Foveal Rendering

Eric Knapik and Victoria McGowen

December 9 2016

Abstract

This semester project focused on rendering a scene using eye gaze data from the observer. A mobile eye-tracker was used to gather the gaze information in screen space, which was then translated into world space to manipulate the polygon density of the scene's objects. The closer the observer's gaze point was to an object, the scene would render a higher polygon count for that corresponding mesh. This rendering approach is used to save memory and increase GPU performance by optimally minimizing rendered polys in a scene.

1 Background and Theory

Foveal rendering is a process of rendering amount of detail in a scene similarly to sharp vision in the human visual system. The human eye is a roughly spherical, light-tight enclosure consisting of a hard, white outer wall called the sclera, a clear cornea that provides most of the optical power of the eye, the lens which can adjust its optical power to let you focus on objects at different distances, and a light-sensitive layer at the rear of the eyeball called the retina where the image is formed. Colour, light, and dark vision are due to photoreceptor cells in the retina called rods, cones, and ganglion cells. Rods are light intensity sensitive, are responsible for peripheral vision, and help in dark adapted situations. Cones provide colour vision and sharpness. There are three different types of cone cells: short, middle and long wavelength cones. These cones correspond to blue, red and green light respectively. Ganglion cells take visual information and transport it to the rest of the visual system in the brain. The fovea is a small pit in the back of the eye that is responsible for colour vision as it has the highest concentration of cones. Because of this, this region is responsible for the highest visual acuity and is known as foveal vision. The sharpness of vision quickly decreases outside of the fovea in the peripheral vision as there is a sharp decrease in cones in the rest of the retina (Fig 1).

With this high level understanding of how the eye works, a foveal renderer renders the highest level of detail at the fixation location of the observer (i.e. where the observer's fovea would be looking at). Outside this predetermined foveal rendering radius, the level of detail in the scene would fall-off at a rate

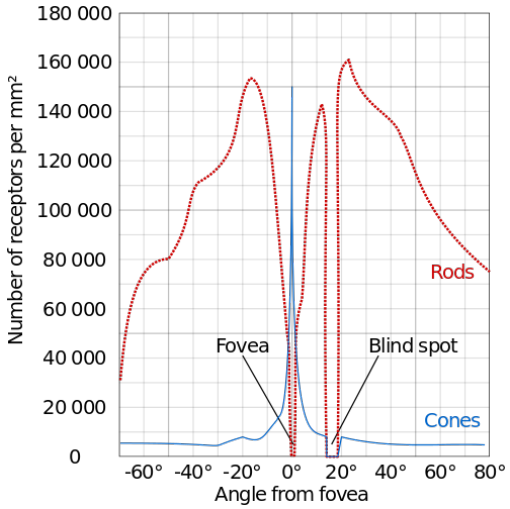


Figure 1: Distribution of rods and cones in the eye. Notice how the highest number of cones occur in the fovea and there are no rods in the fovea. [1]

that mimics peripheral vision. This technique of rendering was initially created in the late 1990s to save processing power as an observer would not notice the level of detail for an object outside their sharp vision.

In previous work, ray tracers have been modified to subsample the scene at different rates depending on predefined pixel distances away from the fixation point [5]. This approach, however, increased spatial aliasing that was rather distracting to the user and more intensive antialising techniques needed to be incorporated. An improvement on this was masking and blending into these regions with the different subsampled rendered versions of the scene as shown in Figure 2. For this approach, the scene was rendered at different levels of details to match the number of needed masks, which initially costs processing power, but overall improved the visible distortions in subsampling at different rates.

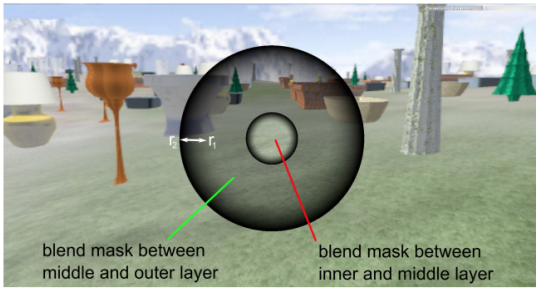


Figure 2: Mask and blending margin used in Guenter 2012 [2]

There is also the model based approach. This approached varied by either changing the simple mesh by increasing or collapsing edges for the different levels of detail [3]. Techniques like this were improved by modeling the pixel distances for level of detail fall-off from the Contrast Sensitivity Function (CSF) [4]. Contrast is the difference between colour and illumination between objects in a scene, making them distinguishable from one another. The CSF is directly related to visual acuity as it shows the relationship between visual sensitivity to a given frequency (i.e. the ease of distinguishing different objects at different angluar frequencies) shown in Fig 3. To further understand CSF, think a white fence along an angled plane. Up close, every board is individually noticed, but the fence becomes a white blur the further away it is. For this example, the sensitivity is highest at the part where each board is noticeable, and quickly falls off when each board appears to blend together. Using CSF as a means to determine detail fall-off by converting cycles/degree to pixel distance from foveal center prevented the need to dramatically manipulate the mesh itself in every layer, and decreased artifacts in the models caused by creating or collapsing edges [4]. Since changing the model's mesh on the go can be costly, preloaded models for each of the different levels were also tested, and shown to increase the process [3]. The level of detail in polygons for each model was arbitrarily chosen to produce an aesthetically pleasing fall-off.

