POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

The application of ray tracing to efficiently simulate fisheye lenses.

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Simone Abelli

Advisor: Prof. Marco Gribaudo

Academic year: 2022-2023

1. Introduction

In computer graphics, traditional rendering techniques are based on rasterization, a process that turns 3d geometries into 2d images, assuming that their basic primitives - i.e. the triangles - will not change their shape. For this reason they struggle in trying to simulate the visual distortion caused by some lenses. This is the case of fisheye images, which usually require a slow procedure to be rendered.
However another rendering method, the ray tracing, known for the superior level of realism it can achieve, is not based on rasterization and, as such, doesn't suffer this problem.
The purpose of this work is to show that, since hardware acceleration for ray tracing has become available in 2018, ray tracing is more efficient than traditional methods in rendering fisheye images and therefore not only produces better images, but it also has better performance.
To do so I have implemented fisheye both with a rasterization-based approach and with ray tracing, and I performed a comparison between their results.

2. Fisheye

Fisheye lenses are ultra wide-angle lenses, that produce pictures with wide field of view, while introducing strong visual distortion. These lenses are characterized by two factors: the focal length and the mapping function.
The focal length is a value that measure how the lens converges (positive focal length) or diverges (negative focal length) light. The smaller the focal length, the wider the angle of view will be.
The mapping function, instead, is a mathematical function that maps the angle of a point from the optical axis θ to the distance of that point from the center of the final image r, given the focal length f. In other words, it maps the direction of the incoming ray of light to its new direction, after it has passed through the lens.
The general form of any fisheye mapping function is the following:

$$r = \begin{cases} \frac{f}{k_1}\tan(k_2\theta) & for\ 0 < k_2 \leq 1 \\ f\theta & for\ k_2 = 0 \\ \frac{f}{k_1}\sin(k_2\theta) & for\ -1 \leq k_2 < 0 \end{cases} \quad (1)$$

By fixing the parameters k_1 and k_2 a specific mapping function is defined. The most important and most used mapping functions however, are the following four:

$$r = 2f\tan\frac{\theta}{2} \quad (2)$$

$$r = f\theta \quad (3)$$

$$r = 2f \sin\frac{\theta}{2} \tag{4}$$

$$r = f \sin\theta \tag{5}$$

(2) refers to the "stereographic" fisheye function; it tends to compress the center of the image, while keeping marginal objects bigger.
(3) is the "equidistant" mapping function; it maintains angular distances and can be practical for angle measurement.
(4) is the mapping function for "equisolid angle"; it compresses marginal objects and looks like a mirror image on a ball.
(5) corresponds to the "orthographic" function; It highly distorts objects near the edge of the image.

3. Ray tracing

Ray tracing is a rendering technique that tries to model light transport, by emulating the behaviour of rays of light and their interactions with the environment.
The image is produced by casting a ray for every pixel and then tracing its path, checking for intersections with the environment. Each ray is assigned a payload, that will store its information. When the ray hits a surface, a program is executed to define how this collision affects the ray itself. This may entail some color to be stored in the payload and, potentially, one or more recursive rays to be cast. When the tracing of the ray is completed, the payload of the ray will return the color that the pixel should assume.
The only part of this process affected by fisheye is the primary ray generation: instead of a normal generation, where every ray points towards the center of its corresponding pixel, the ray direction is modified by the fisheye effect. This new direction can be computed starting from the inverse of the mapping function, which returns the angle θ that the outgoing ray forms with the optical axis given the focal length f and the distance r of a pixel from the center of the screen. The inverse mapping function for the main fisheye types are:

$$\theta = 2\arctan\frac{r}{2f}$$

$$\theta = \frac{r}{f}$$

$$\theta = 2\arcsin\frac{r}{2f}$$

$$\theta = \arcsin\frac{r}{f}$$

Once the angle is defined, the direction vector can be computed by imposing that it lies on the same plane of the optical axis and the relative pixel of the screen. Calling v_1 the normalized optical axis, v_2 the normalized vector pointing to the pixel and v_3 the final normalized direction vector, the equation of the plane is:

$$a * v1 + b * v2 = 0$$

And since v_3 lies on that plane, there exist some values a' and b' such that:

$$v_3 = a' * v1 + b' * v2 \tag{6}$$

The second condition, knowing that the vectors are normalized, is that:

$$\cos\theta = v1 \cdot v3 \tag{7}$$

The third condition is that:

$$|v_3| = 1 \tag{8}$$

Solving the system of equations (6), (7) and (8) (where v_1 and v_2 are known) for a', b' and v_3, the result is:

$$a' = \cos\theta - b'(v1 \cdot v2)$$

$$b' = \sqrt{\frac{\sin^2\theta}{1-(v1 \cdot v2)^2}}$$

$$v3 = a' * v1 + b' * v2$$

This solution, then, is used during the ray generation to define the direction of each ray, as if it was distorted by a fisheye lens. The ray will then be traced as normal and return the pixel's color.
Therefore, this is the only extra computation needed to apply fisheye in the case of ray tracing. Moreover, the fisheye direction can also be stored in a texture, so that it doesn't need to be recalculated each frame. In the end, fisheye can be applied to ray tracing almost at no cost.

4. Rasterization

If the ray tracing procedure stays for the most part the same in case of fisheye, with rasterization it is impossible to directly render the fisheye

image. As such, an intermediate step is needed: the rendering of a cube map. Since this implies rendering the scene six times, the cost of applying fisheye becomes very high.
Some techniques can be used to lower this cost: for example it is possible to use deferred rendering. This involves to split each rendering in two passes: in the first the full scene is rendered, but instead of computing the lighting directly, the values needed to do so are saved in the textures of a buffer called G-Buffer. Then, the G-Buffer is used to compute the actual image in the second pass. The advantage of this technique is that the lighting is calculated only for the visible surfaces, since the G-Buffer only contains what can be seen by the camera.
An alternative to deferred rendering is to use the "single pass render to cube map" algorithm. It relies on a particular shader program called geometry shader, which receives as input the vertices of a single primitive and can transform them, possibly generating more vertices and primitives, before sending them to the next pipeline stage. Therefore, instead of repeat the rendering of the same scene six times, it is more convenient to let the geometry shader produce a new triangle for each cube map face, projected according to the corresponding view matrix. In this way the cube map is rendered in a single rendering pass.
Once the cube map is ready, its faces can be used to generate the final image.
First the fisheye direction vector must be defined for each pixel; this can be done either by computing it from scratch or by sampling it from a texture (as seen in section 3). Then, every pixel of the resulting fisheye image is sampled from the position of the cube map that the direction vector was pointing at.
The right face of the cube map can simply be determined by finding the component of the direction vector with the greater absolute value. For example, if it is the y component and its value is positive, then the target face is the top one; if it is negative then the direction vector points to the bottom face. If it is the x component the face will either be the left or the right, and if it is the z component, then the front or back face will be sampled.
Then, by dividing the other two component by the greatest one, it is possible to obtain the UV coordinates of the face to sample. For instance, if the greatest component is z and it is positive, then the face to sample is the front one and the sampling point is:

$$UV = (\frac{fisheyeDir.xy}{fisheyeDir.z} + 1)/2$$

By similar reasoning the UV coordinates for the other faces can be found.
Clearly this whole process is much more costly than the normal rendering of a scene, even when using deferred rendering or the geometry shader. For this reason fisheye has a huge impact on the performance of rasterization-based algorithms.

5. Validation

In order to validate the fisheye algorithm I checked the correctness of some rendered images of a grid with respect to the corresponding pictures obtained through real fisheye lenses. By setting the correct parameters in the model, the resulting render corresponds to the real picture.

Nikon Nikkor AF-S Fisheye 8-15 mm f/3.5-4.5E ED

Camera	Focal l.	Map. func.	Focal l.
Nikon D500	15 mm	Orthographic	1.54
Nikon D500	12 mm	Orthographic	1.35
Nikon D3x	10 mm	Orthographic	0.72
Nikon D3x	8 mm	Orthographic	0.62

Samyang 7.5 mm f/3.5 UMC Fisheye MFT

Camera	Focal l.	Map. func.	Focal l.
Olympus E-PL1	7.5 mm	Equisolid angle	0.93

Canon EF 8-15 mm f/4 L Fisheye USM

Camera	Focal l.	Map. func.	Focal l.
Canon 1Ds MkIII	15 mm	Equidistant	0.785
Canon 1Ds MkIII	8 mm	Equidistant	0.42

There are a few things to notice:

- A real lens, for obvious physical reasons, has a limited field of view, often smaller than the simulated one (which can easily reach 360 degrees and, for the equidistant mapping function, even more).
- The mapping function of real lenses often doesn't match perfectly with one of the four main mapping functions (stereographic, equidistant, equisolid angle and orthographic). However most fisheye lenses can be traced back to one of the four main functions with a fairly small error, possibly with an adjustment to the distance between the camera and the target.
- There is not a direct correlation between the focal length of a real fisheye lens and the one in the model, since the former has a physical meaning, while the latter does not. In reality the final image is influenced by the curvature, size and number of the lenses, by their distance from the film or sensor, the size of the latter, etc. Different film or sensor sizes produce different effects, even if the focal length is the same. The digital version, instead, doesn't have any sensor, and the effect is simulated with the mathematical model. Here the screen always has the same - normalized - size, and thus only the focal length is required. However, this focal length, since it doesn't refer to the size of any sensor, is just a positive number. For this reason, different lenses with the same focal length can have rather different focal lengths in the model.

With the parameters in the tables, the software fisheye is able to correctly reproduce the optic effect, applying the same distortion on the grids of a real lens.

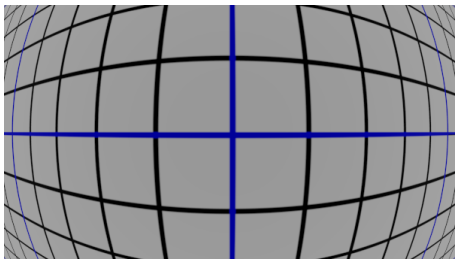

Figure 1: Example: Samyang 7.5 mm f/3.5 UMC Fisheye MFT

6. Experimental results

To compare the two algorithms I have first checked their performance: to do so I considered some 3d scenes and I defined the path of the camera in each of them. Then, running the program in different setups(fisheye on/off, RTX on/ off, deferred/forward rendering), I have registered the frame rate. These tests were done on a desktop Windows computer with a Nvidia RTX3050 GPU and an i5-11500 Intel Core CPU and the application was made with DirectX 12.
The results show a trade off between ray tracing and rasterization: in case of scenes with many small objects, the former outperforms the latter, while the opposite happens in scene with few highly-detailed objects. This can be explained by the nature of ray tracing: when there are many small meshes, most of them will be easily discarded during the BVH traversal; at the end the ray will intersect only a very small number of triangles, so the computation is not too expansive. On the other hand, with few complex meshes finding the collision may become much more complex, since the bounding volumes of the BVH's nodes will be tightly packed and, thus, more tests must be performed.
Scanline-based algorithms, instead, are very efficient at drawing complex objects, but when the number of draw calls increases, the performance drops.
Another observation is that in almost all cases, ray tracing's frame rate is much less stable with respect to traditional rendering techniques.

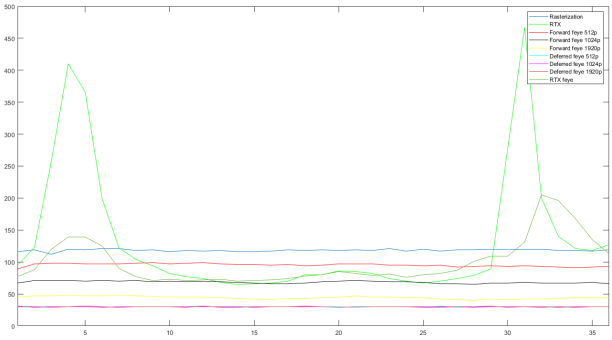

Figure 2: Frame rate in the "Sun Temple" scene (Open Research Content Archive).

Figure 2 is a clear example of this: for the most part, rasterization is better than ray tracing, but the latter has some very high peaks in a couple of points, which increase the average frame rate

(131.28 fps with a variance of 10282.15 for ray tracing against 118.28 fps with a variance of 3.18 for rasterization). This can also be explained by the fact that the cost of a trace ray call depends on the BVH traversal: when rays have to intersect a complex structure, the rendering is slow, but when the intersection is easily found (for example when pointing to a wall) the speed can increase tremendously.
However, when it comes to fisheye, ray tracing always outperforms rasterization.

