Path Tracer

This time, the Path Tracing brute force algorithm was implemented. To do so, there's a fixed number of samples with random directions taken from each pixel and each ray makes a set number of reflections, also in random directions. The **diffuse BRDF** class was implemented and now all primitives have a BRDF as its attribute.

Below we can see the new loop inside the **integrate function**:

```
// Loops over image columns
for ( std::size_t x = 0; x < buffer_.h_resolution_; x++ )
{

    for ( int samples = 0; samples < nsamples; ++samples )
    {
        float r1 = dist(rng);
        while (r1 == 1.0f)
            r1 = dist(rng);

        float r2 = dist(rng);
        while (r2 == 1.0f)
            r2 = dist(rng);

        Ray ray{ camera_.getWorldSpaceRay( glm::vec2{ x + r1, y + r2 } ) };

        buffer_.buffer_data_[x][y] += L(ray, 0);
    }

    buffer_.buffer_data_[x][y] /= nsamples;

}
```

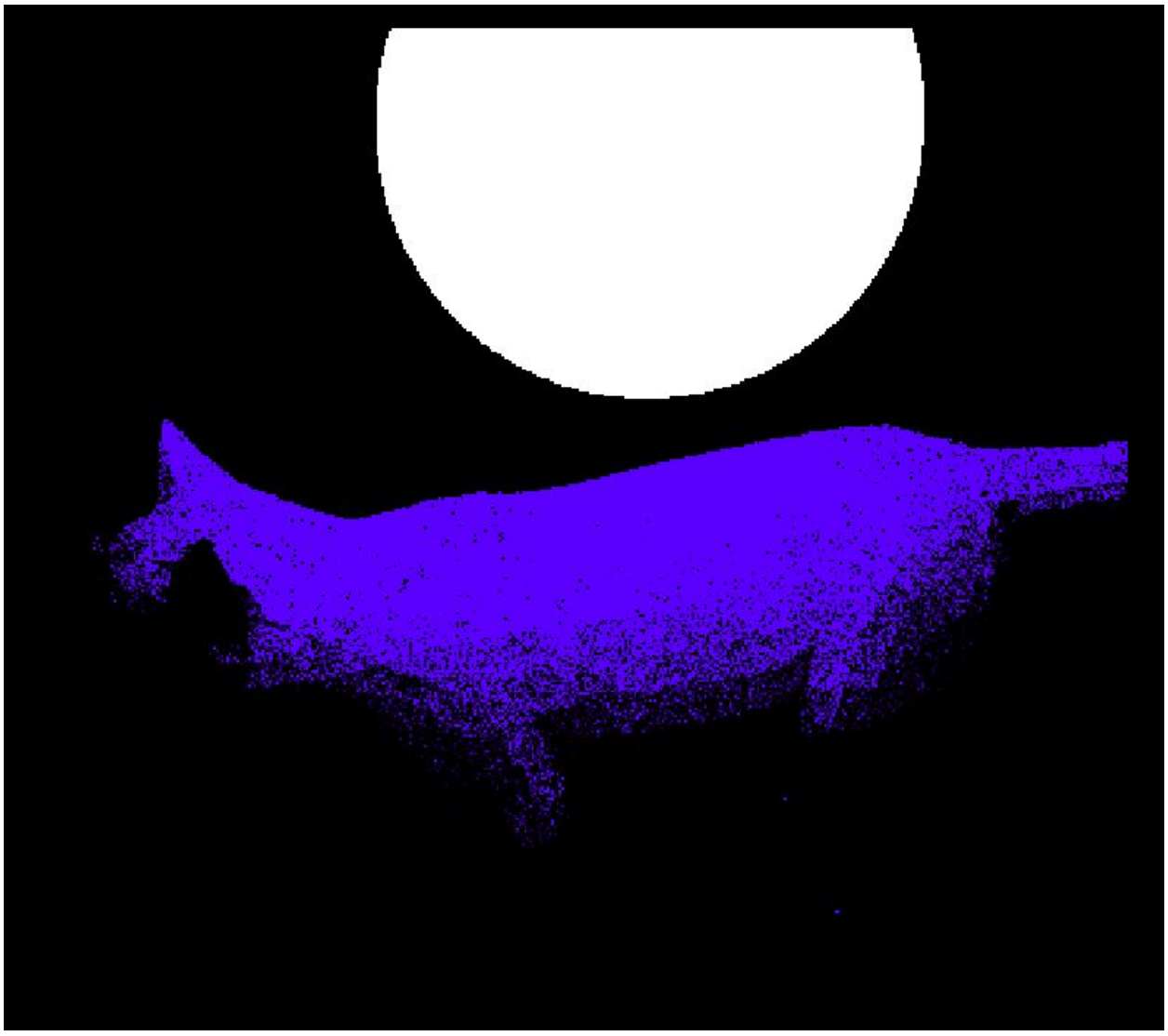
part of the integrate function

We can see that now we use an **L function** to determine the **buffer_data_**. The function implementation is shown below:

```
glm::vec3 PathTracer::L( Ray &ray, int curr_depth )
{
    IntersectionRecord intersection_record;
    intersection_record.t_ = std::numeric_limits<double>::max();

    glm::vec3 Lo( 0, 0, 0 );

    if (curr_depth < 5) {
        if (scene_.intersect(ray, intersection_record)) {

            float r1 = dist(rng);
            while (r1 == 1.0f)
                r1 = dist(rng);

            float r2 = dist(rng);
            while (r2 == 1.0f)
                r2 = dist(rng);

            float theta = acos(1 - r1);
            float phi = 2 * ((float)M_PI) * r2;
            float radius = 1;

            float x = radius * sin(theta) * cos(phi);
            float y = radius * sin(theta) * sin(phi);
            float z = radius * cos(theta);

            glm::vec3 dir(x, y, z);
            ONB onb;
            onb.setFromV(intersection_record.normal_);
            dir = glm::normalize(dir * onb.getBasisMatrix());

            Ray refl_ray(intersection_record.position_, dir);
            float dot = glm::dot(refl_ray.direction_, intersection_record.normal_);

            if (dot < 0) {
                refl_ray.direction_ *= -1;
                dot = -dot;
            }

            Lo = intersection_record.material_.emission_ + 2.0f * ((float)M_PI) *
                 intersection_record.material_.brdf_.fr() * L(refl_ray, ++curr_depth) * dot;
        }
    }
    return Lo;
}
```

L function

With all of this it was possible to render the image of a cat obj, made of a fair amount of primitives, with a luminous sphere above it. The path tracer was set to use 50 samples and the rays reflected 5 times each. It took around 10 minutes:



rendered image