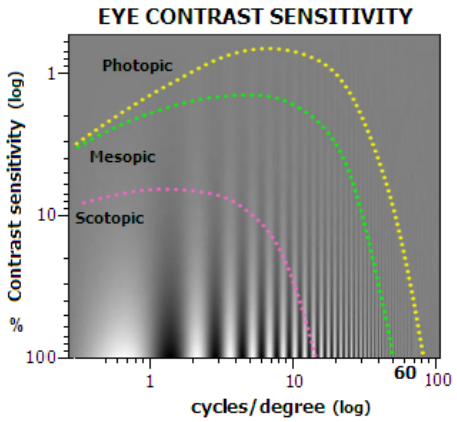


Figure 3: Contrast Sensitivity Function of the average human in light adapted environments (Photopic), in dim light environments (Mesopic), and in dark adapted environments (Scotopic)

Most previous systems have been hindered by latency [2]. This issue is still a problem in more recent work as latency depends on the refresh rate of the eye-tracking equipment and the graphics card on the computer. Users were also very aware of the changes to the scene in all approaches, thus the optimal fall-off function is still a problem that is being worked through today.

2 Method

The project was written in C++ and DirectX libraries were used for the graphics. A Tobii EyeX eye-tracker was used to collect the eye gaze data. The Tobii has a refresh rate of 60 Hz and reports eye-tracking data in screen resolution space, so a conversion between screen and scene is needed to work with the data. The Tobii eye-tracker is compatible with Windows OS, and in choosing to use DirectX for graphics, our system can only be used on Windows machines. The project has an option for either eye-tracking input or mouse input, which allows it to run on systems without eye-tracking sensors.

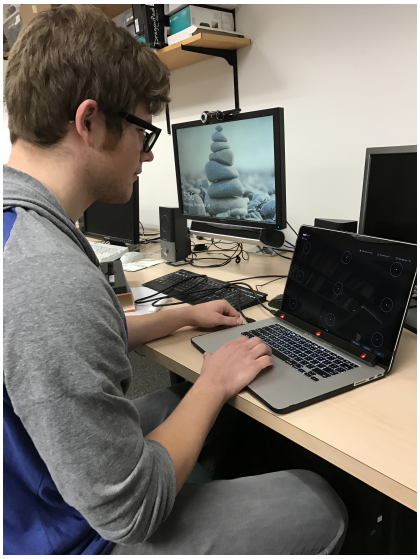


Figure 4: Observer calibrating the Tobii EyeX eye-tracker

A combination of approaches was taken for this project: the model based approached and the different level of detail masking approach. An object set of well known Stanford meshes were used within the test scene (Fig 5). Each high poly mesh was down sampled by 90% to achieve the mid level of detail mesh, and the mid poly meshes were further down sampled again by 90% to get the low level of detail meshes. The exact number of polygons in each mesh is outlined in Table 1. These meshes were placed throughout the scene at various locations to further see the difference in level of detail across the xyz axes. The difference between rendering the scene at just low level of detail to just high level of detail can be seen in Fig 6.

The project implementation strongly utilized a deferred rendering system. Deferred rendering uses auxiliary buffers to store scene information such as, color, depth, normals, position, etc. This is typically called the Gbuffer, and is utilized so that the lighting of the scene can take place later on a per pixel level rather than a per object level. How this normally works is to render all objects

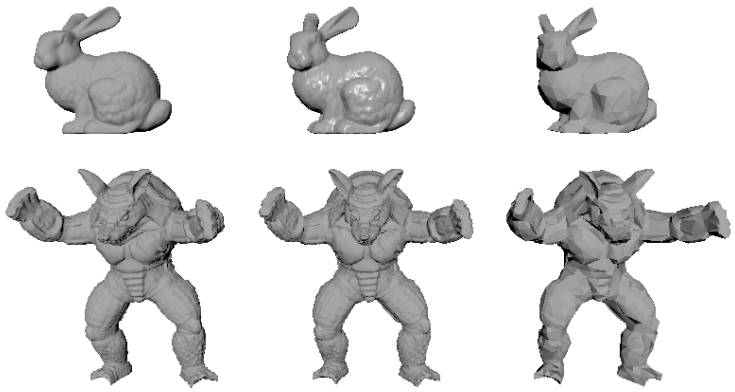


Figure 5: Test object set. Top Row: Stanford Bunny. Left to Right: high to low poly. Bottom Row: Stanford Armadillo. Left to Right: high to low poly

