Refraction caustics in forward path tracing

Shawn Halayka*

Sunday 3rd September, 2023 20:54

Abstract

In this paper we introduce a method for producting refraction and reflection caustics in forward path tracing. These caustics do not rely on a light location, and as such, do not rely on bidirectional or backward path tracing. Thus, these caustics allow for as many light sources as one would care for; the shapes and positions of the lights are completely arbitrary (can be bunny shaped, etc). We use the standard Cornell box for testing the path tracer.

1 Rasterizer versus ray tracing versus path tracing

Here we rate the three main visualization algorithms. We rate them in terms of what is relatively easy, or naturally occurring, in each of the visualization algorithm.

Algorithm	Reflections	Shadows	Refraction caustics	Reflection caustics
Rasterizer	No	No	No	No
Ray tracer	Yes	Hard	No	No
Forward path tracer	Yes	Hard / soft	Yes	Yes

Clearly, the most suitable visualization algorithm for now (and in the future) is the forward path tracer. This not to say that, for instance in the case of the rasterizer, there is no such thing as shadows. It's just that the implementation of shadows is a kludge, or not naturally occurring.

2 Path tracer code

A full Vulkan code exists [1], using a very simple (e.g. not physically accurate) light transport algorithm. The results given here can only get better with the improvement of the light transport algorithm. This code is based off of Sascha Willems' work [2, 3].

3 Acknowledgement

The Cornell boxes used in this paper were developed by Rob Rau.

*sjhalayka@gmail.com

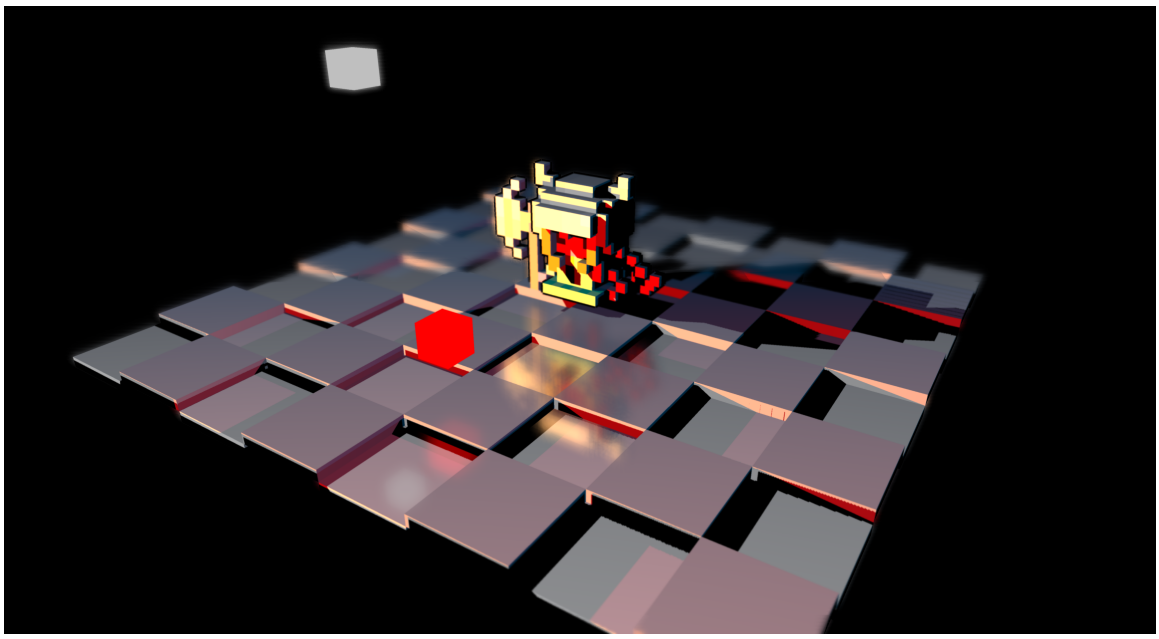

Figure 1: Standard rasterizer. The surface is lit using Phong shading and omnidirectional shadow maps. The reflections are faked – the camera is flipped upside down, and reflected things masked and drawn. In other words, reflections and shadows are not naturally occurring. Global illumination is not taken into account. This knight model, as well as MagicaVoxel, were made by the Twitter user @ephtracy.

