

3GC3 Final Project

Emily Horsman

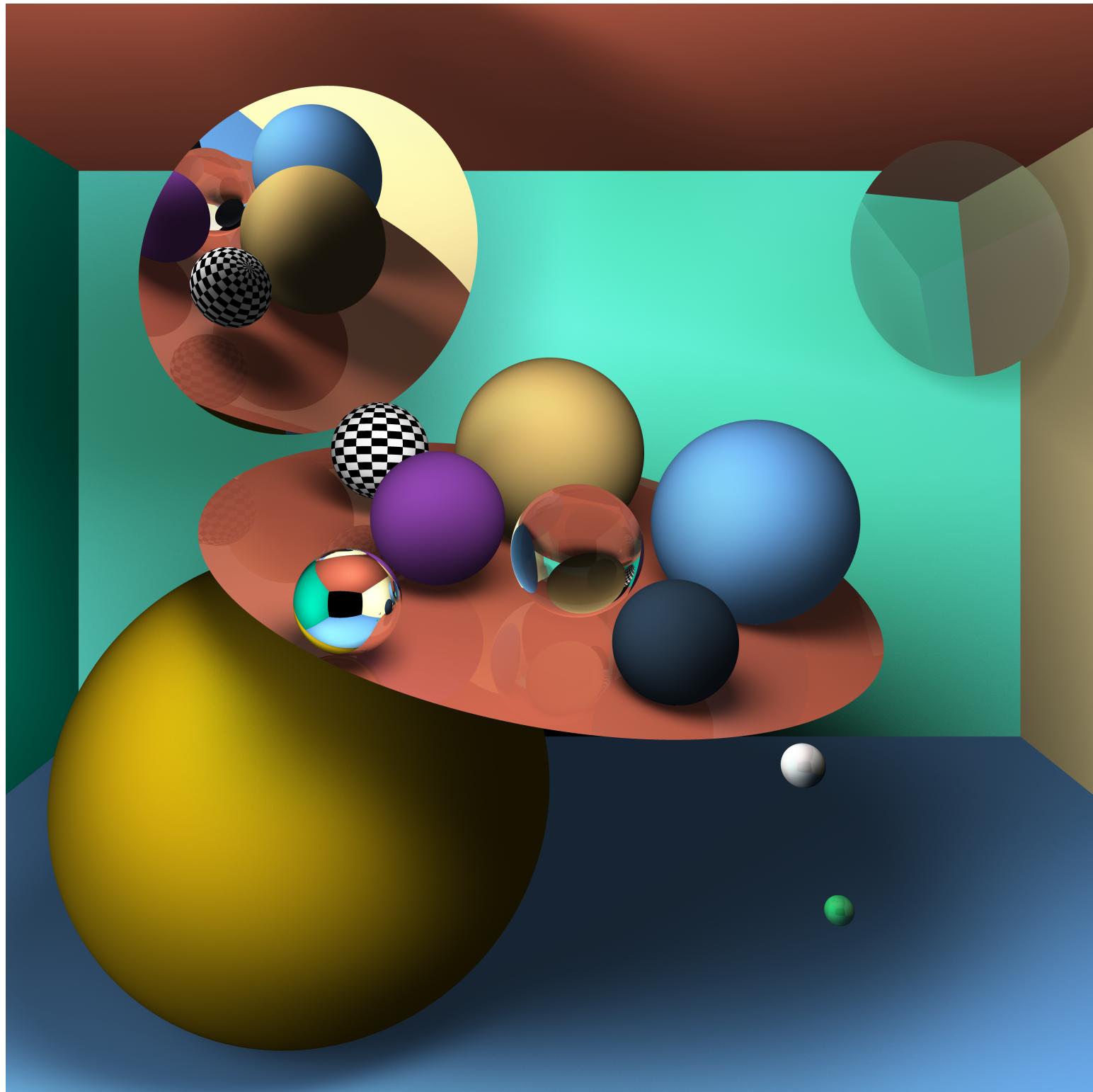


Figure 1: `examples/Sample.scene` Disks, planes, and spheres demonstrating reflection, refraction, checkered texture mapping, and material configurations. 4x regular sampled anti-aliasing with 100 iterations for soft shadows. 36 hours of CPU time for a 4000x4000px image.

Contents

1 Feature Overview	2
2 Execution	2
2.1 Compile	3
2.2 Unit tests	3
2.3 Generating documentation	3
2.4 Run a scene file	3
2.5 Writing a scene file	3
2.6 Guidance	4
2.7 Scene file properties	5
3 Depth of Field	6
4 Soft Shadows	7
5 Refraction	8
6 Reflection	8
7 Anti-Aliasing	9
8 Further Scenes	14
9 What's Missing?	16
10 References	17

1 Feature Overview

- CPU-based ray tracer
- Output to PPM bitmaps and a GLUT window
- Scene files for configuring rendering and scene parameters, materials, lights, and objects
- Support for planes, spheres, and disks
- Color and checkered material with proper texture mapping for spheres
- Materials specify coefficients for: ambient, diffuse, and specular light; transmission; index of refraction
- Multi-threaded rendering with a configurable number of threads
- Rudimentary refraction (no Fresnel effect)
- Soft shadows achieved by “jittering” point lights and averaging multiple renders
- Anti-aliasing with both regular (uniform) and random sampling techniques
- Adjustable depth-of field and camera field of view
- Code documentation from Doxygen in `html/index.html`

2 Execution

Running the ray tracer requires a scene file. The root of this repository includes `sample.scene` which will be used by default if no scene file is provided. Running `make` from the root of the repo will compile and run the ray tracer with the sample scene. Note that the sample scene file specifies `threads: 12` by default. This should be adjusted according to your system.

The image will be rendered to a PPM bitmap and a GLUT window. I highly recommend viewing the PPM bitmap, but the GLUT window is convenient for a quick glance.

2.1 Compile

```
$ make
<...compiling and linking...>
./Ray
==== Render Info ./sample.scene ====
Target          OpenGL, ./sample.ppm
Image Dimension 900 x 700
Threads         12
Max Depth       3
Anti-Aliasing   4
Sampling Method Regular
Soft Shadows?   No
Iterations       1

Field of View   90
Eye             0, 0, 0.25
Focal Point     0.5, 0.8, -2
Aperture Radius Pinhole

Objects         9
Lights          1
```

2.2 Unit tests

There is a small suite of unit tests included. These primarily cover ray-object intersections.

```
$ make test
```

2.3 Generating documentation

This repository uses `Doxygen` to generate code documentation. It includes a minimally configured `Doxyfile`.

```
$ make docs
```

2.4 Run a scene file

The `Ray` executable accepts a single command-line argument for a scene file path.

```
$ ./Ray ./examples/DepthOfField.scene
```

2.5 Writing a scene file

Scene files are just plain text files containing sections separated by an extra newline. The best way to write new ones is to reference `sample.scene` and `examples/*.scene`.

Each section starts with a tag such as `Renderer` or `Material`. A tag is followed by properties in the format of `key: value`. Global parameters such as rendering (e.g., number of threads) and camera options (e.g., field of view) are placed under a `Renderer` and `Camera` section. Objects reference materials by an ID you give when writing the material. You must specify the material section before the objects which use it.

A minimal sample scene file is produced below.

```
Renderer
antiAliasing: 4
samplingMethod: random
useSoftShadows: true
iterations: 30
threads: 12
outputFile: ./minimal.ppm

PointLight
position: 0, 1, -1
radius: 0
intensity: 1
```

```
CheckerboardMaterial largeCheckers
color: 1, 1, 1
odd: 0, 0, 0
grain: 0.5
ambient: 0
diffuse: 0.9
specular: 0.1
```

```
Plane
material: largeCheckers
point: 0, -1, 0
normal: 0, 1, 0
```

Materials are tagged by their type, followed by their unique ID: `Material`, `CheckerboardMaterial`. Objects are tagged by their type: `Plane`, `Sphere`, `Disk`. A light is tagged with `PointLight`.

