STUDENT Maksym Pylypenko
ID: 7802672
Email: pylypenm@myumanitoba.ca

Ray tracing project

This is a recursive implementation of a ray tracer in C++. List of completed features below...

Reflection

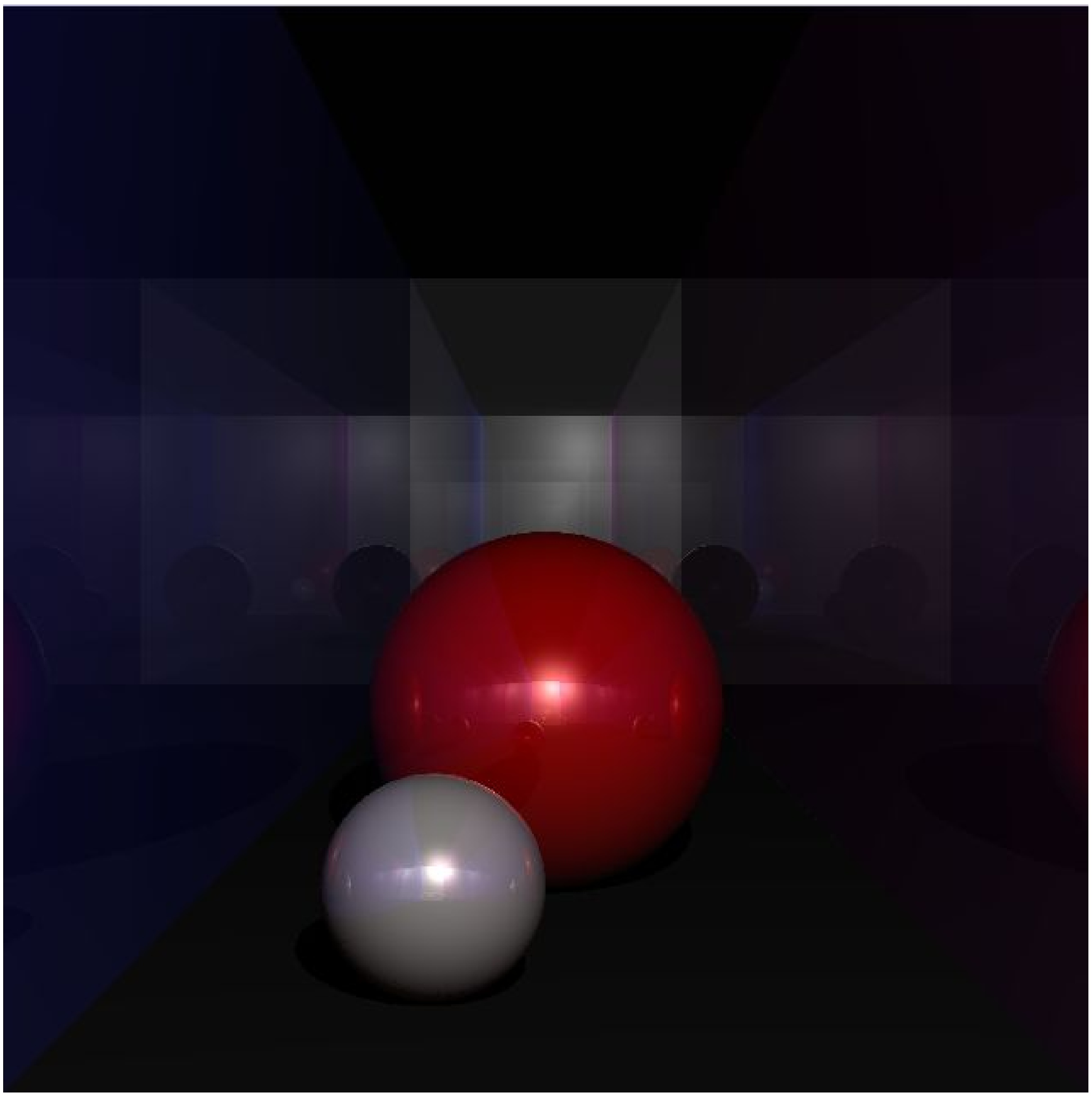
When the ray hits a reflective surface, it can bounce.

Usage example (add this for any object):

```
{
        "type": "sphere",
        "radius": 1.0,
        "position": [ 0.0, -1.0, -10.0 ],
        "material": {
                "ambient": [ 0.2, 0.0, 0.0 ],
                "diffuse": [ 0.6, 0.0, 0.0 ],
                "specular": [ 0.7, 0.6, 0.6 ],
                "shininess": 25,
                "reflective": [ 0.5, 0.5, 0.5 ]
        }
},
```

