Raytracer with spatiotemporal reprojection

Martin Jakobsson*

Lund University
Sweden

Abstract

A raytracer has been implemented together with two denoising techniques: multiple samples per pixel and accumulation. To compensate for camera movement and rotation, spatiotemporal reprojection was used in the accumulator. The compensation worked well for diffuse surfaces, but not as well for reflective and refractive surfaces. The accumulation technique was cheaper than the use of multiple samples per pixel. The application was able to run in real-time with reasonable quality.

1 Introduction

In computer graphics rasterization is the most common rendering technique for real-time applications. There are however limitations to rasterization making it less suitable for rendering objects with reflections and refractions as well as global illumination. Raytracing is an alternative that doesn't have these limitations. There are however a few problems with raytracing, one of which being the rendering time needed to reach a good quality image. For a real-time application the rendering time allowed is limited, resulting in lower quality and a noisy image. The noise can be reduced with different denoising techniques, one of which is accumulation. To compensate for camera movement spatiotemporal reprojection can be used. These are the techniques implemented in this paper.

2 Application

The program uses raytracing to draw a simple scene. The output from the raytracer is then used in an accumulator, which can be seen in figure 1. The accumulator uses the data from the raytracer together with data from the previous frame to determine the final color. To adjust for camera movements spatiotemporal reprojection is used (more on that in section 2.4). The raytracer outputs both the color of the light entering the camera, as well as the distance to the closest object. This data is needed in the reprojection step in the accumulator.

2.1 Raytracer

The raytracer runs entirely on the GPU in a fragment shader written in GLSL. The scene to be rendered is hard coded into the shader and contains three spheres, a plane and a spherical light source. When a ray hits an object the ray will bounce in one of the following ways: diffuse reflection, specular reflection or refraction. Much of this code is based on an article called *Ray Tracing in One Weekend* [Shirley 2020]. More specifically the diffuse reflection is based on section 8.1, the specular reflection is based on section 9.6 and the refraction is partly based on section 10.

While the ray bounces around in the scene it remembers a color multiplier accumulated from the previous bounces. This multiplier can be seen as an RGB color filter, and is used when the ray hits a light or any other light emitter. The light color will then be multiplied by the multiplier before contributing to the pixel color. This is what gives color to the objects.

There is also a possibility to make refractive materials absorb light, which means that the color multiplier described above is affected by different amounts depending on the distance traveled through a medium. The color multiplier is multiplied by α^d where $0 \leq \alpha \leq 1$ and d is the distance traveled since the last bounce. Each object has an α value for each RGB component.

The physics implementation described above is not completely correct, but it is good enough for this demonstration.

For each pixel the ray performs at most 8 bounces and outputs the final pixel color as well as the distance to the first object encountered by the ray. The output (color) can be seen in figure 2.

*e-mail: ma5210ja-s@student.lu.se

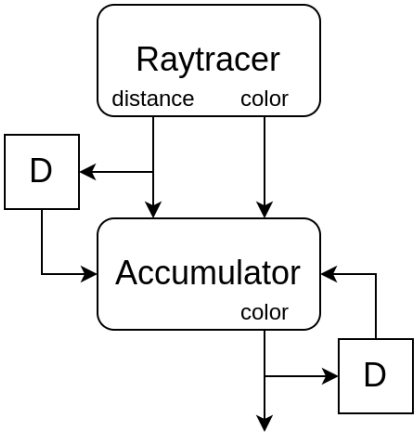

Figure 1: The dataflow between the raytracer and the accumulator. The D-boxes delay the input one frame.

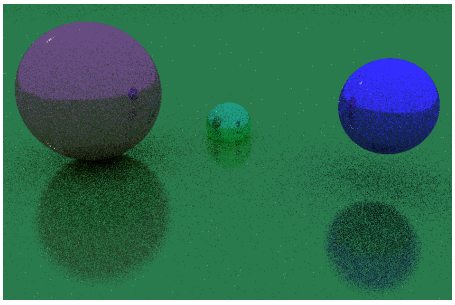

Figure 2: The output of the raytracer with 1 spp.

