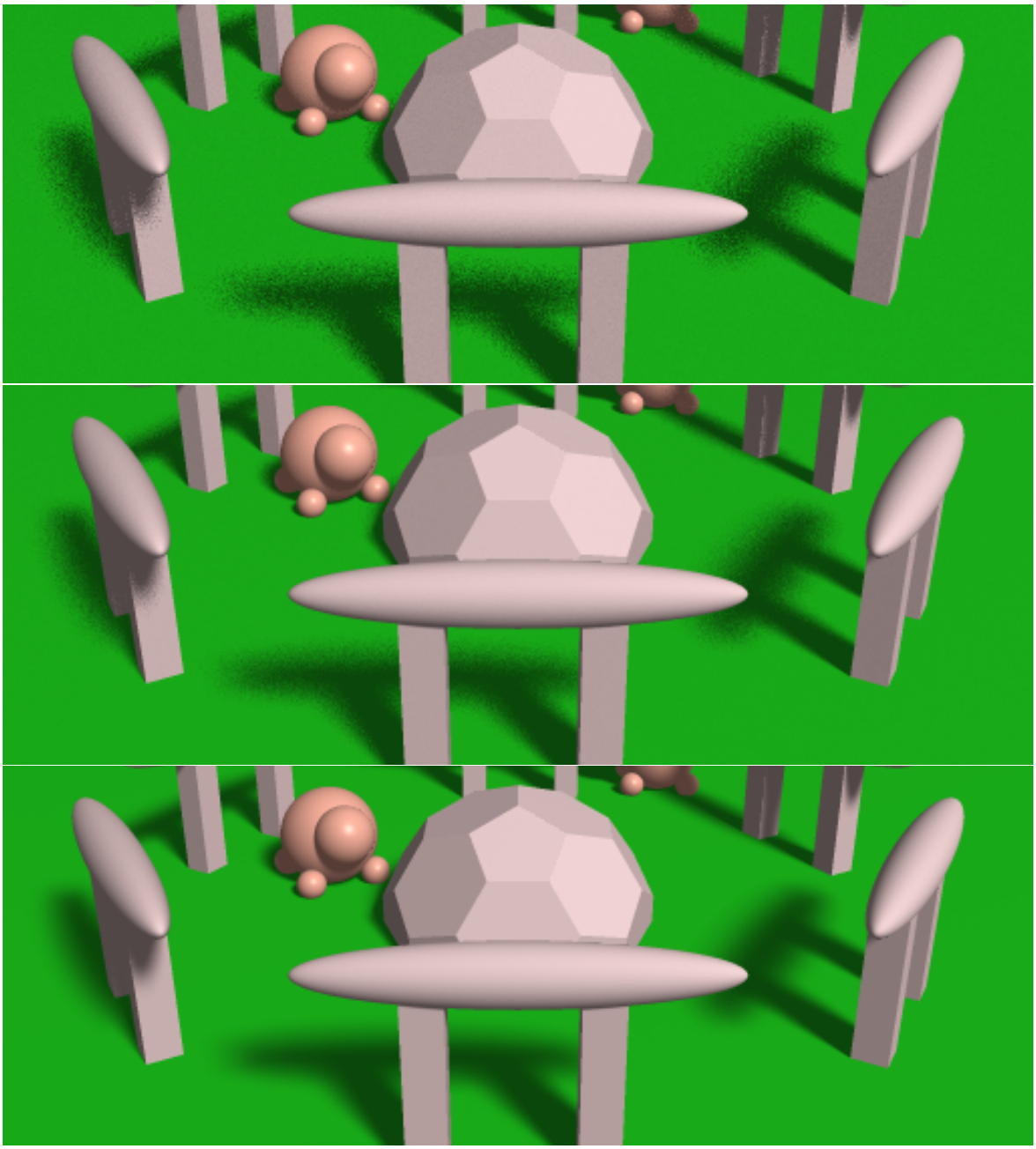

Figure 14: Monte Carlo soft shadows with 5° jitter, using, in order, 10, 50, and 5000 lighting rays per original lighting ray.

The rendering times increase more than linearly with soft shadow count due to adaptive antialiasing. It was found experimentally that 1° jitter with 40 lighting rays per original lighting ray gave reasonable results at lower resolutions with sufficient fidelity. This is the setting that will be used in the final render.