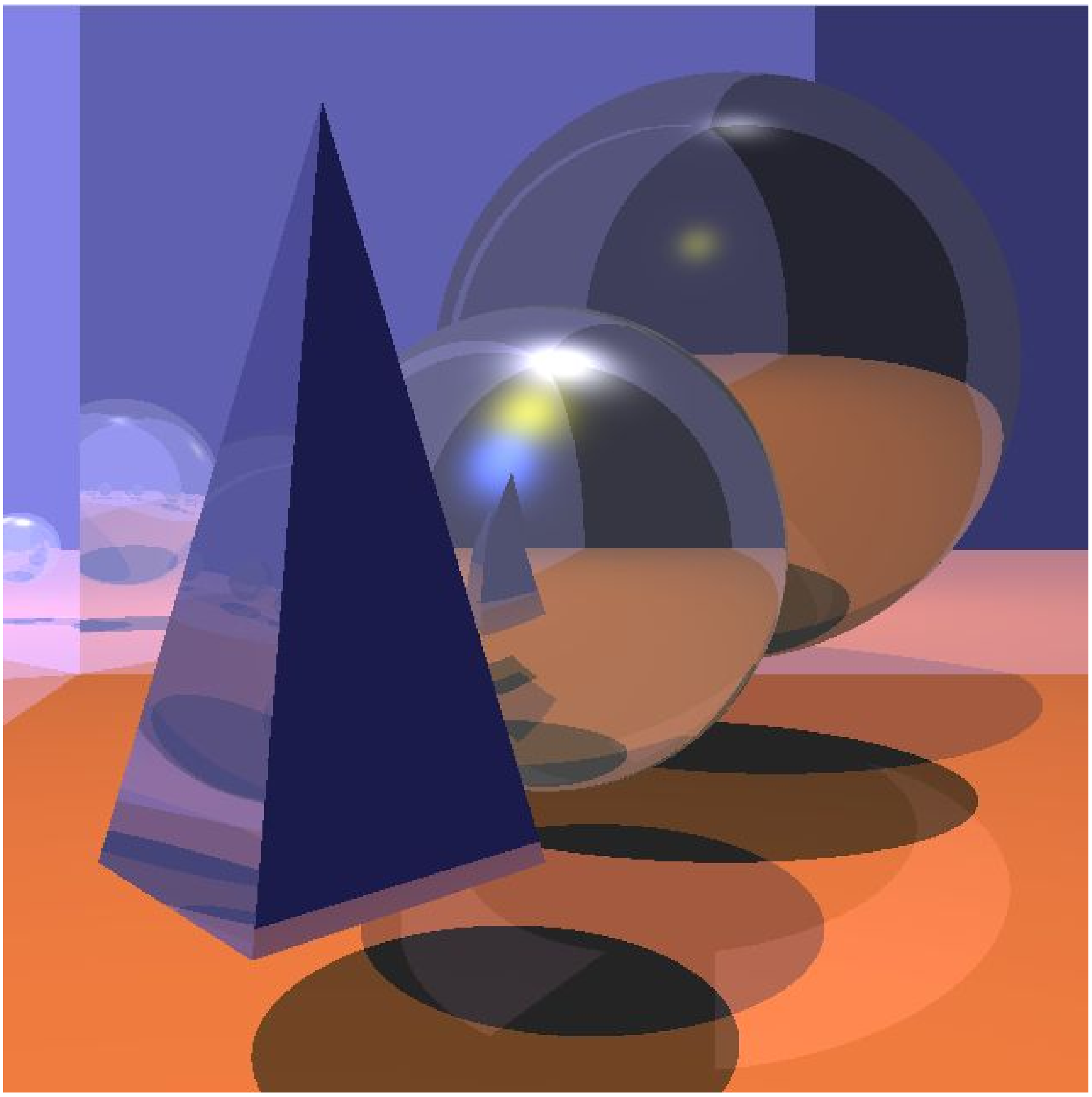
Walls are mirrors

- Render time: 19 seconds
- Bounce limit: 5



Objects are mirrors

- Render time: 34 seconds
- Bounce limit: 5



Refraction

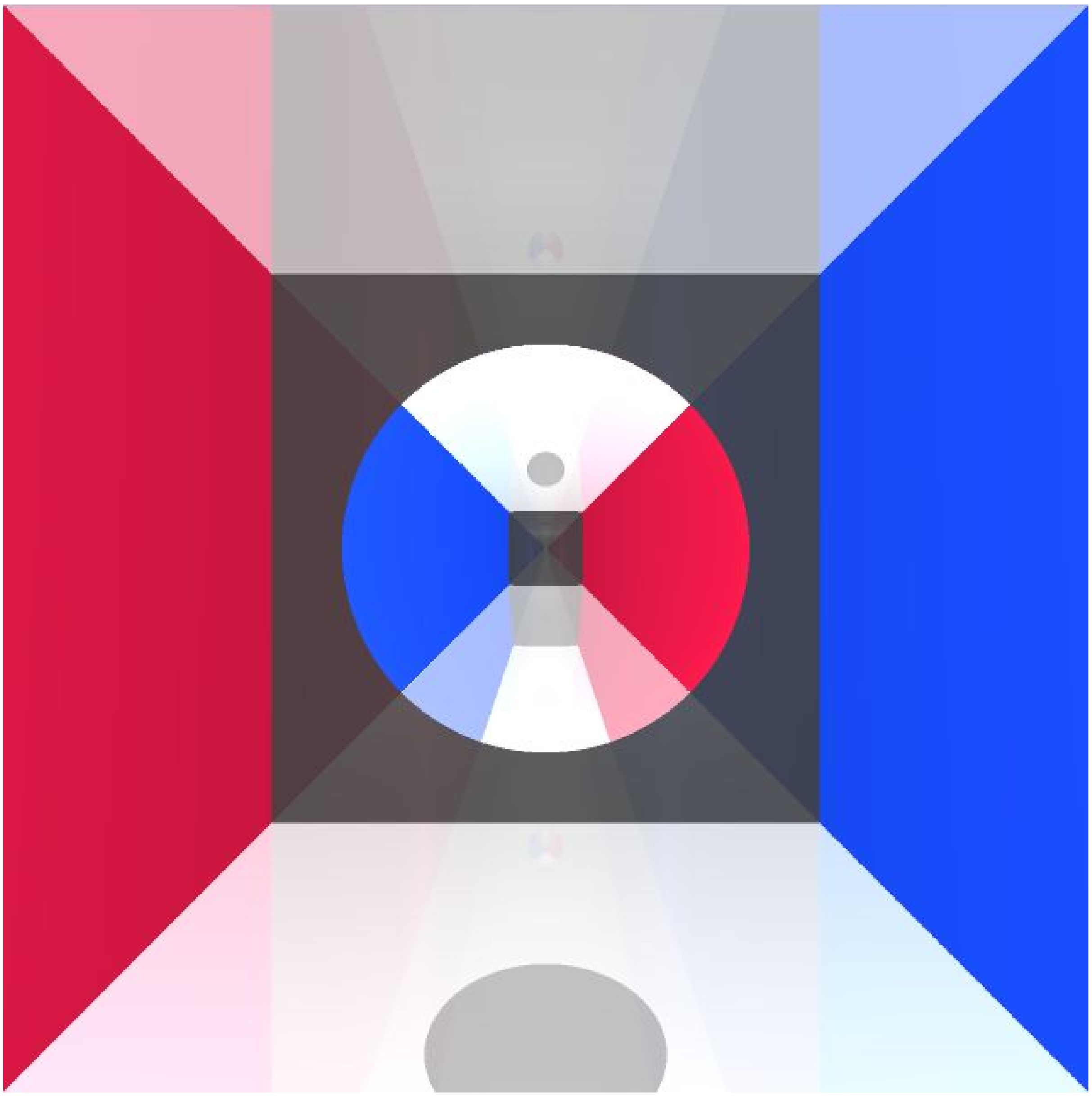
When the ray hits a refractive surface like water or diamond, it refracts.