2.6 Guidance

- Turning on soft shadows or using a non-zero aperture radius will require setting `iterations` higher to reduce noise. The precise value depends on the desired quality, effect (sometimes the noise is pleasant, like an artistically chosen film speed in photography), and computational resources.
- I've found `samplingMethod: random` to produce better quality images over `samplingMethod: regular`.
- Objects and lights are placed with $z < 0$. The camera `eye` is $z > 0$.
- Keep soft shadows off and iterations low while testing, and then increase these properties to the desired values.
- `antiAliasing: 4` will improve image quality considerably without much extra render time.

2.7 Scene file properties

Enumerations are delimited by pipes |.

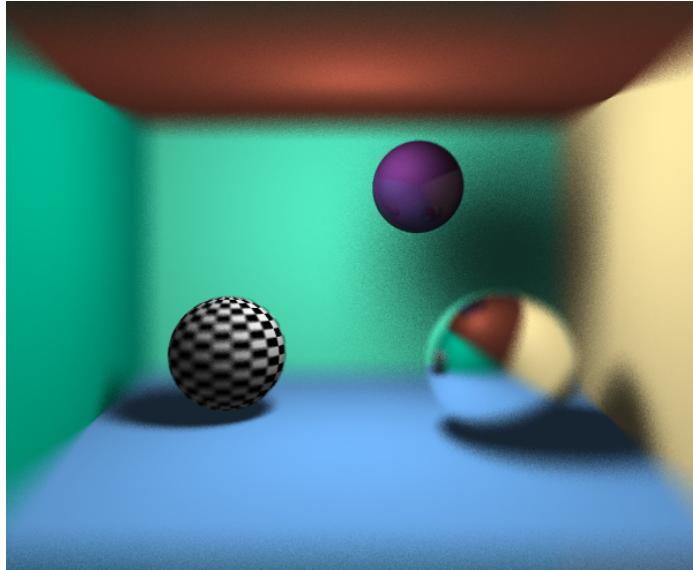
Tag	Property	Type	Example	Description
Renderer	width	int	1200	
	height	int	900	
	maxDepth	int	10	Maximum recursion depth for specular and transmission rays.
	antiAliasing	int	16	Number of anti-aliasing samples per pixel, must be a square number.
	samplingMethod	string	regular random	The type of anti-aliasing sampling.
	iterations	int	100	Render the image for n iterations and average the result.
	threads	int	4	Multi-threading.
	useSoftShadows	bool	true false	Soft shadows will “jitter” point lights around their radius.
Camera	outputFile	string	./Scene.ppm	Rendered image path.
	fieldOfViewDegrees	int	45	
	lookAt	Vec3f	0.5, 0.5, -2	Focal point.
	eye	Vec3f	0, 0, 0.25	Camera position.
	apertureRadius	float	0.2	Use 0 for pinhole/infinite DOF. Higher values have larger DOFs.
PointLight	position	Vec3f	0, 1, -1.5	
	intensity	float	0.7	[0,1]
	radius	float	0.1	Must enable soft shadows in Renderer .
Material id	color	Vec3f	1, 0.1, 0.2	
	ambient	float	0.1	[0,1]
	diffuse	float	0.7	[0,1]
	specular	float	0.2	[0,1]
	transmission	float	1	[0,1] From opaque to 100% transparent.
	refractiveIndex	float	1.5	$\eta \geq 1$
CheckerboardMaterial		Shares all Material properties.		
id	odd	Vec3f	1, 0, 0	Color of odd squares. color is for even squares.
	grain	float	0.5	Size of checkers.
Sphere	origin	Vec3f	0, 0.5, -1.5	
	radius	float	0.5	
	material	string	blue	Must be a previously tagged material.
Plane	point	Vec3f	0, -1, 0	Any point in the plane.
	normal	Vec3f	0, 1, 0	
	material	string	blue	Must be a previously tagged material.
Disk	origin	Vec3f	0, -1, 0	Any point in the plane.
	normal	Vec3f	0, 1, 0	
	radius	float	0.05	
	material	string	blue	Must be a previously tagged material.

3 Depth of Field

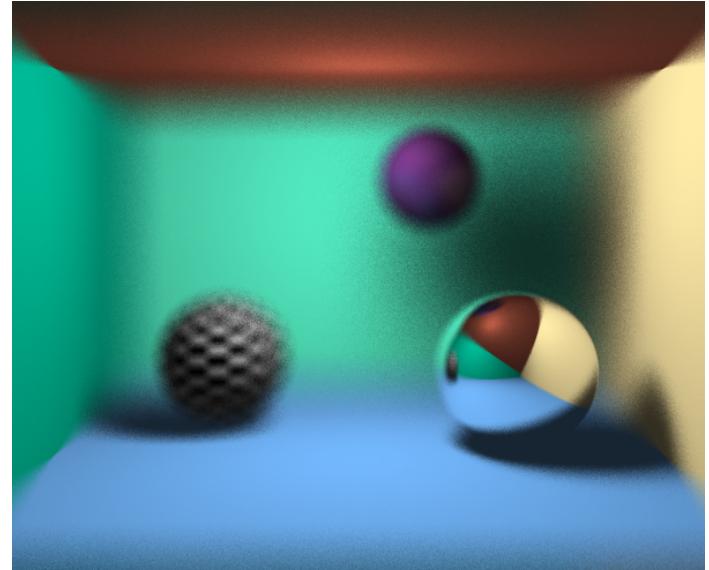
Primary rays normally are generated by casting a ray from a pinhole lens (a single point — the camera position) through pixels in the image plane. Call one of these primary rays \hat{p} . This achieves an infinite depth of field. This ray tracer simulates depth of field by computing a different primary ray from \hat{p} .

1. The `lookAt` property under `Camera` in your scene file defines the position of the fixed-normal focal plane in the scene. This should be in front of the camera and image plane. `lookAt` can be conveniently set to the origin of an object you'd like to be in focus.
2. Find the focal point f by computing the intersection from the eye and direction \hat{p} to the focal plane.
3. Find the aperture point a by taking the current pixel on the image plane and displacing it by a random amount within the circular aperture (defined with `apertureRadius` under `Camera` in your scene file).
4. Return a new primary ray with origin a and direction $f - a$ (normalized).

This process will introduce noise in the image due to the random displacement of the aperture point. This is solved by rendering multiple iterations and averaging the result to get the final render.



(a) `DepthOfField.scene`



(b) `DepthOfField2.scene`

Figure 2: A scene displayed with two different focal points and a very shallow depth of field. These renderings have minimal noise reduction applied. It can be increased with the `iterations` property in the `Renderer` section of your scene file.