2.2 Denoising

If the raytracer output is used as is, the resulting image will contain large amounts of noise. This is mostly due to the random nature of diffuse reflection. To make the image less noisy the ray tracer is run several times and the resulting images are averaged to get the final image. To further reduce the noise, each run of the ray tracer uses a different point within the pixel. This can be seen as performing anti-aliasing. The resulting image can be seen in figure 3 where 8 samples per pixel were used.

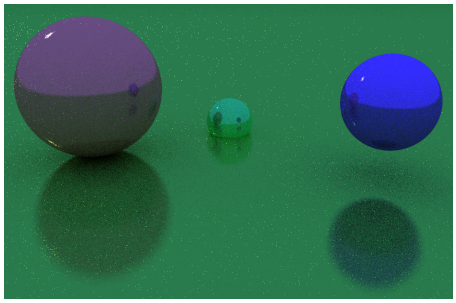

Figure 3: The output of the raytracer with 8 spp.

2.3 Accumulator

Using multiple samples per pixel does decrease the noise, but has diminishing returns, which makes it too computationally expensive for a real-time application. Another way of reducing the noise is to reuse the images from the previous frames. This can be done by using e.g. 90% of the previous frame and adding 10% of the new frame for each pixel. The resulting image can be seen in figure 4.

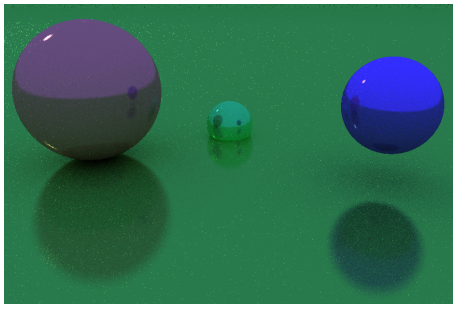

Figure 4: The output of the accumulator with 1 spp.

This simple solution works very well as long as the objects are static and the camera doesn't move. Since this is a real-time application the camera could certainly move, which results in severe visual artifacts as seen in figure 5. This movement can be adjusted for using spatiotemporal reprojection.

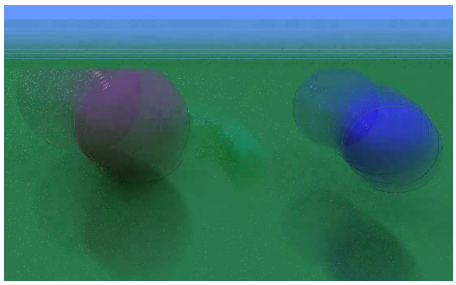

Figure 5: Blur caused by camera rotation if no compensation is performed.

2.4 Spatiotemporal reprojection

When the accumulator adds the new pixel value to the value from the last frame it uses the same pixel coordinates for the new and the old data. To adjust for the camera movement we need to find the coordinates in the old image that correspond to the same point in space as the current pixel in the new image. This can be seen in figure 6 where the world is seen from above and the blue lines end up at different pixels depending on the camera.

To find these coordinates we first need to find the point in space for the object, i.e. the point where the blue ray hits the green oval in figure 6. The direction can be calculated using the inverse view-projection matrix for the new camera, and together with the distance (from the ray tracer) the ray can be extended to the appropriate length. This gives $P = C_{new} + vt$ where C_{new} is the position of the new camera. The world position of the object can then be transformed using the view-projection matrix of the old camera. This gives the screen coordinates for the old camera.

There is one slight problem left to address: as seen in the figure the red line finds a point that was hidden from the old camera. This means that the accumulator will use the color of the small oval instead of the big oval. Since the correct color is not present in the old image the accumulator should simply ignore it and only use the value from the ray tracer.

To find out that the object is being obscured we calculate the distance to the object from the old camera and compare it with the previous distance value from the raytracer (see figure 1). The distance to the point is simply calculated as the distance between the point and the old camera position. If these distances differ too much the accumulator only uses the output from the raytracer.

The implementation of the spatiotemporal reprojection technique was inspired by an implementation on Shadertoy [Raxvan 2016], specifically the mainImage method in Buffer B where the actual algorithm is implemented.