Usage example (add this for any object):

```
{
        "type": "sphere",
        "radius": 0.6,
        "position": [ 0.0, 0.0, -6.0 ],
        "material": {
                "transmissive": [ 1.0, 1.0, 1.0 ],
                "refraction": 1.95
        }
},
```

MInimalistic upside-down

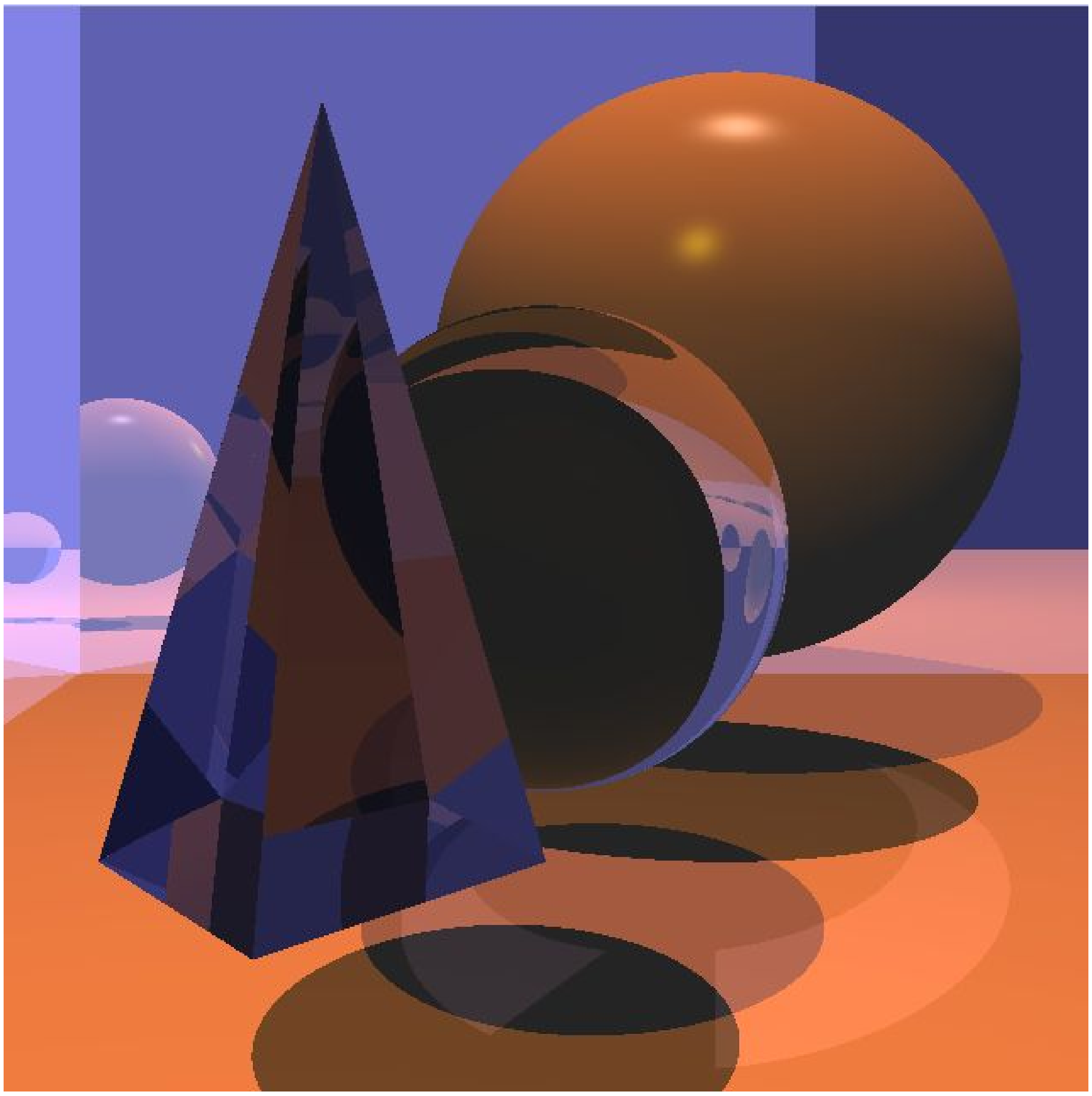
- Antialiasing: SSAA x4
- Render time: 64 sec
- Bounce limit: 5