4 Soft Shadows

Soft shadows are achieved by randomly displacing each point light within a sphere every time the direction of the light is computed for an intersection point. The radius of the volume is configured per light. This technique requires rendering multiple times and averaging the results. Only non-directional point lights are implemented. Thus, the direction of a radius r point light at position \hat{l} for an intersection \hat{p} is computed and normalized:

$$\hat{l} + r * \text{randomVector}(-1, 1) - \hat{p}$$

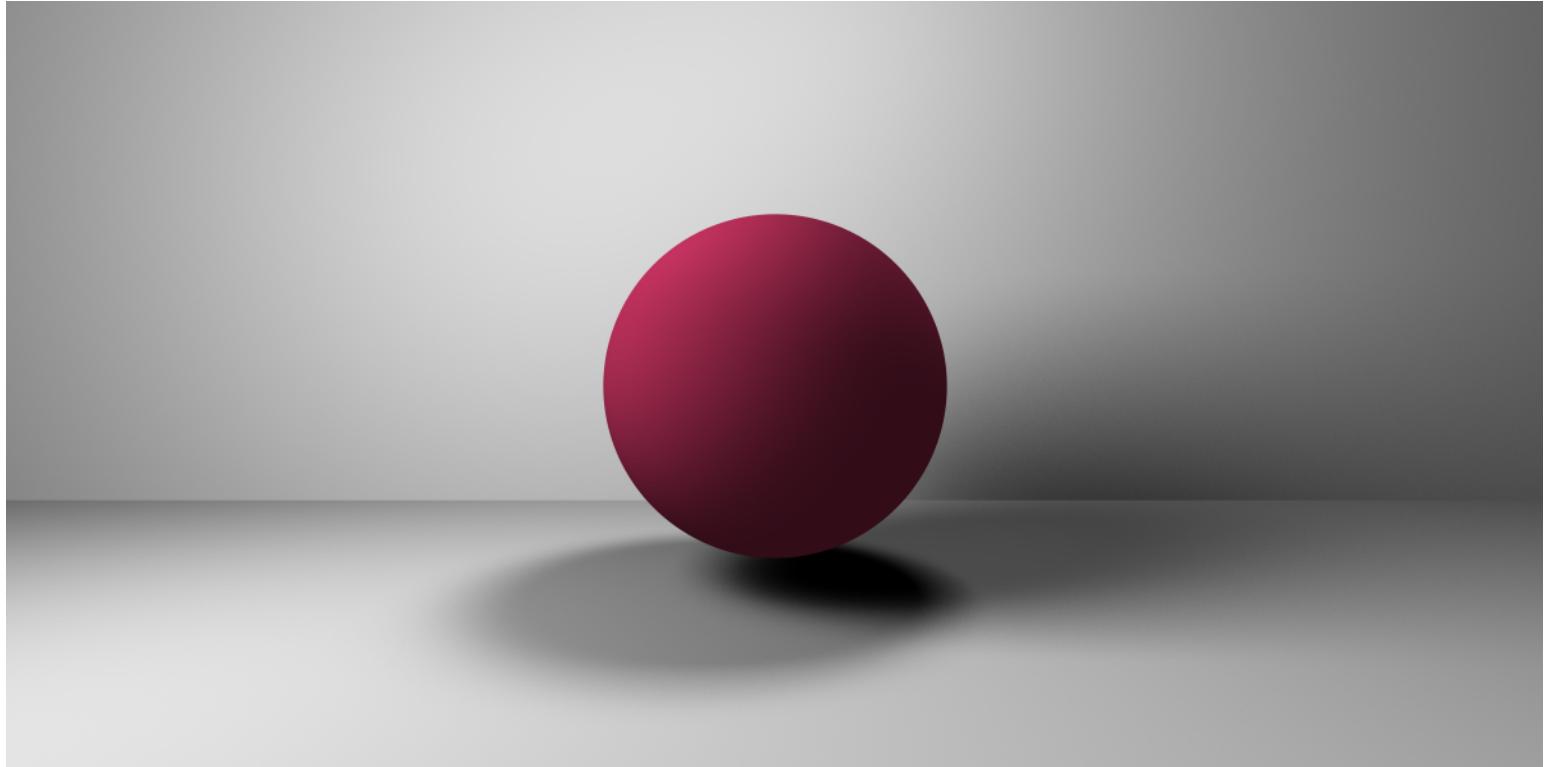


Figure 3: `SoftShadows.scene`. The `Renderer` section has properties `useSoftShadows: true` and `iterations: 500`.

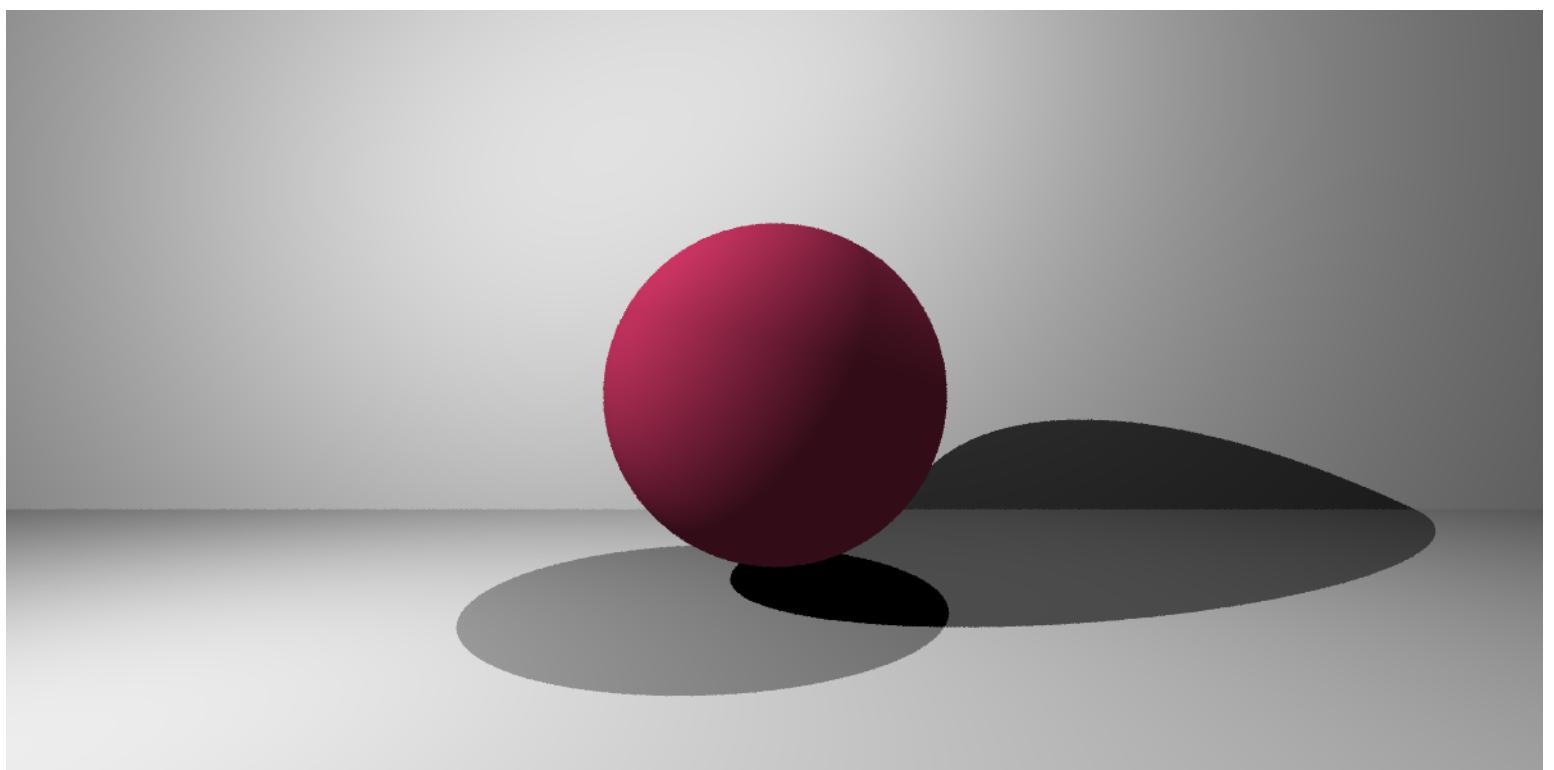


Figure 4: `SoftShadows.scene`. Properties `useSoftShadows: false` and `iterations: 1`.