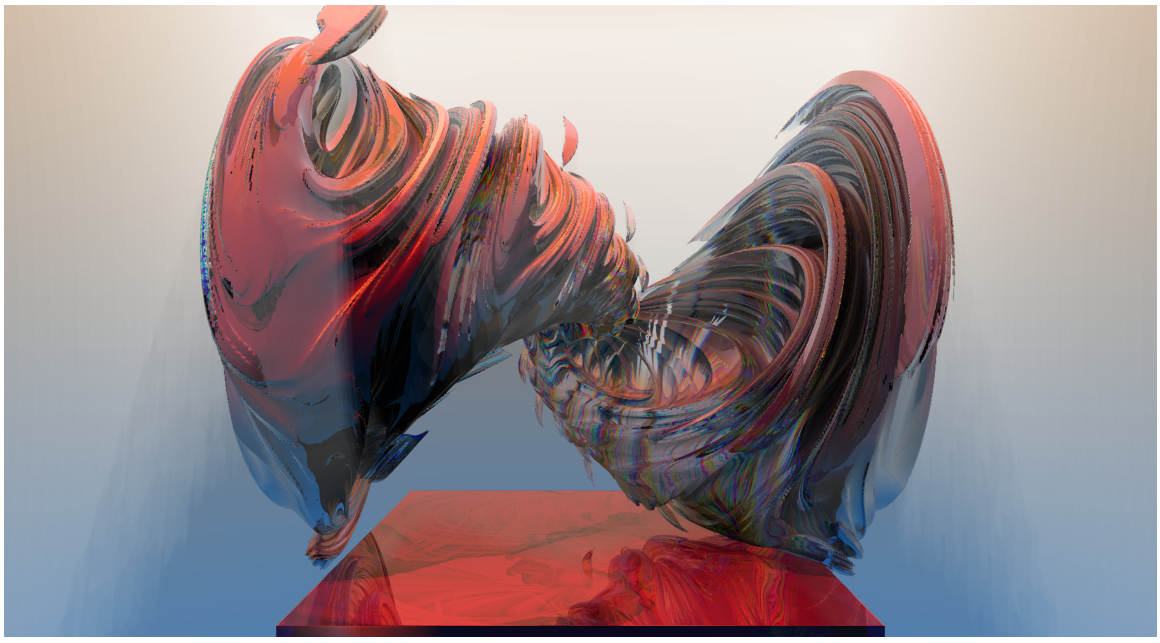

Figure 2: Standard ray tracer, taking into account transparency. The surface is lit using Phong shading and sharp shadows. In other words, reflections and shadows are naturally occurring. Global illumination is not taken into account.

```
vec3 hitPos = o + d * rayPayload.distance;

// If partially transparent
if(rayPayload.opacity != 1.0)
{
    // Incoming ray
    if(dot(d, rayPayload.normal) <= 0.0)
    {
        o = hitPos.xyz - rayPayload.normal * 0.01;
        d = refract(d, rayPayload.normal, eta);
    }
    else // Outgoing ray
    {
        vec3 temp_dir = refract(d, -rayPayload.normal, 1.0/eta);

        if(temp_dir != vec3(0.0))
        {
            o = hitPos.xyz + rayPayload.normal * 0.01;
            d = temp_dir;
        }
        else
        {
            // Total internal reflection
            o = hitPos.xyz - rayPayload.normal * 0.01;
            d = reflect(d, -rayPayload.normal);
        }
    }
}
else // Fully opaque
{
    o = hitPos + rayPayload.normal*0.01;
    vec3 d_cos = cos_weighted(rayPayload.normal, prng_state);
    d = mix(d_cos, reflect(d, rayPayload.normal), rayPayload.reflector);
}
```

Figure 3: Taking transparent objects into consideration. In essence, instead of always producing a pseudorandom cosine-weighted reflection vector, refraction occurs for transparent objects.