There is also a special case where the ray would reflect instead of refracting. This is called a total internal reflection. You can observe this effect inside the following pyramid

Total internal reflection

- Render time: 40 sec
- Bounce limit: 5



Acceleration data structure

Bounding Volume Hierarchy (BVH) for meshes significantly improved the rendering speed of very complex objects. I also used the following article to improve the efficiency of my AABB intersection test.

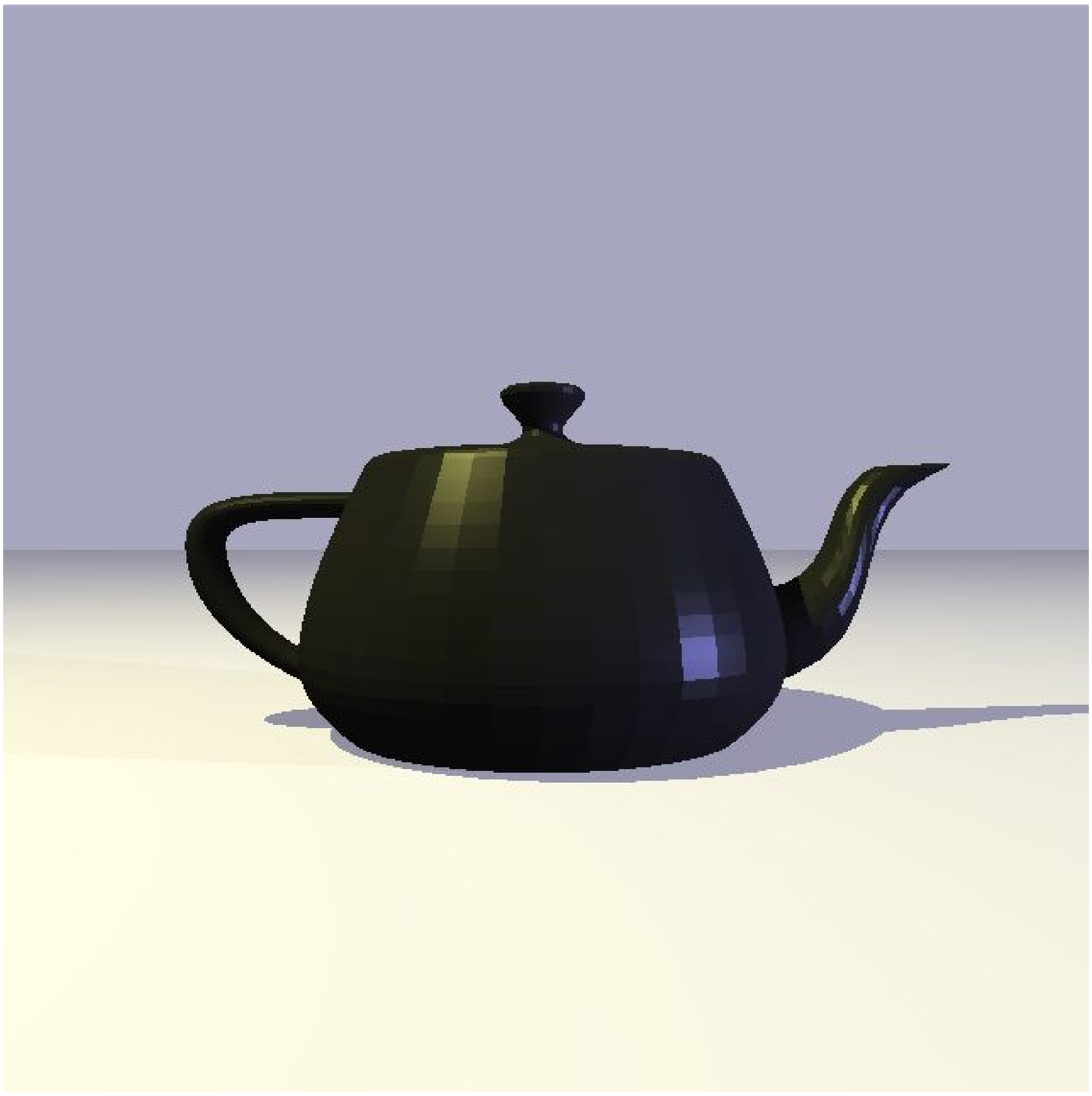
You may use the following script to convert .obj files into a json format

```
srs/utility/obj2json.py
```

Utah teapot

- Triangles count: 6320

- Render time: 317 seconds = 5.3 mins
- Render time (without BVH): approximately 317 * 10 / 60 = ~53 mins