5 Refraction

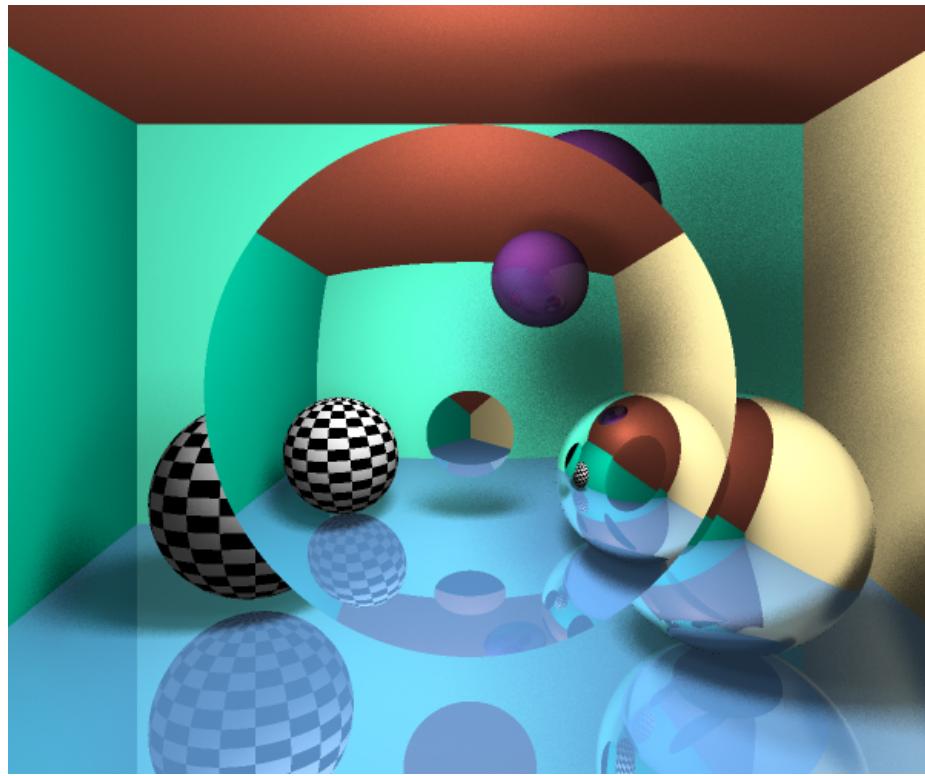


Figure 5: `DiskLens.scene`. A 100% transparent disk with a refractive index of 2.5 placed in front of the scene.

6 Reflection

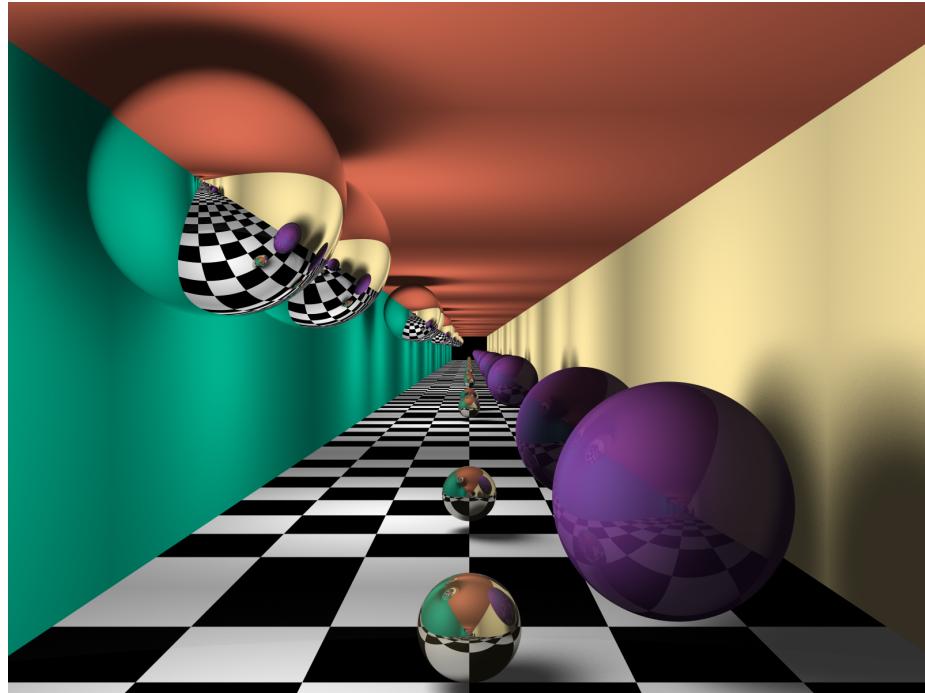


Figure 6: `HallOfMirros.scene`. Hall of mirrors with `maxDepth: 15` under `Renderer`.

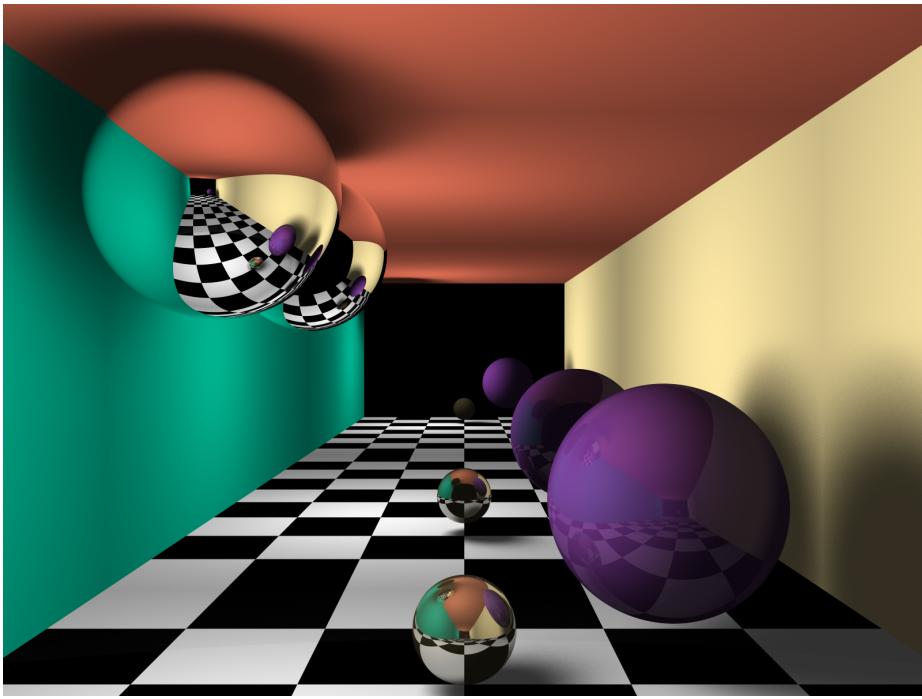


Figure 7: HallOfMirros.scene. Hall of mirrors with `maxDepth: 2` under `Renderer`.

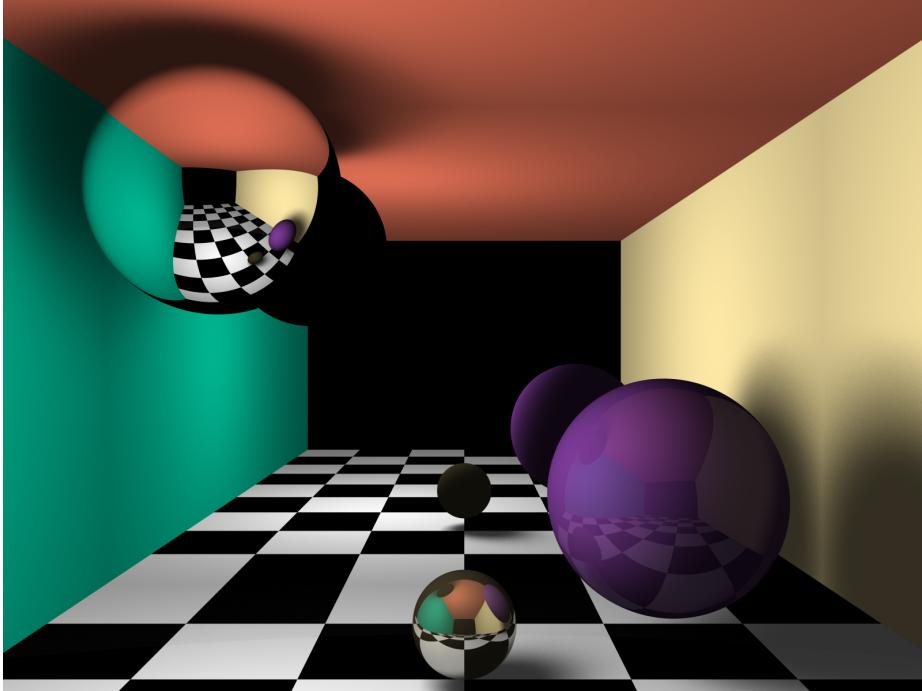


Figure 8: HallOfMirros.scene. Hall of mirrors with `maxDepth: 1` under `Renderer`.