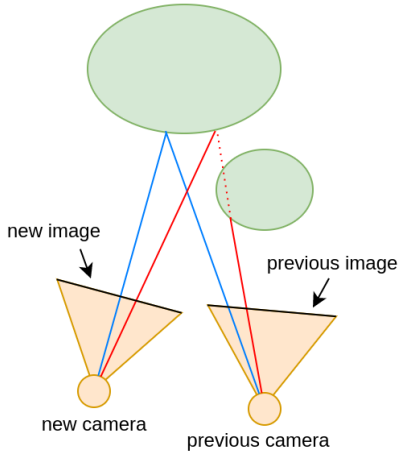

Figure 6: Spatiotemporal reprojection

3 Results

The final results can be seen in figure 7 where the accumulation technique is combined with multiple samples per pixel.

As stated before, the raytracer can handle refractions and reflections, which can be seen in figure 8 where the ground has been replaced with a checkerboard pattern and the camera is looking through a slightly blue glass sphere. A green sphere and a blue sphere can be seen through the glass, where the blue sphere is partially refractive. This image was generated with the same quality settings as figure 7.

The performance of the raytracer depends on the number of samples per pixel as well as the window size. With 8 spp and a 720p window size the application ran at several frames per second on an integrated graphics card (Intel HD Graphics 520), which would likely be much higher for new GPUs and even higher for discrete

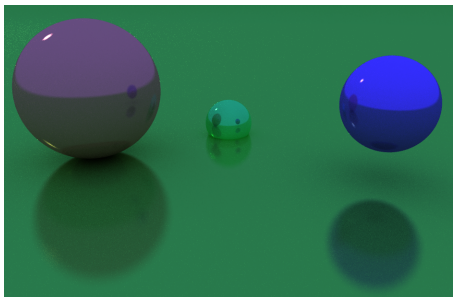

Figure 7: The output of the accumulator with 8 spp.

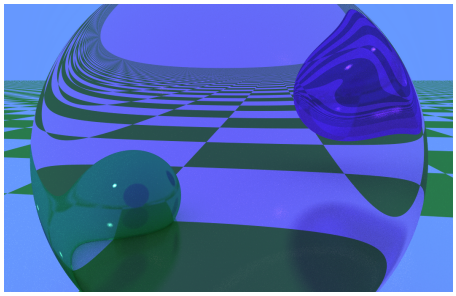

Figure 8: Refraction through tinted glass with a checkerboard ground. The sphere to the right is partially refractive.

GPUs. With fewer samples per pixel the performance was greatly increased.

4 Discussion

The quality of the image in figure 7 is noticeably better compared to only using one of the techniques. This increase in image quality comes at a price: it takes more time to render a frame. When using 8 spp the scene is drawn 8 times to get one frame, so the rendering time is expected to be 8 times higher. The accumulation technique, on the other hand, only needs one pass of the raytracer and an extra pass for the accumulation, which makes the accumulation technique much cheaper.

There is however a downside to the accumulation technique: the image can potentially contain visual artifacts from old frames. This problem is greatly reduced with spatiotemporal reprojection, but is not completely removed.

It should also be noted that there are several possible improvements to this raytracer. One possible improvement is the following. When the accumulator cannot use the data from the previous frame it will only use the new raytracer output. This makes the regions close to the edges of the objects noisy, which could be compensated by running the raytracer again for those regions. This would make sure that all parts of the image are based on data from multiple passes of the raytracer.

4.1 Limitations

As previously stated, spatiotemporal reprojection doesn't work well with reflections and refractions. The reason is that it assumes that all objects will look the same regardless of the viewing angle. This is true for diffuse materials and objects with textures, but not for specular reflections and refractions. This can be partly compensated for with a more advanced technique, but not with the technique implemented in this application. This makes reflections and refractions a major limitation to the spatiotemporal reprojection technique.

5 Conclusion

The accumulation technique combined with spatiotemporal reprojection is able to produce an image with the same quality as an image created using multiple samples per pixel, but with higher performance, making it suitable for real-time applications.

References

RAXVAN, 2016. Temporal reprojection, August. https://www.shadertoy.com/view/ldtGWl.

SHIRLEY, P., 2020. Ray tracing in one weekend, December. https://raytracing.github.io/books/RayTracingInOneWeekend.html.