Textures

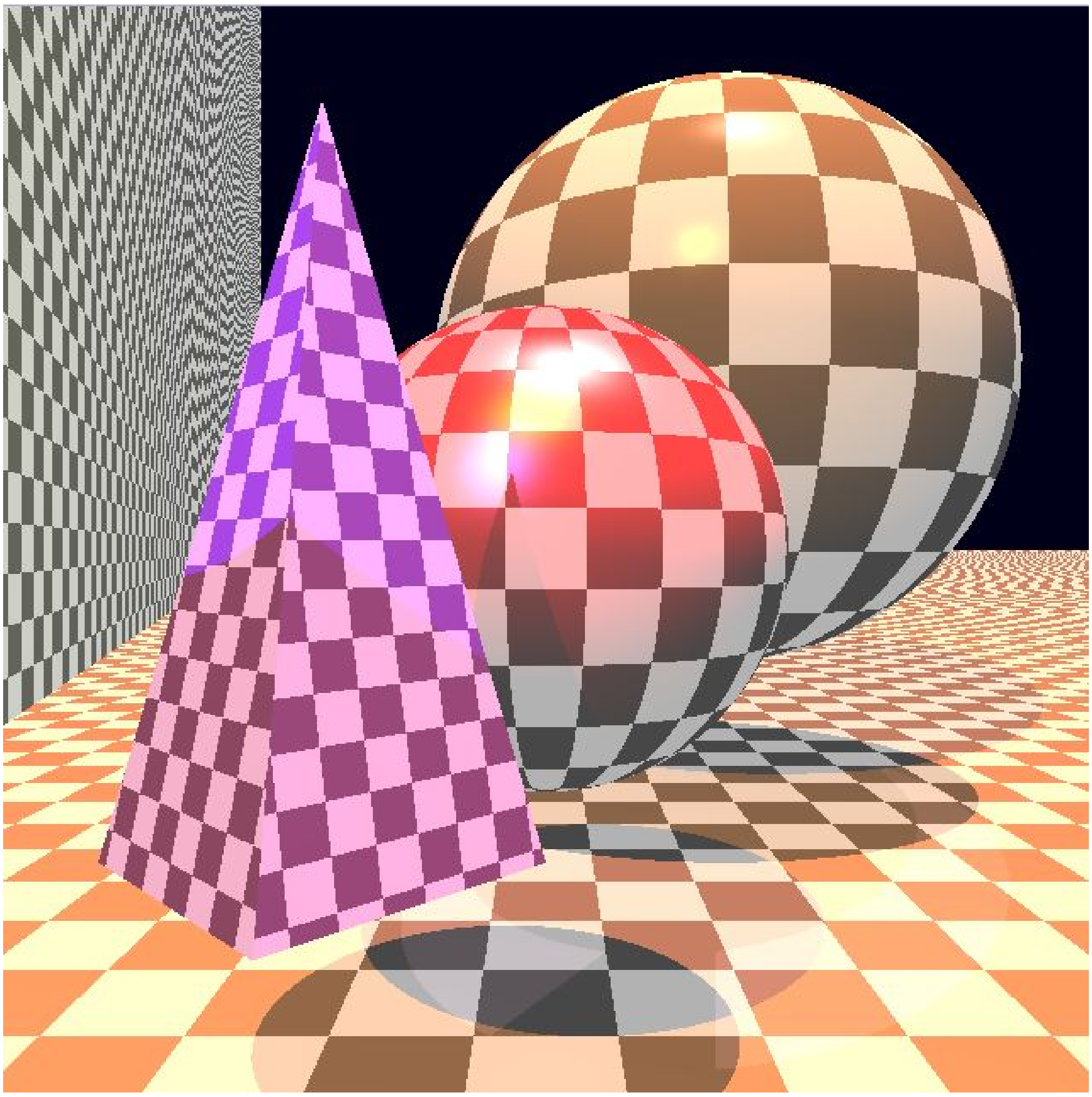
There is an option to turn on **procedural textures** that generate a checkerboard pattern on the fly (e.g after a hit). The following article was very helpful for this feature.

Usage example (add this for any object):

```
"texture": {
        "procedural": "checkers",
        "scale": 12.0
}
```

Checkers

- Render time: 19 seconds
- There is aliasing on the infinite planes. An effective solution might be to add a large bounding box for the whole scene.