7 Anti-Aliasing

Each pixel of the image is computed with a primary ray cast through the center of the pixel, when `antiAliasing` is 0 under `Renderer` in your scene file. Turning on anti-aliasing makes the renderer trace multiple rays per pixel and then averages the result. Regular anti-aliasing (`samplingMethod: regular`) uses a uniform grid with the unit square pixel for its sampling points. Random anti-aliasing (`samplingMethod: random`) selects random points within the unit square pixel as its sampling points. Figure 13 and 14 show how these two sampling methods look different, particularly in areas of the scene that are very far from the camera. Without anti-aliasing and with small amounts of regular anti-aliasing, these areas typically look distorted.

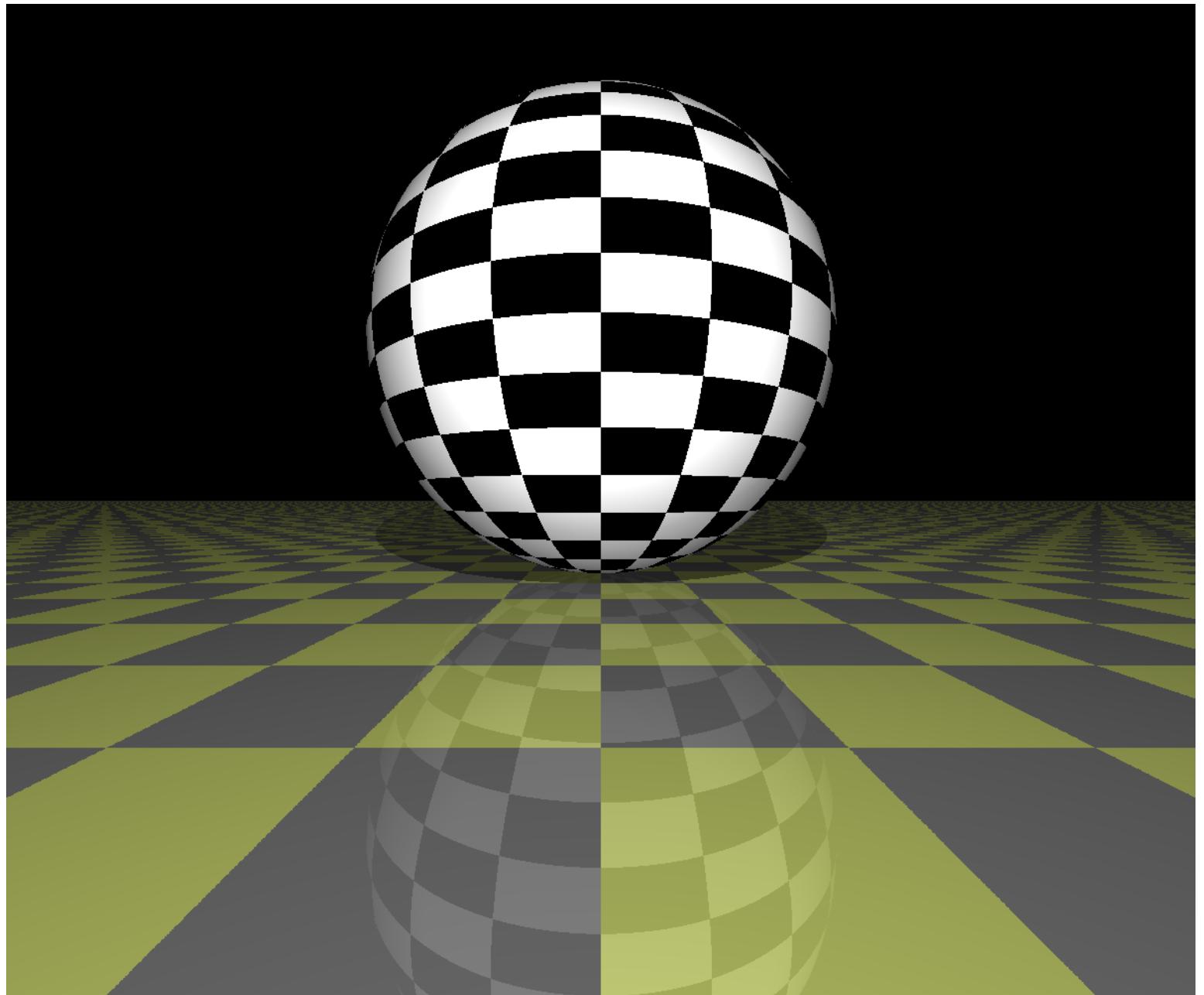


Figure 9: No anti-aliasing.

