Simulation of Diabetic Macular Edema in Virtual Reality

Thomas Koch, 01526232

Abstract

Simulation of diabetic macular edema (DME) [1] is implemented in a virtual reality simulation using Unreal Engine 4 [6, 7, 8]. Common symptoms of DME are blurry vision, loss of contrast, floaters and distorted vision. We use different computer graphics techniques to create effects which resemble such symptoms. An eye tracker from Pupil Labs [2] is used in order to make effects gaze dependent. The implementation of these effects is discussed and adjustable parameters of the effects are explained.

Introduction

Common symptoms of DME are shown in Figure 1. In the following the implementation as well as the parameters of the effects are discussed. A variety of parameters are exposed in order to allow for greater flexibility. This also simplifies reusing certain effects for the simulation of other eye diseases.

Figure 1: From [3]. ©2015 by The Angiogenesis Foundation, Inc., All Rights Reserved. Symptoms of DME: Blurry vision, loss of contrast, floaters and distorted vision.

Distorted Vision

One of the effects, implemented in this project, is a circular distortion and can be parameterized to resemble either an inward or outward bulge. This is implemented using blueprints and is used in the `PP_DiabeticMacularEdema` blueprint. The idea is to distort the texture coordinates which are then used to sample the scene texture in a post processing stage. To reduce code duplication a material function (`MF_Distortion`) is created and has five parameters to control the effect:

- `Radius`: The radius of the circular distortion.
- `Center`: The radius of the undistorted center. Must be smaller than the radius.
- `Position`: The position where the distortion is placed on the screen.
- `Strength`: Controls the strength of the distortion. A number greater than zero results in an outward bulge (convex) while a number lower than zero results in an inward bulge (concave).
- `In`: The texture coordinates that are distorted.

The core of the effect is a simple `smoothstep` function which causes the distortion.

Note: Multiple distortion effects can be overlapped by using the output of one distortion as the input of the `In` parameter of another distortion.

Figure 2 shows two distortion effects of different strength.

Figure 2: Distortions are applied around the center and the top right of the image.

Floaters

In the case of DME, floaters are small dark spots appearing in the field of vision caused by bleeding [4]. The typical behavior is that floaters drift away when trying to look at them.

A set of textures is used, each containing blurry black spots. The floaters move with the movement of the eye and have an offset relative to the gaze point.

The contiuous update of the position each frame is handled on the CPU in `BP_Pawn`. The rendering of the floaters is handled on the GPU in the material `PP_DiabeticMacularEdema`. The relative movement speed of the floaters is stored in the `FloaterVelocities` array and the relative position is stored and continuously updated in the material parameter collection (MPC) `MPC_Floaters`. Due to a limitation of Unreal Engine 4.20.3, it is not possible to send an array to the GPU to be accessed in a material. Therefore it was necessary to employ an alternative solution using a MPC to store parameter values of floaters that should also be accessible in a post processing material in order to render them. Another idea would be to store those parameters in a texture which is updated on the CPU and sent to the GPU each frame.

At the time of writing, the parameter values for floaters are generated randomly in `BP_Pawn`. Repeatedly pressing the Diabetic Macular Edema button in the UI generates new, random floaters.

Figure 3 shows four floaters of different scale and opacity.

Figure 3: Four floaters of different scale and opacity can be seen.

MPC_Floaters

The parameter collection contains three parameters for each floater. `<i>` has to be replaced by an index with the first floater starting at 0. This associates the

i-th floater with its parameter values. This allows accessing and iterating over the MPC entries live over an array, from inside a material. The MPC contains the following parameters for each floater:

- `PositionEye<i>`: Stores the two-dimensional position of the i-th floater in the xy-coordinate. The index of the eye (0 or 1), the floater should be visible for, is stored in the z-coordinate. This parameter represents the start position of the i-th floater.
- `ScaleOpacityTexture<i>`: A texture that stores a uniform scaling factor of the floater texture in its x-coordinate, an opacity value in its y-coordinate and a texture index, which specifies the floater texture to be used, in its z-coordinate.
- `Offset<i>`: Stores the two dimensional offset relative to the gaze point as xy-coordinate.

The MPC contains the `PositionEye<i>` and `ScaleOpacityTexture<i>` parameter interleaved and the `Offset<i>` parameter at the end. For two floaters the MPC should be structured as follows:

- `PositionEye0`
- `ScaleOpacityTexture0`
- `PositionEye1`
- `ScaleOpacityTexture1`
- `Offset0`
- `Offset1`

Parameters cannot be added to a MPC via code and therefore have to be added by hand.