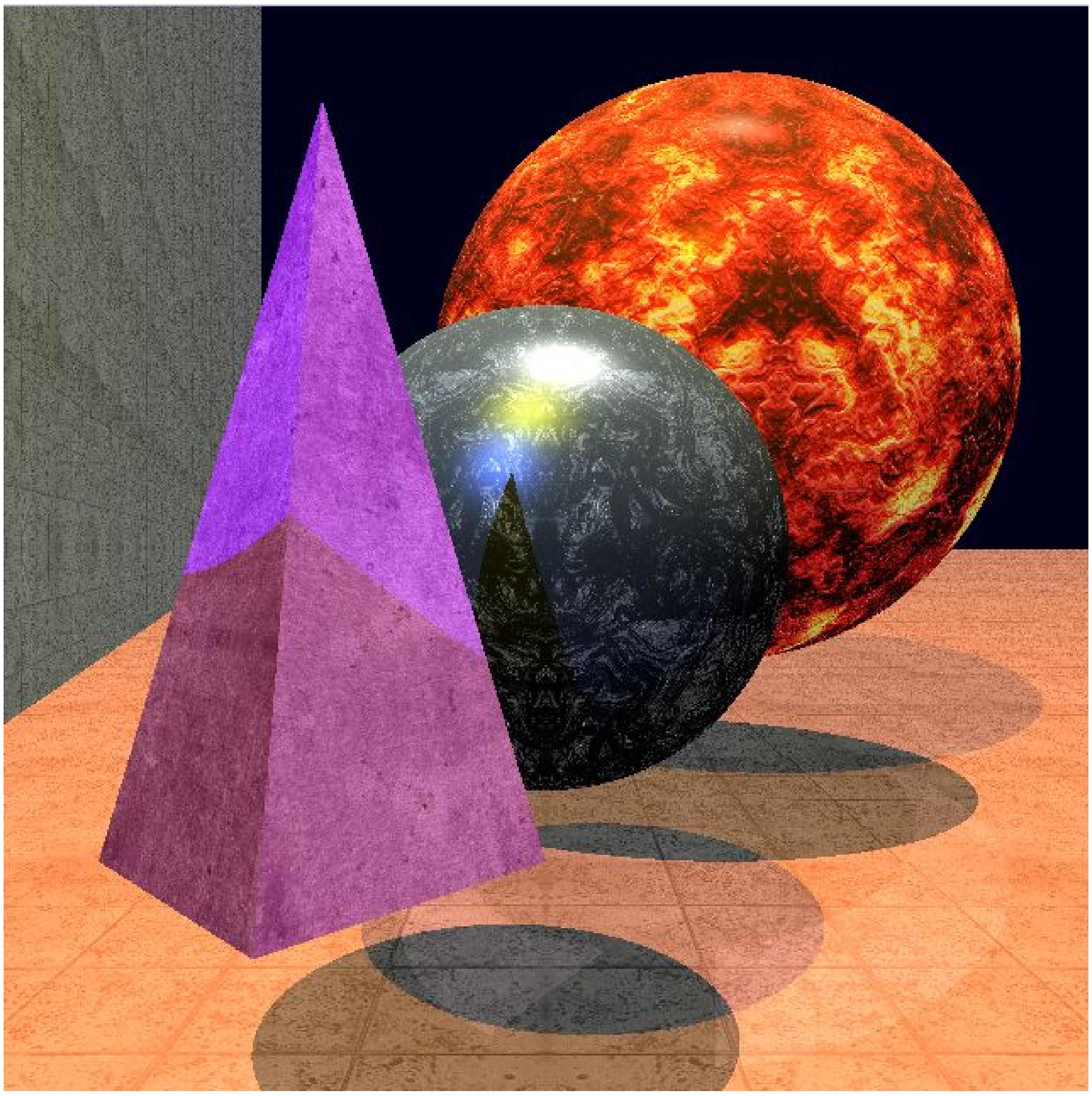
There is also an ability to add custom textures. Note, that these textures should be uniform, since objects do not carry u,v coordinates (this is calculated on demand).

Usage example (add this for any object):

```
"texture": {
        "custom": "obsidian",
        "scale": 2600.0
}
```

Custom texture

- Render time: 19 seconds



Transformations

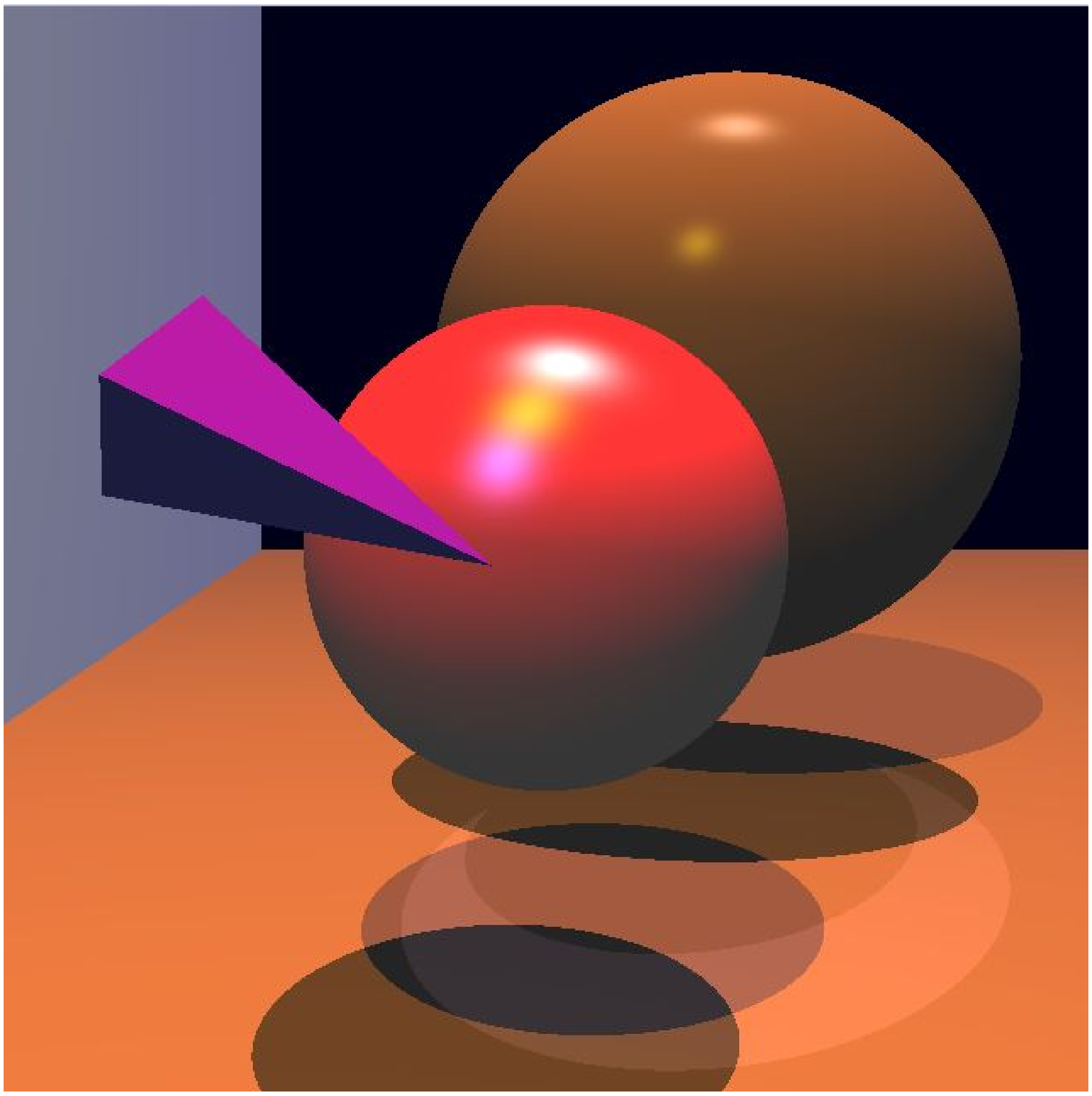
Meshes can be translated, scaled and rotated (using arbitrary axis + angle).