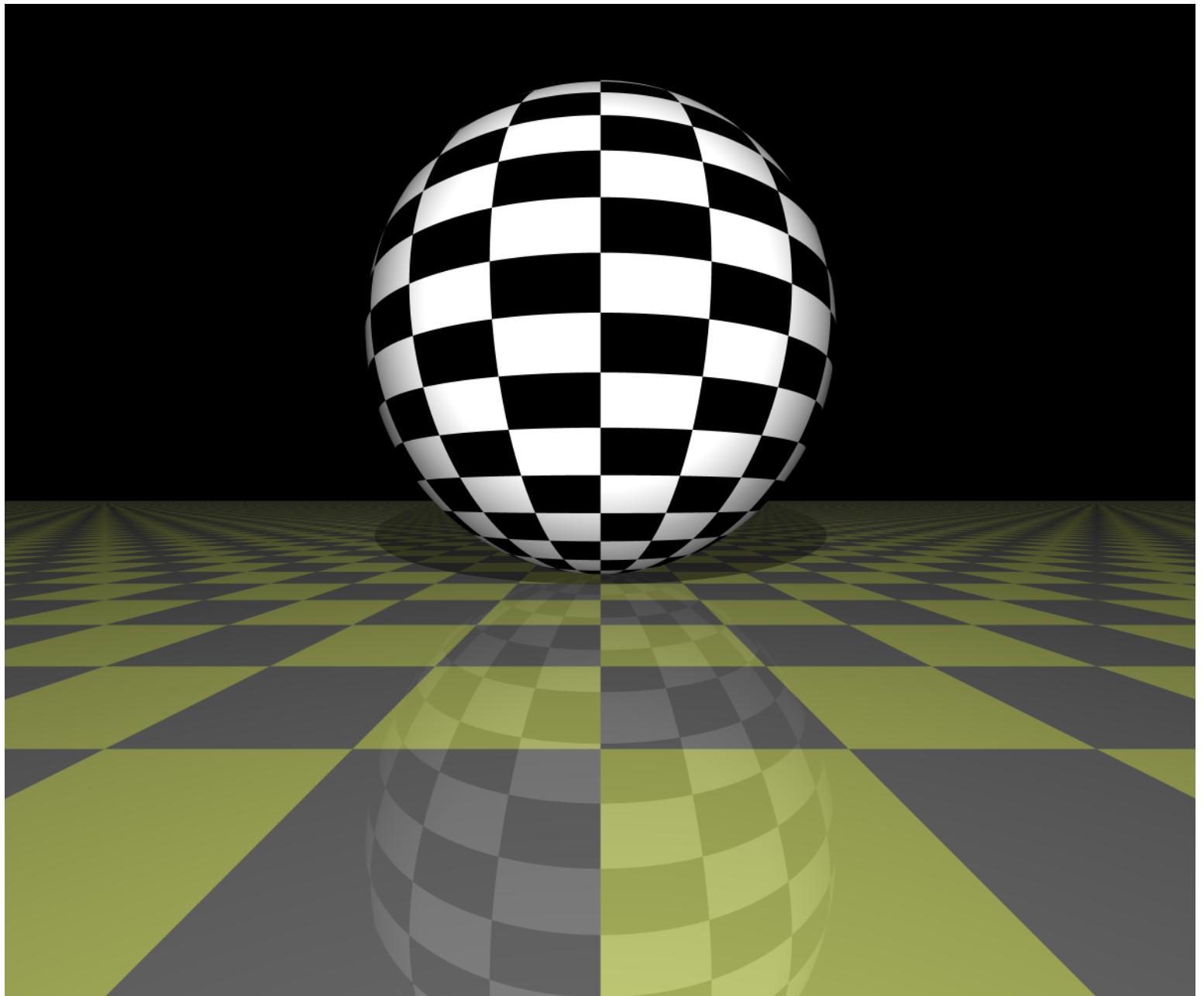


Figure 10: 64x AA using regular sampling.

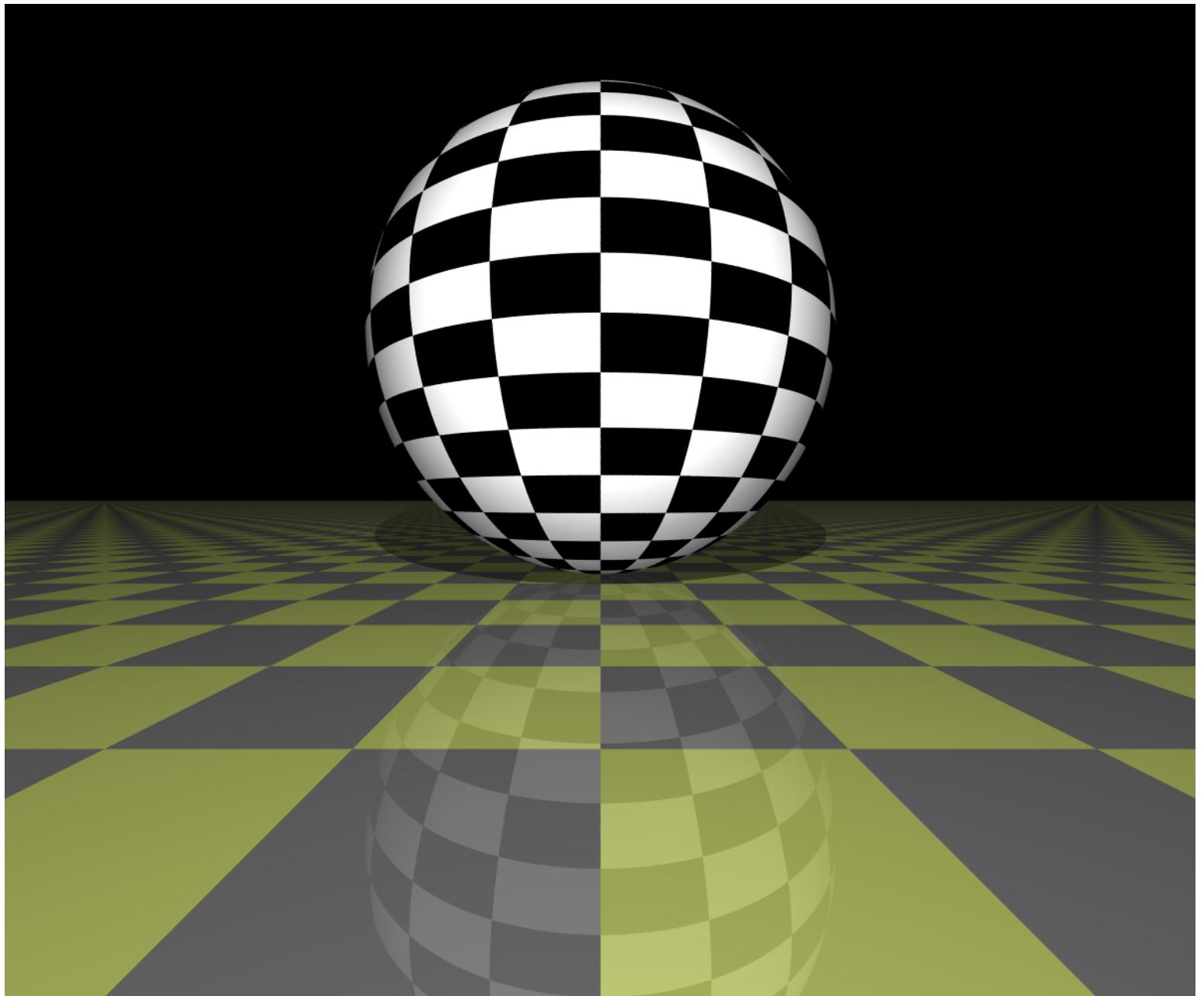


Figure 11: 64x AA using random sampling.

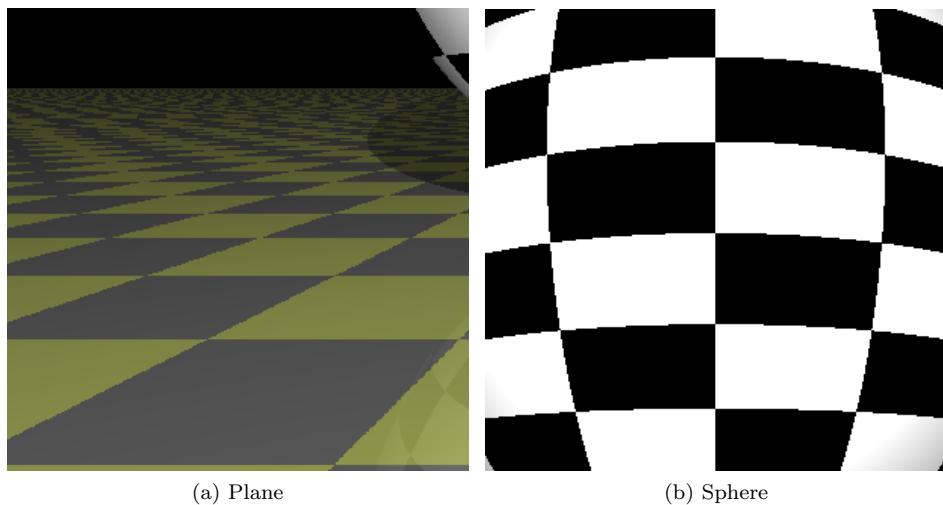


Figure 12: Vanishing plane and checkered sphere without anti-aliasing.