Average	1	2	3	4	5
Raster	118.3	62.8	76.1	685.4	14.7
RTX	131.3	133.6	62.0	409.2	190.0
Forward 512p	95.3	56.2	31.7	165.2	14.1
Forward 1024p	68.4	52.4	28.3	158.2	14.1
Forward 1920p	44.2	42.7	22.7	142.0	13.9
Deferred 512p	29.9	14.6	18.1	465.3	3.9
Deferred 1024p	29.8	14.1	18.1	348.9	3.9
Deferred 1920p	29.9	13.9	17.9	180.5	3.9
RTX fisheye	99.4	108.3	54.0	574.2	209.4

Table 1: Mean values of the frame rates in various scenes. The first two rows are relative to the experiments without fisheye.

Table 1 shows evidently that simulating fisheye strongly degrades the performance of all non-RTX renderings, due to the multiple scene renders necessary for the cube map. Ray tracing frame rate, instead, is generally similar with or without fisheye. Sometimes fisheye can even increase the performance, in case the camera focus on a highly detailed object (like in the fourth scene): since fisheye increases the field of view, a smaller number of rays will intersect the object, and, as such, the number of slow BVH traversal is lower.
On the rasterization side, between the single pass forward rendering and the six deferred rendering the former is in general the best (except in scene 4, where the number of draw calls is very small).
A possible way to improve the frame rate is to reduce the resolution of the cube map's textures, however this implies the reduction of the image quality as well. In order to consider the impact of this modifications, as well as the general difference between the quality of ray tracing and rasterization, I have also compared images obtained with the various methods, computing the Mean Squared Error between the RGB values of their pixels.
Having used recursive ray tracing, the main differences between ray tracing and rasterization is in three aspects: reflection, refraction and shadows.

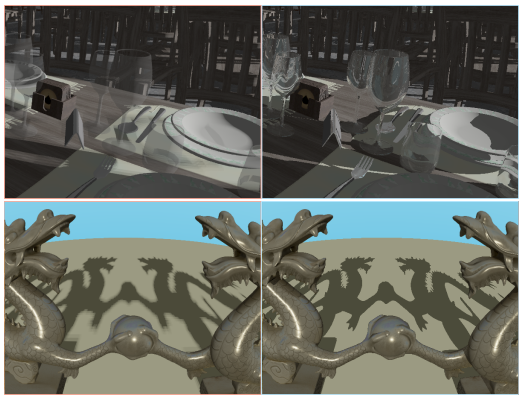

Figure 3: RTX on the right, rasterization on the left.

There is also a clear quality drop between a 1920p and a 512p cube map, with usually a MSE greater than 0.1% (>1% in average). This apparently small number makes a big difference, as shown in the following image:

Figure 4: Difference between a 512p (left) and a 1920p (right) cube map in the "Amazon Lumberyard Bistro" scene (Open Research Content Archive).

7. Conclusions

The results presented here show that ray tracing is a very efficient way to simulate the pres-

ence of a fisheye lens. Not only the image can benefit from the advantages in terms of realism provided by the ray tracing, like dynamic reflections, refraction, precise shadows and, possibly, even more (with advanced ray tracing techniques), but the rendering itself is faster, since fisheye has a minimal effect on the computational load. Its implementation with traditional rendering methods, instead, requires a considerable extra cost, due to the need to render a cube map per frame.
This application of ray tracing could be implemented in the major 3D game engines, like Unreal Engine, Unity and CryENGINE, giving in this way the developers another tool for their applications.
Moreover fisheye is not the only type of lens that can take advantage of the structure of the ray tracing algorithms. Many other optical effects present in real lenses can be simulated with this method, just by applying the model of the light transmission through the lens to the ray generation program.
Obviously the use of this technique comes with a cost: in order to take advantage of the hardware acceleration required for efficient ray tracing, a modern - and possibly more costly - GPU is indispensable. Nevertheless, the advancement brought by ray tracing to the field of computer graphics is undeniable. As the most recent frontier in this scope, many studies have been conducted in the last few years to test its potential and undoubtedly there will be further developments and applications. With this work I tried to add my contribution to this research, showing another use of ray tracing that gives great benefits over the normal rasterization procedures.

References

[1] F. Bettonvil. Fisheye lenses. *WGN, Journal of the International Meteor Organization*, 33(1):9–14, February 2005.

[2] Adam Marrs, Peter Shirley, and Ingo Wald. *Ray Tracing Gems II - Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Springer Nature, 2021.

[3] Microsoft. DirectX Raytracing (DXR) Functional Spec, October 2018.

[4] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, jun 1980.

[5] R. W. Wood. Xxiii. fish-eye views, and vision under water, August 1906.