Usage example (add this in the mesh description):

```
"transform": {
        "new_origin": true,
        "scale": 0.5
        "rotation": {
                "axis": [ 0.0, 0.0, 1.0 ],
                "angle": 90.0
        }
}
```

Scaling + Rotation

- Render time: 14 seconds



Constructive solid geometry (CSG)

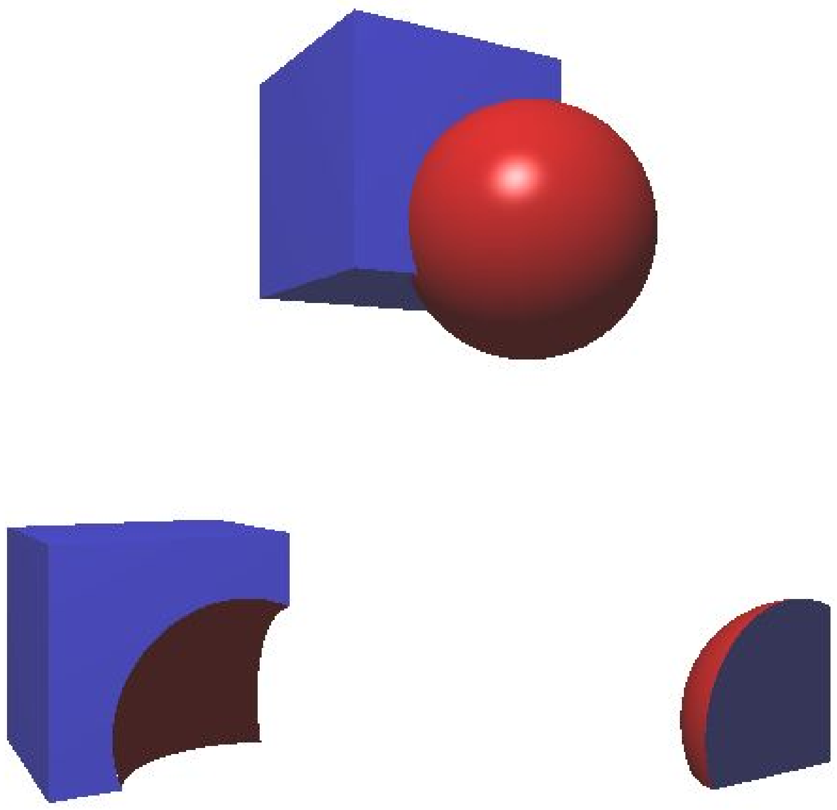
Ray tracing allows us to perform bool operations on the ray. For instance, union is automatically implemented, while intersection is effectively subtract + subtract.

Usage example (add this in the object description):

```
"subtract" : true
```

Blending mode

- render time: 15 sec



Area lighting

This type of light can create more realistic shadows, however there is a cost for that.. For instance, we need to send multiple samples to a random location on the light’s surface. Their average is then used to compute the final colour. I also used a simple heuristic to stop sending shadow rays after a long sequence of misses.