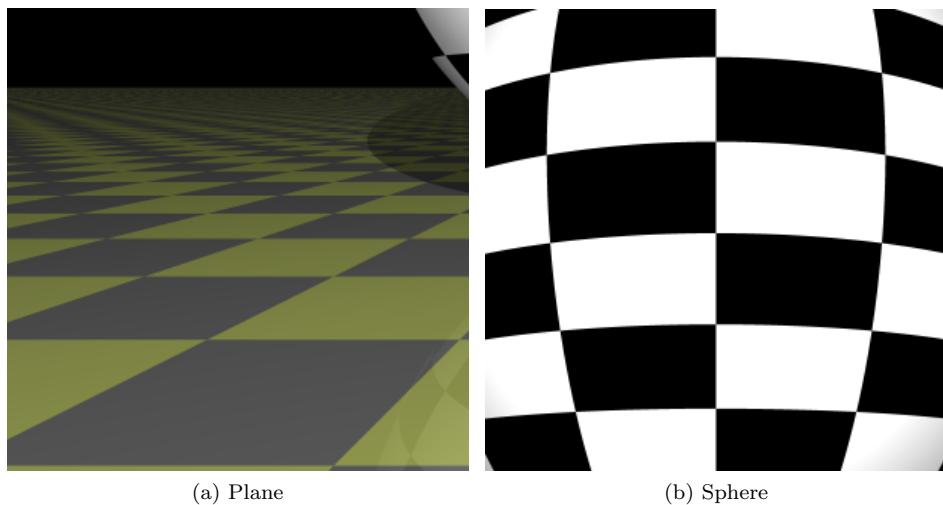


Figure 13: Vanishing plane and checkered sphere with 64x AA using regular sampling.

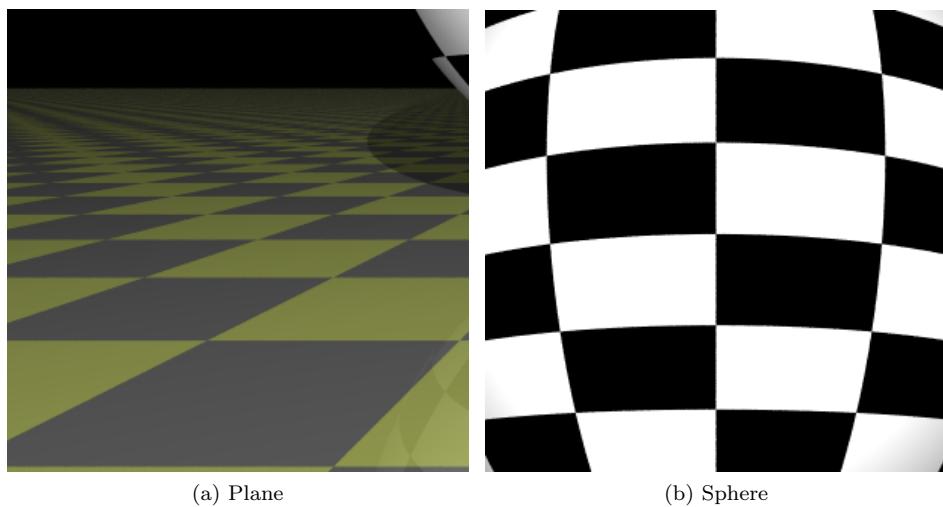


Figure 14: Vanishing plane and checkered sphere with 64x AA using random sampling.

8 Further Scenes

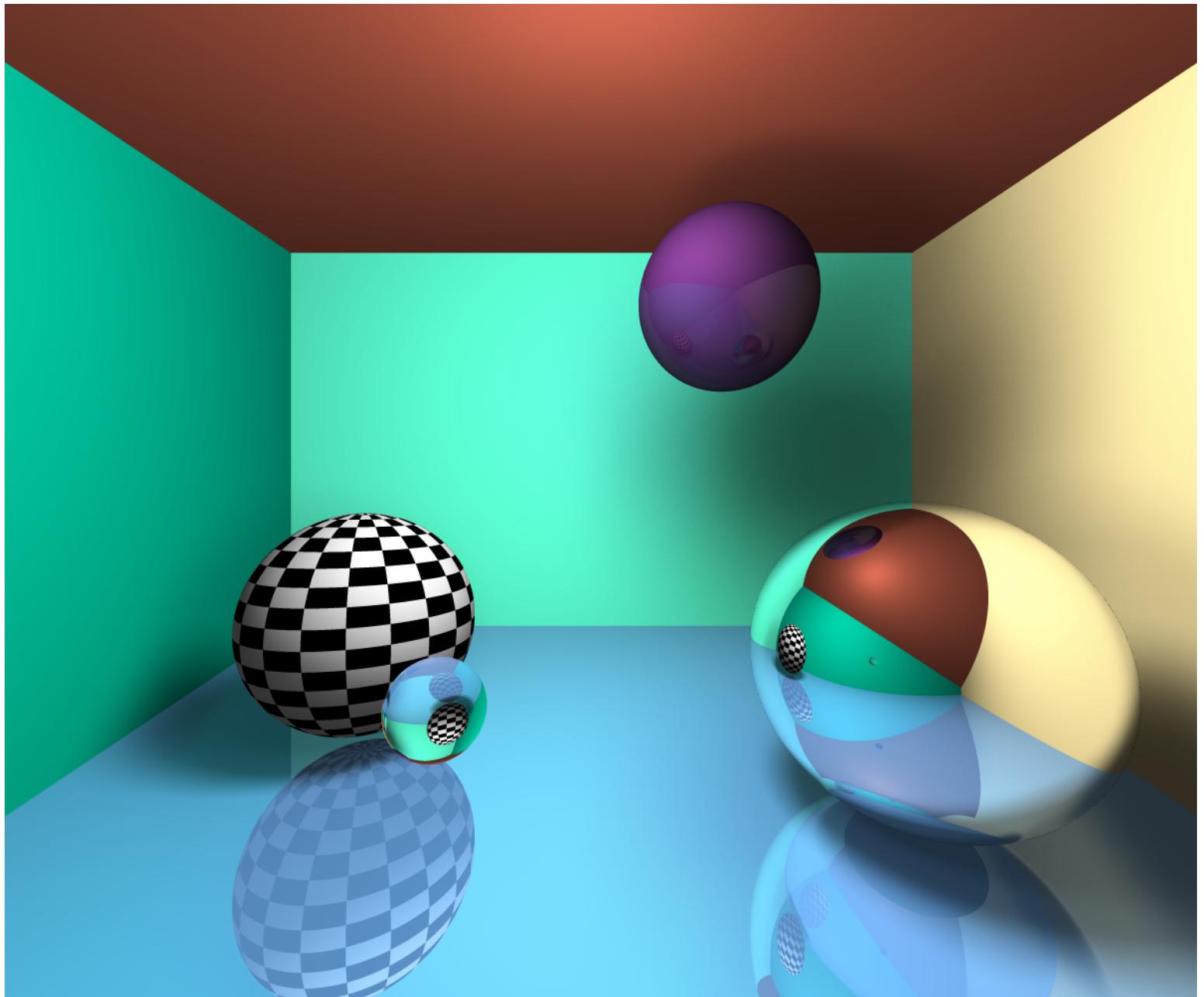


Figure 15: Reflective floor, 100% transparent sphere showing refraction, checkered sphere with texture mapping, mirror sphere. 64x random sampled anti-aliasing with 300 iterations for soft shadows. 13h50m of CPU time.