Note: The number of floaters is stored in the variable `NumFloaters` in the blueprint `BP_Pawn` and has to be set correctly.

Floater textures

The texture resources have to use the naming scheme `T_Floater<k>`. The texture index specified in the `ScaleOpacityTexture<i>` parameter is the number `k`.

Usage in the material

The MPC is accessed and the floaters are rendered using the custom shader node `Floaters` in the `PP_DiabeticMacularEdema` material. The shader node has the following parameters:

- `ViewportUV`: UV coordinates used to sample the floater textures.
- `FloaterTex<i>`: The floater textures where `<i>` has to match the `<i>` in the name of the texture resource (`T_Floater<i>`) which is connected to it. Example: Texture `T_Floater3` should be connected to the input parameter `FloaterTex3`.
- `Dummy`: A dummy input that is never used. However, the MPC has to be connected with the shader as input parameter even though the input parameter is never used. Otherwise the MPC will not be accessible in the shader. The MPC is accessed via the variable `MaterialCollection0` which does not have to be specified as input parameter but allows to access the MPC like an array.
- `NumFloaters`: The number of floaters. This variable has to be set in the `BP_Pawn` blueprint.

Contrast Loss and Blurry Vision

In the work of Thompson et. al. [9] a method is described which handles contrast loss depending on the frequencies appearing in the image by using a contrast sensitivity function (CSF) [5]. A CSF specifies the lowest contrast level for each spatial frequency one can detect. This method yields believable results, however, it is not suited to be implemented as is in an application requiring interactive frame rates, due to the use of computationally expensive operations. Therefore, a simpler method is implemented which gives visually comparable results and runs in real-time, but needs more tuning via parameters to get the desired effect.

The idea is that areas with fine detail should be blurred much more than strong lines with a good contrast to their background. This is implemented in code as HLSL shader (`EdgeBlur`) and is used in the `PP_DiabeticMacularEdema` blueprint. The shader takes four parameters:

- `SceneTexture`: The texture the shader modifies.
- `BlurRadius and BlurSigma`: Control the strength of the blur. These are used directly as parameters for the Gaussian blur that's used for blurring.
- `UV`: Viewport UV coordinates.
- `BlurFineDetail` and `EdgeThreshold`: If `BlurFineDetail` is set to 1, `EdgeThreshold` is used to include weak edges, meaning that a low threshold (e.g. 0.01) blurs edges with weak contrast (i.e. fine detail). Raising the threshold will blur more and more strong edges as well. On the other hand, if `BlurFineDetail` is 0, the lower the threshold the blurrier the overall image, the higher the threshold the smaller the number of weak edges that are blurred. Thus, a high threshold (e.g. 0.15) will only blur strong edges.

Essentially, the boolean `BlurFineDetail` determines, if weak or strong edges should be blurred more.

At its core, the shader works as follows: First, edges are detected by calculating partial derivatives. The edge detection is relatively fast and simple, using the HLSL function `fwidth`, which is the same as writing `abs(ddx(x)) + abs(ddy(x))`. The output of the shader is the linear interpolation between `SceneTexture` and a blurred version of `SceneTexture` using the blurred edge response as interpolation value. The edge image is blurred in order to smoothen transitions as seen in Figure 4.