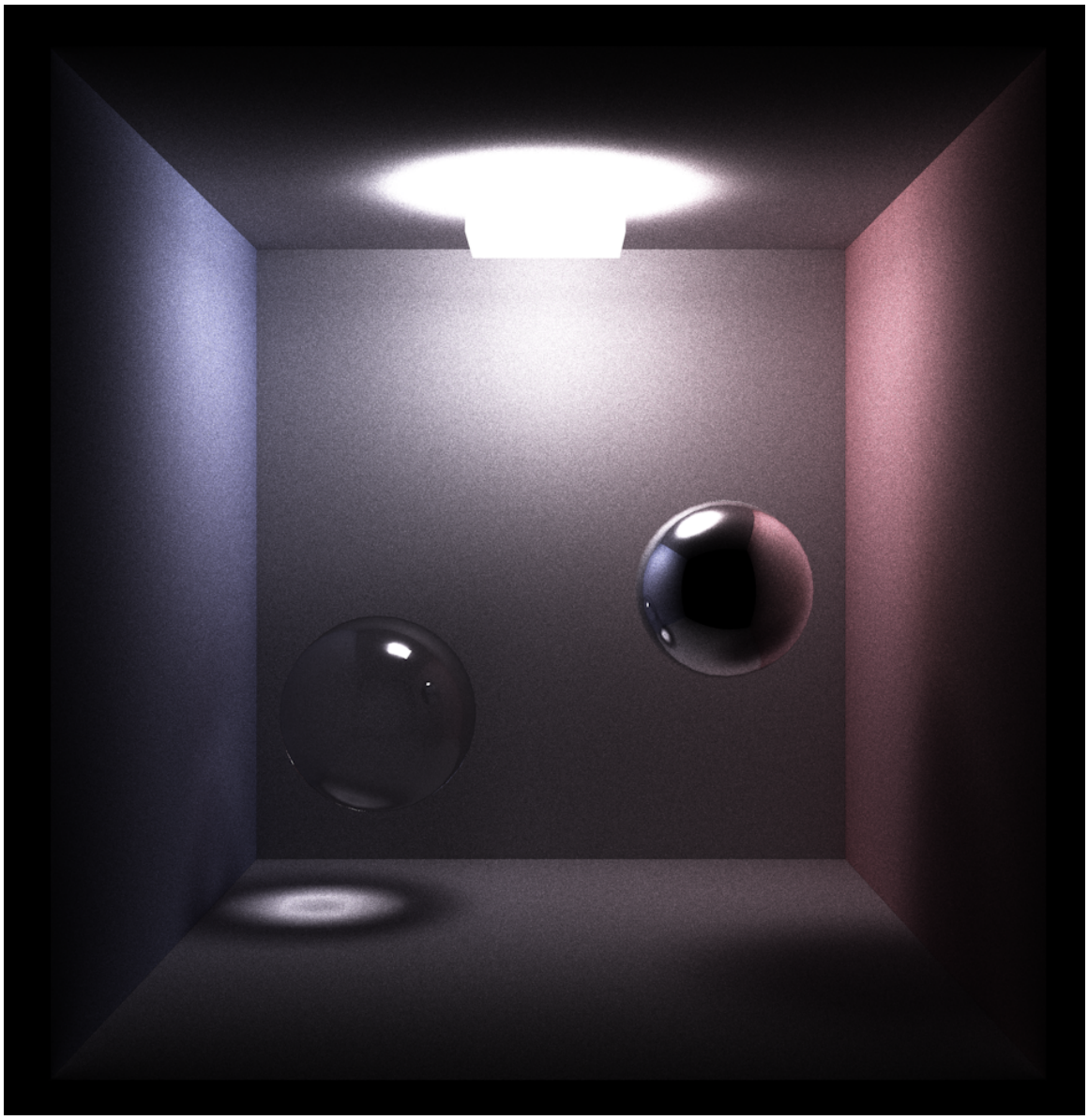

Figure 4: Standard path tracer, taking transparent objects into consideration. Note the naturally occurring refraction caustic, without chromatic aberration. Note the soft shadows. Global illumination is taken into account.