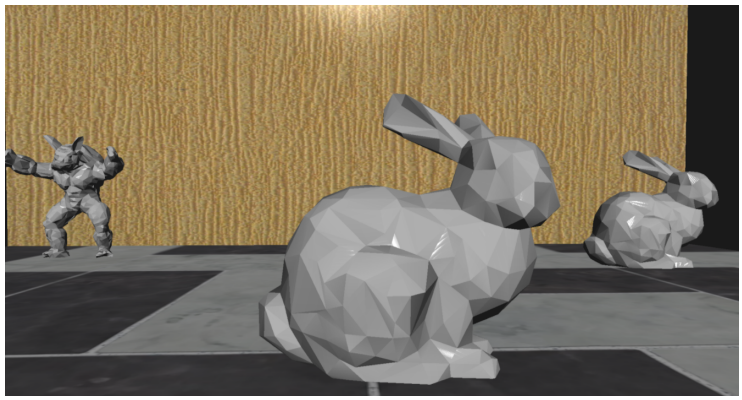
Objects	Polys in Mesh
Bunny - Low Poly	692
Bunny - Mid Poly	6960
Bunny - High Poly	69630
Armadillo - Low Poly	2122
Armadillo - Mid Poly	21254
Armadillo - High Poly	212565

Table 1: Poly count comparison throughout pipeline

to the auxiliary buffers, which are the same size as the screen. The normals, the positional data, the color information, etc, are put into their own buffer for later handling. For our system, we paid attention to the object's position, normals, and color.

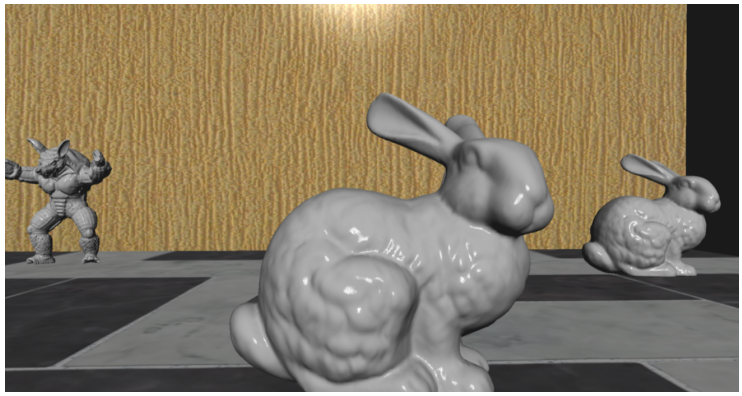
The rendering process for this system is as follows.

1. Render all of the objects in the scene at their lowest resolution.
2. Render the foveal mask to the stencil buffer, this is just a simple circle at the correct screen space position with a variable circle radius.
3. Render higher resolution geometry only inside that foveal mask rewriting values inside the Gbuffer.
4. Iterate steps 2 - 3 rendering a smaller mask each time and increasing the object's detail at each iteration, for our implementation we use two tiers of foveal mask resulting in three different levels of geometry in total. (low resolution, medium resolution, high resolution)
5. Finally render to the screen doing typical deferred lighting calculations.



(a) Low poly version

(b) Mid poly version



(c) High poly version

Figure 6: Unmasked low to high poly count representations of the test objects in the scene space.

Below are samples of the final result. Fig 7 shows the actual masks overlaid onto the scene. If you look at the armadillo in the far left, it can be seen that it is being rendered entirely in low poly mode, as it is within our base layer. The high level of detail can be best seen in the nose of the forward most bunny as it appears free of sharp edges. The gradual transition between low and high level of detail is thanks to the mid poly level, which can best be seen around the hip and knee of the forward most bunny, or the left half of the back right bunny.