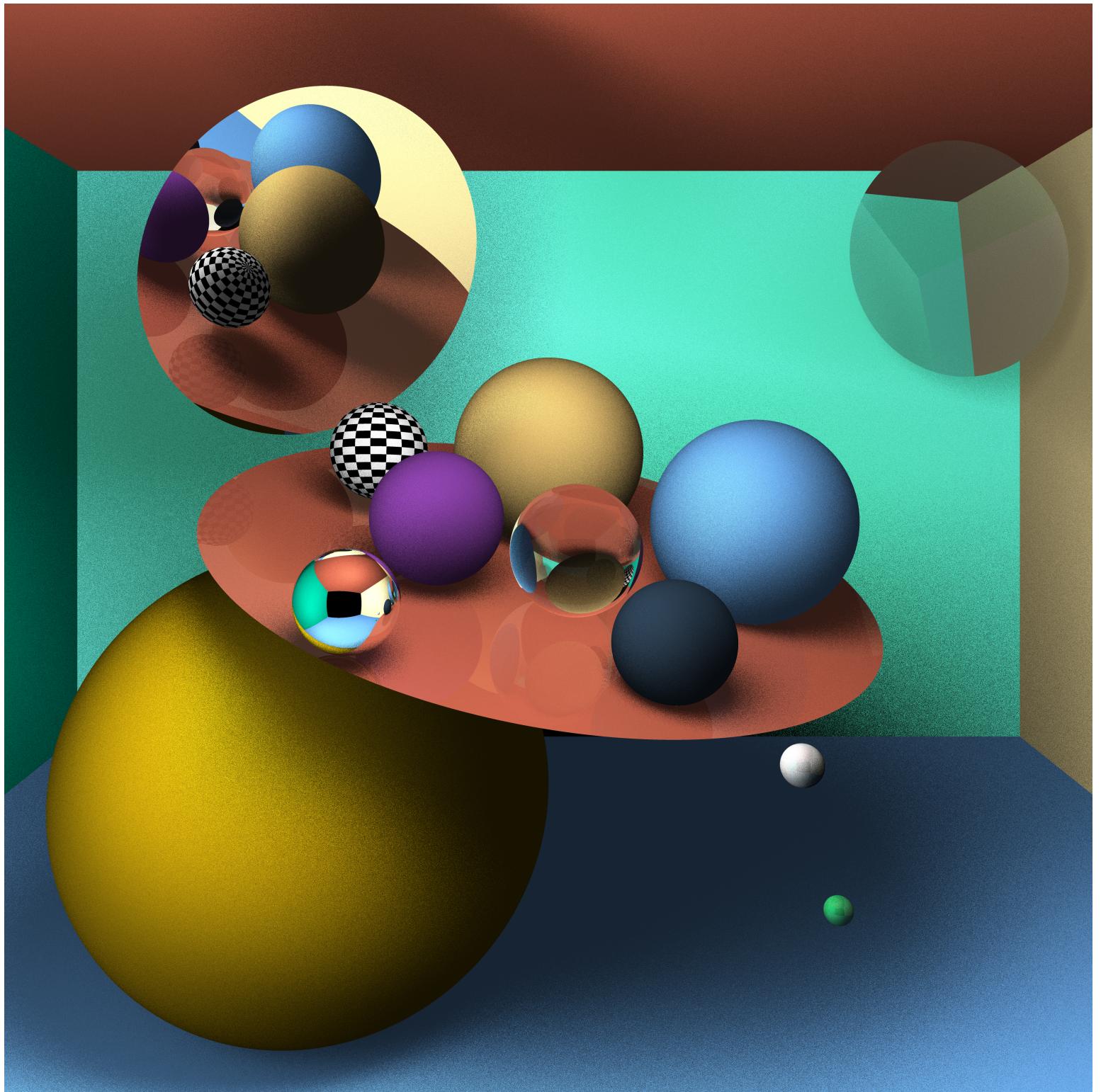


Figure 16: examples/Sample.scene Figure 1 run with only a single iteration, iterations: 1

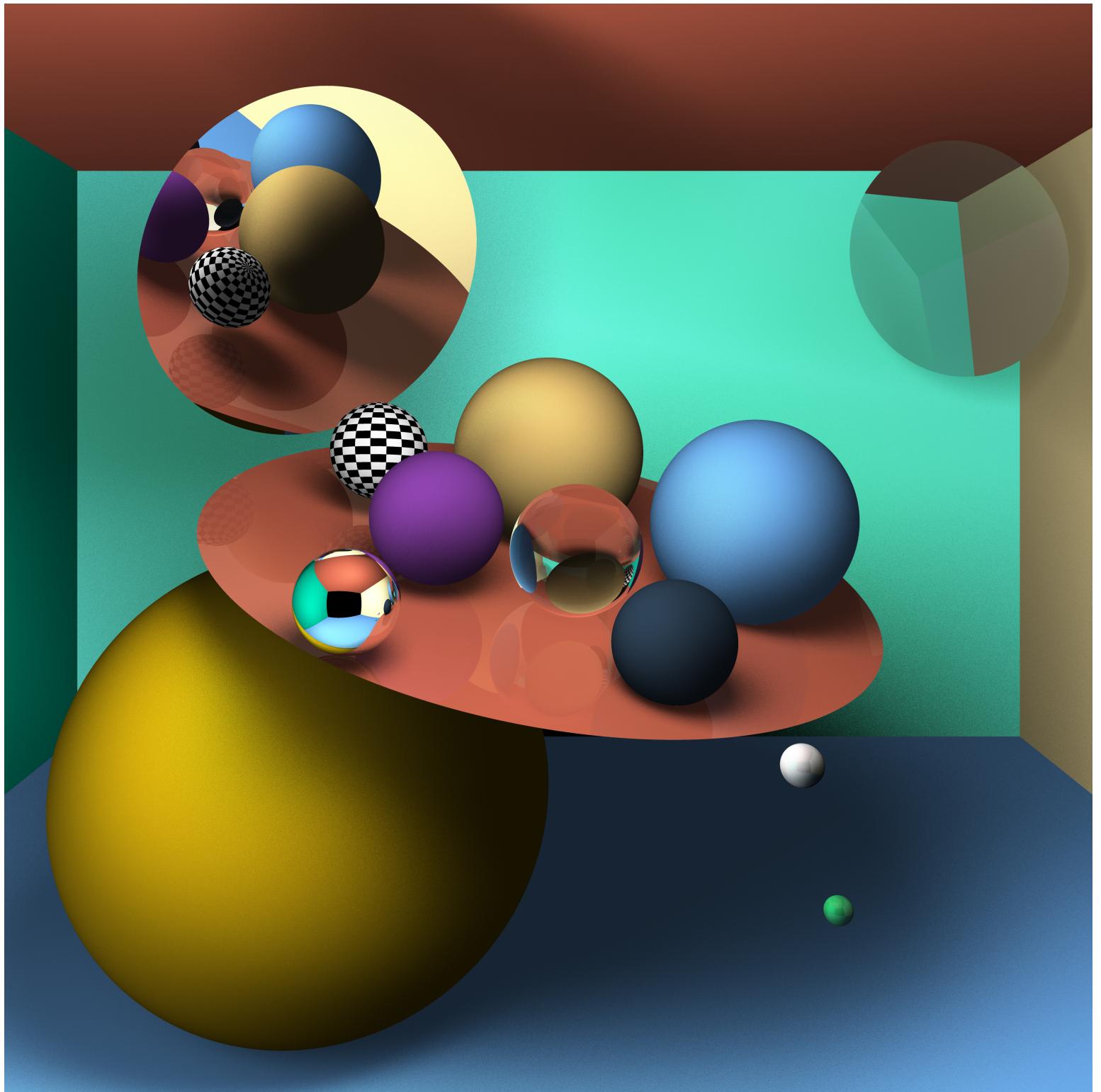


Figure 17: examples/Sample.scene Figure 1 run with iterations: 10

9 What's Missing?

These are some things I'd love to do in the future but didn't have time for this semester!

- Quadric objects
- Fresnel effect
- Rendering lights in the scene
- 6-DOF camera
- Triangles with affine transformations

- Mesh files
- A regular grid or k-d tree for performance gains
- Specular highlights

10 References

These items are referred to throughout the codebase with [n].

1. <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/generating-camera-rays>
2. <http://www.3dkingdoms.com/weekly/weekly.php?a=2>
3. <https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel>
4. https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006-degreve-reflection_refraction.pdf
5. Chapter 10 of “Ray Tracing from the Ground Up” (Kevin Suffern)
6. <http://cg.skeelogy.com/depth-of-field-using-raytracing/>
7. <https://stackoverflow.com/a/13686064/4909532>
8. <https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection>
9. <https://people.cs.clemson.edu/~dhouse/courses/405/notes/texture-maps.pdf>
10. Exercise 18.1 from “Ray Tracing from the Ground Up” (Kevin Suffern) p.350