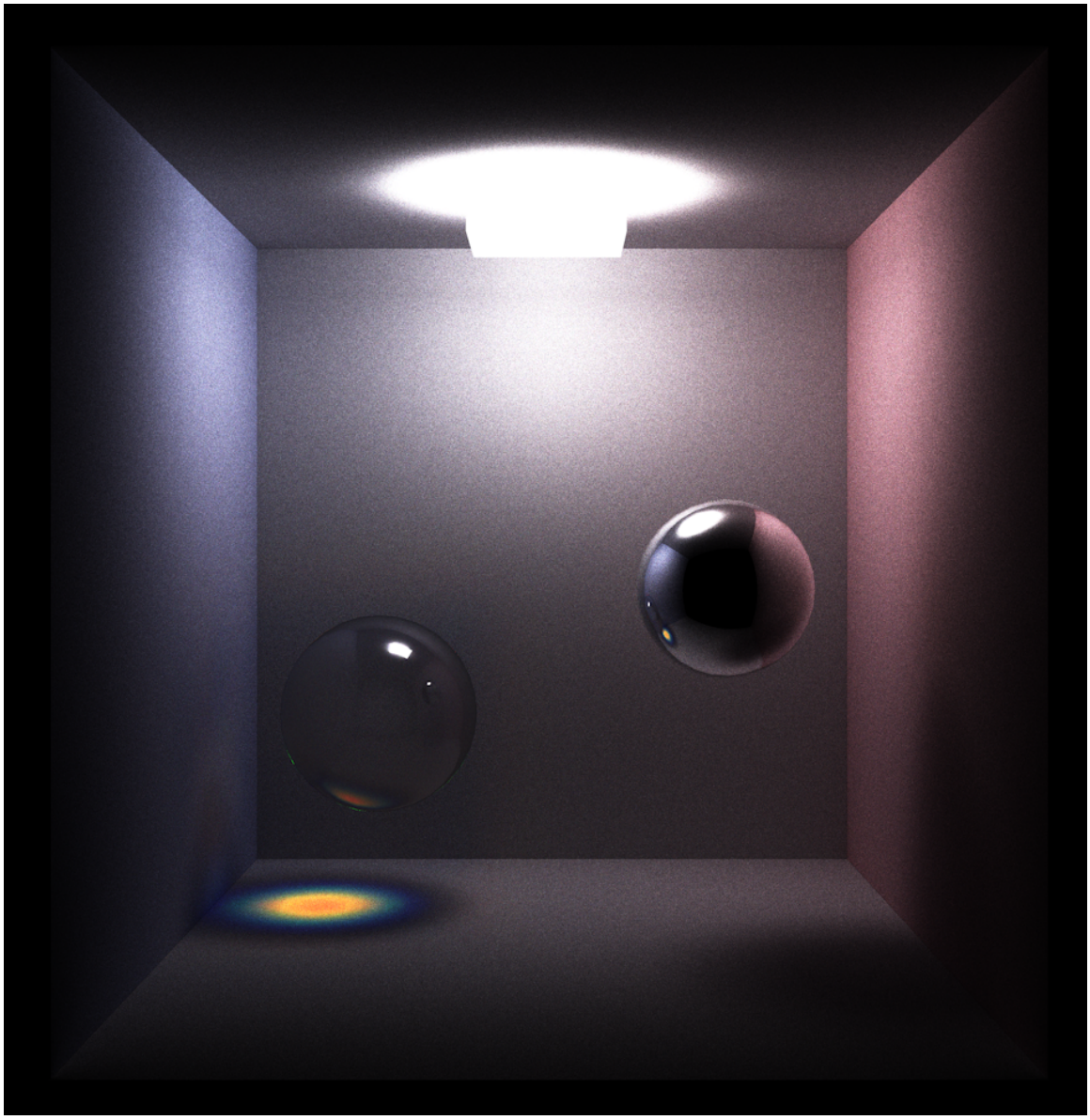

Figure 5: Note the refraction caustic, with chromatic aberration.