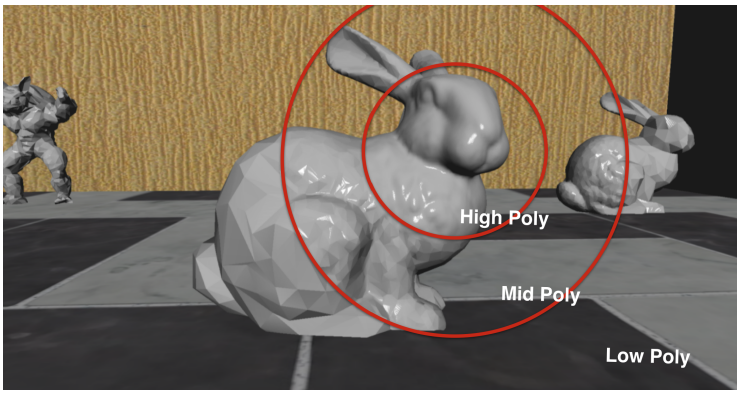


Figure 7: Where the masks overlap in the example rendered frame

Without the mask outliner, it is not as easy to see the location of the masks when looking at the front-most bunny as a whole (Fig 8). Obviously, the difference between the bunny's nose and tail is evident, but to a user looking at the bunny and controlling the level of detail rendering with their foveal vision, the lower masked portion of the mesh is in their periphery and thus not as clearly seen.

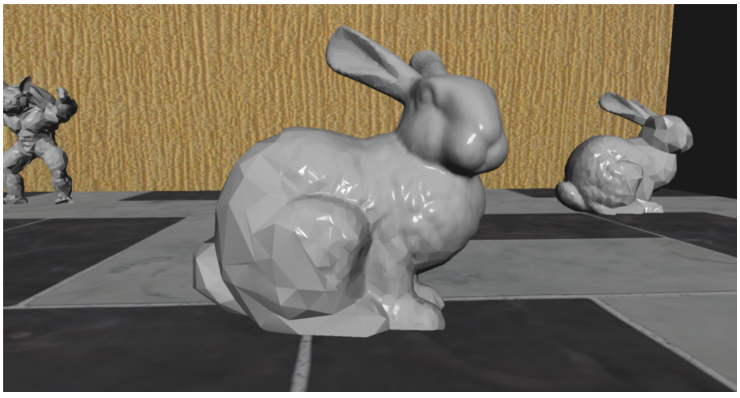


Figure 8: Final rendered frame

3 Conclusion

Between several test cases, it would appear that using foveal rendering for objects close up is very obvious. For small objects in a middle view range and objects at a far view range, subjects conclude that it is seemingly unnoticeable. Further research and incorporating a function that mimics the visual system, such as CSF, could improve the overall look of the scene for objects closest to the camera. Smoothing between the masks might also add the needed blurring to help create a more general transition between the levels of detail. Due to parameters in the program, it could easily be changed to allow objects in the middle ground and background to be more significantly affected by the change of level of detail. This, however, would cause all objects in the foreground to be rendered at their highest resolution. Depending on the scene, the parameters for each level of detail should be updated to best fit the size and distance in the scene (i.e. a shallower scene would need smaller level of detail thresholds than a deeper scene).

There is a noticeable increase in performance speeds when using the foveal rendering (Table 2). For a scene with 628446 polygons (low, mid, and high scenes being rendered at the same time and then integrated) and 11896 visible polygons, the speed resulted to a steady 359 fps versus a 343 fps if rendering and showing the high level of detail scene in its entirety.

Rendering	Total Poly Count	Visible Polygons	GPU Usage	fps
Low LOD Only	5628	5628	24.4%	706
Mid LOD Only	58428	58428	28.6%	522
High LOD Only	564390	564390	51.8%	343
Foveal Rendering	628446	11896	34.7%	359

Table 2: Poly count comparison for different level of detail scenes for the test scene. The total poly count for Foveal Rendering is the sum of low, mid and high scenes as each scene type is essentially rendered to create the masks. Visible polygons relate to what is actually being shown to the user, which only varies for the foveal rendered scene. The GPU usage % was found by averaging the reported % over a minute of rendering on a computer with a NVIDIA TITAN graphics card. The fps is that reported during debugging by Visual Studios' graphical debugger.

Another future improvement that could be added to this system would be to calculate the normals over the masks to hide visual differences between the different levels of detail over a mesh. Adding the normals would give the low poly portions a smoother look and hide the large, blocky tessellation. This can improve visual clarity without increasing geometry.

References

[1] The visual experience - retina, 2014.

[2] Guenter, B., Finch, M., Drucker, S., Tan, D., and Snyder, J. Foveated 3d graphics. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 164.

[3] Murphy, H., and Duchowski, A. T. Gaze-contingent level of detail rendering. *EuroGraphics 2001* (2001).

[4] Murphy, H. A., Duchowski, A. T., and Tyrrell, R. A. Hybrid image/model-based gaze-contingent rendering. *ACM Transactions on Applied Perception (TAP) 5*, 4 (2009), 22.

[5] Reddy, M. Musings on volumetric level of detail for virtual environments. *Virtual Reality 1*, 1 (1995), 49–55.