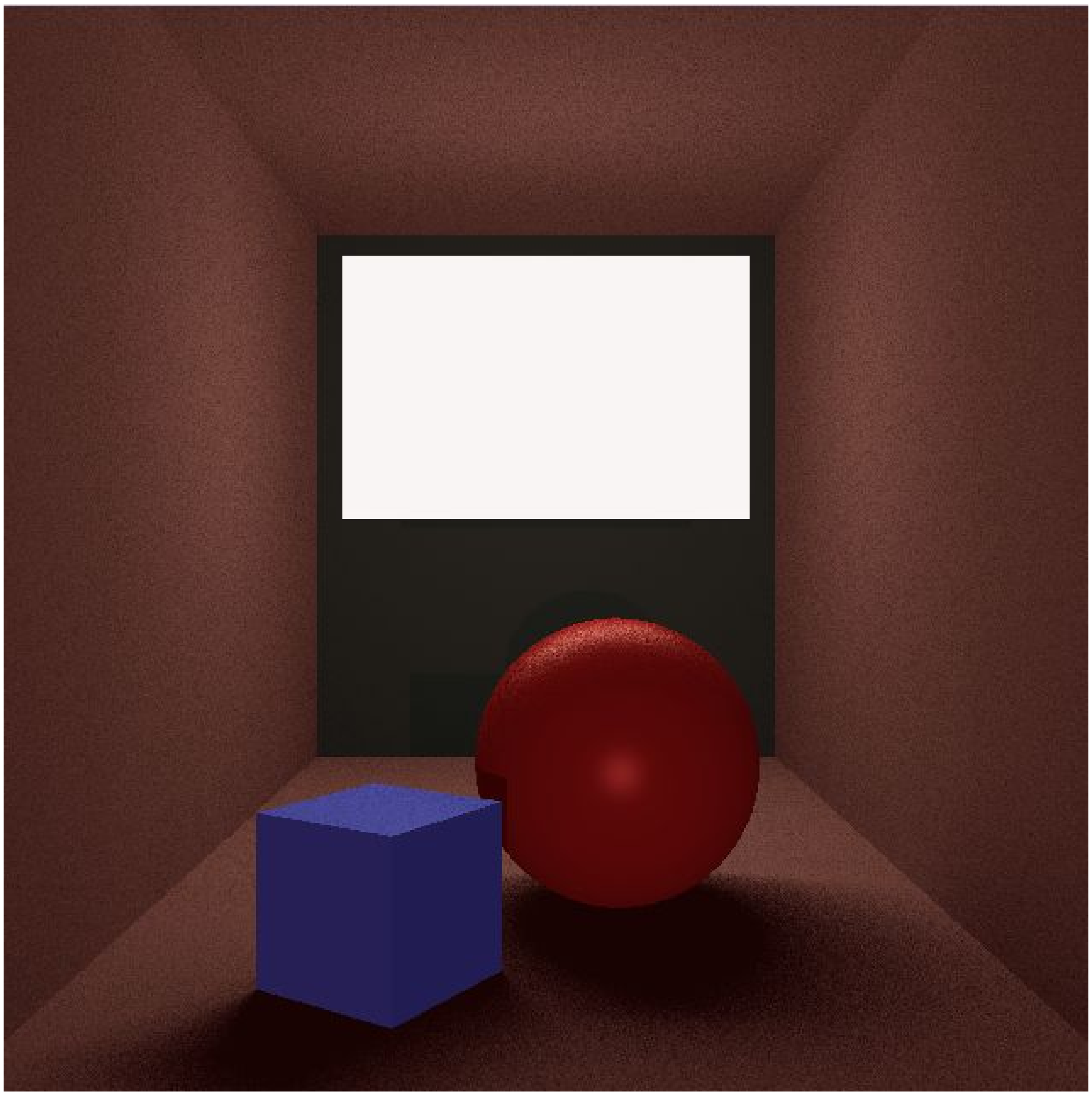
Usage example (add this as a light source):

```
{
        "type": "area",
        "color": [ 1.0,1.0,0.85 ],
        "position": [-3.5, 0.5, -20 ],
        "dirU" : [0,1,0],
        "dirV" : [1,0,0],
        "distU" : 4.5,
        "distV" : 7.0,
}
```

Cinema

- render time: 133 sec
- max number of samples: 20
- longest sequence: 5

},

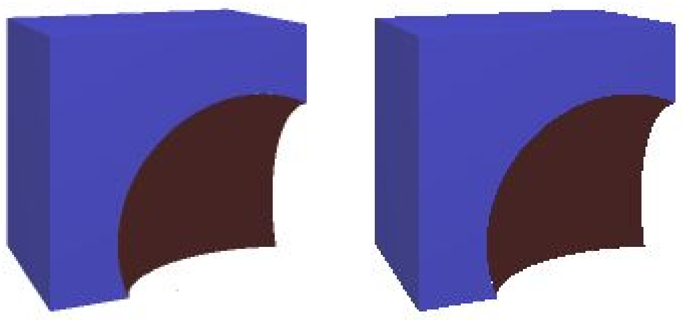


Anti-Aliasing

Image quality can also be improved using Supersampling Anti-Aliasing (SSAA) x4. This is effectively rendering the scene at higher resolution and then compressing it into a desirable resolution.

Usage example (add this in the camera description):

```
"camera": {
   "field":     60,
   "background": [0, 0, 0.1],
   "antialiasing" : false
},
```



Ray Debug

If you click a pixel on the scene, you will receive detailed information about the ray's path. This might be especially helpful for debugging refractions and CSG operations.

```
CASTING A RAY

Triangle HIT from [Outside] @ RayLen = 2.172172,
Refracting AIR --> MATERIAL

Triangle HIT from [Inside] @ RayLen = 0.129709,
Total internal reflection ...

Triangle HIT from [Inside] @ RayLen = 0.214013,
Refracting MATERIAL --> AIR

Plane HIT from [Outside] @ RayLen = 4.186393,
```

External Libraries

- EasyBMP - a library to manage `.bmp` files.
- STB ImageWrite - a library to save rendered images in a `.png`.

PC specs

The program was successfully tested using:

- Processor: Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz
- RAM: 8.00GB
- OS: 64-bit Windows 10
- IDE: Visual Studio 2019 16.4

- Graphics card: AMD Radeon HD 6700M Series
    - Memory size 1024 MB
    - Memory type GDDR5
    - Total Memory Bandwidth 51.2 GBytes/s