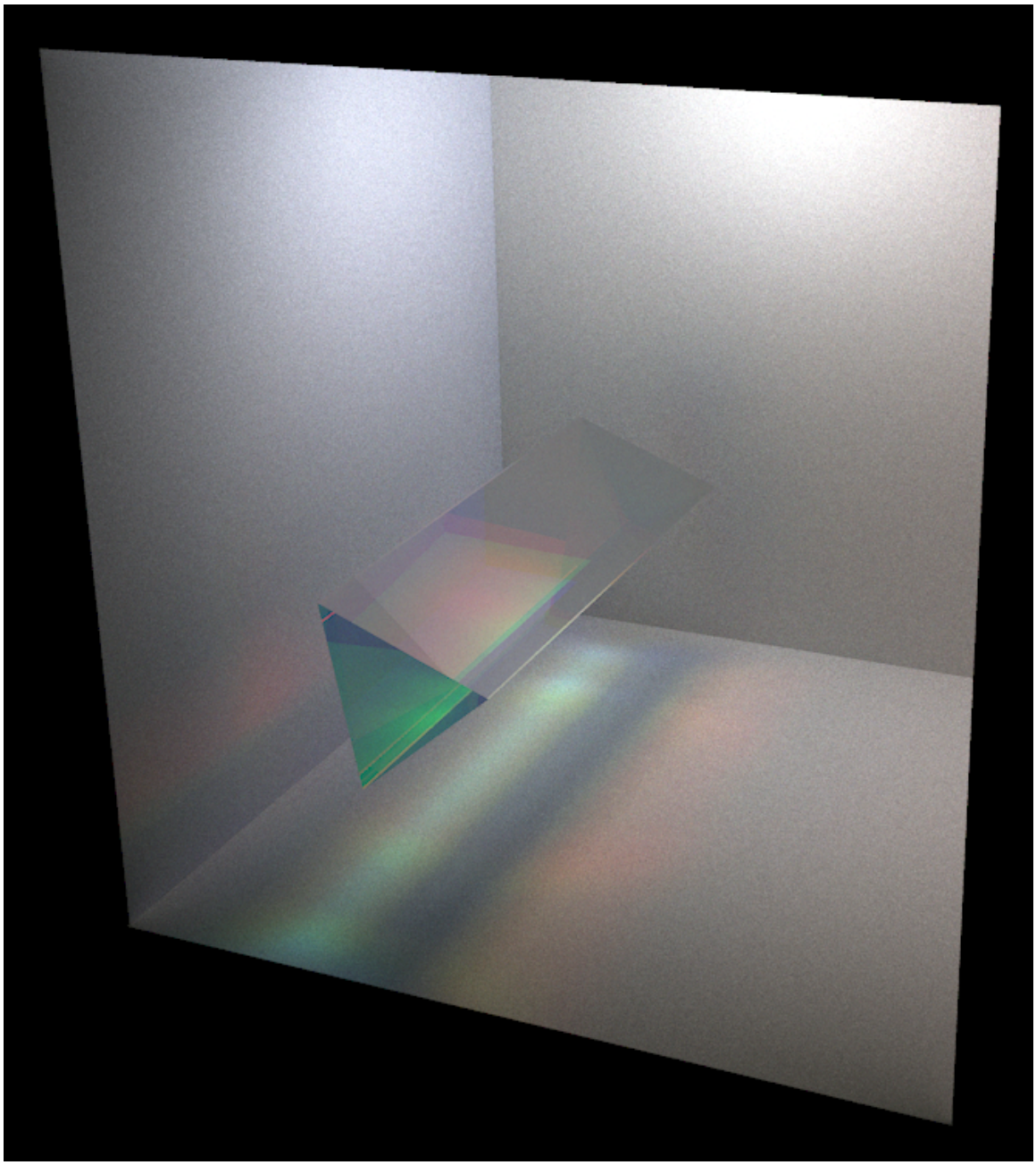

Figure 6: Note the refraction caustic, with chromatic aberration.

References

[1] Halayka. Vulkan code. https://github.com/sjhalayka/cornell_box_textured

[2] Willems. Path tracer code. https://github.com/SaschaWillems/VulkanPathTracer

[3] Willems. Vulkan demo codes. https://github.com/SaschaWillems/Vulkan