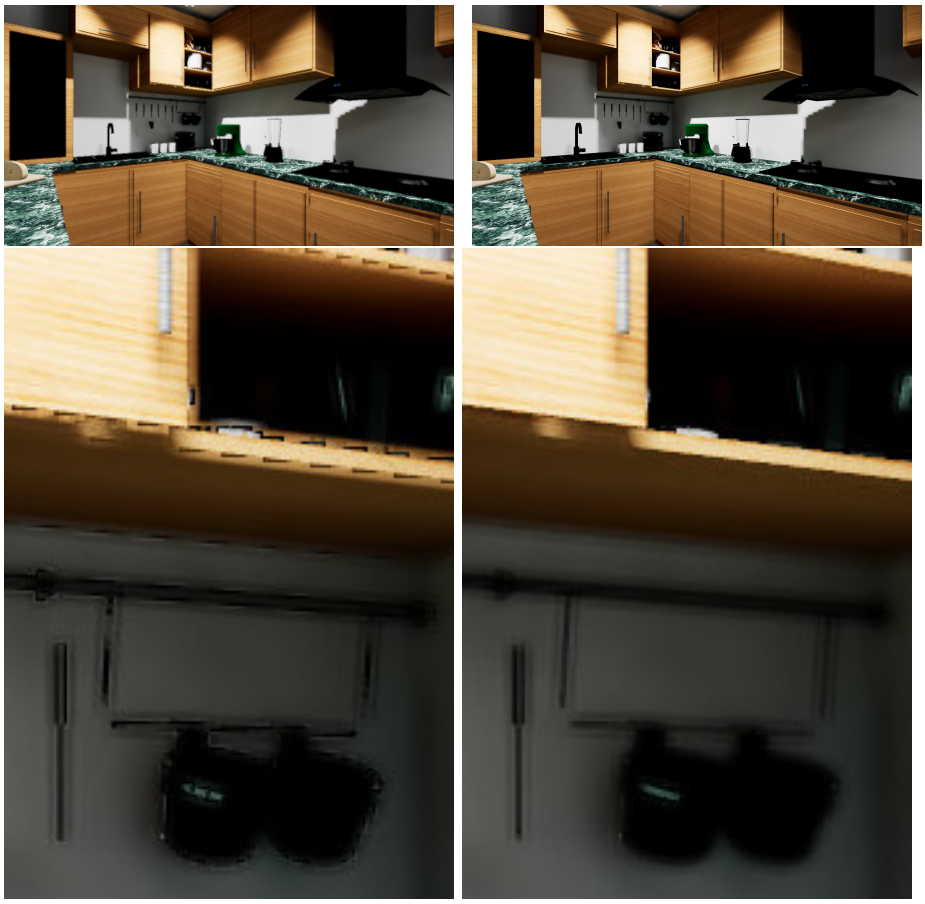

Figure 4: Left: Partially blurry vision without blurring the edge image. Sharp outlines around objects can be seen. Right: Partially blurry vision with the edge image blurred. Artifacts but also the strength of the effect is reduced.

Conclusion and Results

Common symptoms of diabetic macular edema (DME) such as floaters and distorted and blurry vision have been implemented. A variety of parameters is exposed which can be adjusted to get the desired effect. This also simplifies reusing certain effects for other eye diseases. For instance, reusing the floaters effect for other eye diseases or even other symptoms is simple since the appearance only depends on textures and the behavior can easily be adjusted. A combination of all implemented effects can be seen in Figure 5.

Figure 5: All implemented effects are shown: Floaters and distored and blurred vision.

Future Work

Future work involves adding more types of distortion which can also be combined to allow for greater flexibility. One type of distortion could be a wave-like distortion affecting the whole image. The strength as well as the direction of the waves would be parameters that can be exposed.

Furthermore, reworking the way the floaters are implemented is part of future work. The material parameter collection holding parameter values for each floater could be replaced by a texture or a simple array as soon as Unreal Engine allows to make arrays accessible in materials.

Also, exploring more sophisticated approaches for the contrast reduction effect that incorporate contrast sensitivity function can be explored in the future.

References

[1] https://nei.nih.gov/health/diabetic/retinopathy. Accessed 24.08.2019.

[2] https://pupil-labs.com/. Accessed 24.08.2019.

[3] https://www.noweyeknow.ca/dme/symptoms/. Accessed 24.08.2019.

[4] https://www.mayoclinic.org/diseases-conditions/eye-floaters/symptoms-causes/syc-20372346. Accessed 24.08.2019.

[5] https://www.allaboutvision.com/eye-exam/contrast-sensitivity.htm/. Accessed 24.08.2019.

[6] Katharina Krösl. [DC] Computational Design of Smart Lighting Systems for Visually Impaired People, using VR and AR Simulations. In *Proceedings of the 2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. IEEE, October 2018.

[7] Katharina Krösl, Dominik Bauer, Michael Schwärzler, Henry Fuchs, Michael Wimmer, and Georg Suter. A VR-based user study on the effects of vision impairments on recognition distances of escape-route signs in buildings. *The Visual Computer*, 34(6-8):911–923, April 2018.

[8] Katharina Krösl, Carmine Elvezio, Matthias Hürbe, Sonja Karst, Michael Wimmer, and Steven Feiner. ICthroughVR: Illuminating Cataracts through Virtual Reality. In *To appear in 2019 IEEE Virtual Reality (VR)*, March 2019.

[9] William B. Thompson, Gordon E. Legge, Daniel J. Kersten, Robert A. Shakespeare, and Quan Lei. Simulating visibility under reduced acuity and contrast sensitivity. *J. Opt. Soc. Am. A*, 34(4):583–593, Apr 2017.