Afterimage Rendering for Real-time 3D Virtual Environments

Conor O’Brien
B.A.I. Engineering
Final Year Project  April 2014
Supervisor: Dr. John Dingliana

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this project is entirely my own work and that is has not been submitted as an exercise for a degree at this or any other university.

_________________________________________  _____________
Name                                         Date
Abstract

One of the main goals in computer graphics for entertainment is to give the user as much of a sense of immersion in the virtual environment as possible. Visual fidelity is a key aspect in increasing the user’s sense of immersion. This includes modelling how the eye reacts to changes in stimuli. A number of techniques exist that emulates the eye’s behaviour to increase realism. Afterimages are an eye effect that has only recently started to be explored in the context of computer graphics that could greatly contribute to a user’s sense of immersion.

The aim of this project was to create a model for generating afterimages that emulates the physical process that causes afterimages to appear in the visual system, without impacting on the performance of the application.

The resulting program provides a model for afterimages that emulates the physical process of generating afterimages with as much fidelity as possible. The model runs in real time at 60Hz and takes advantage of the graphics hardware to keep the model as lightweight as possible.
# Table of Contents

Declaration...........................................................................................................................................2
Abstract..................................................................................................................................................3
1. Introduction .........................................................................................................................................6
   1.1 Afterimages .................................................................................................................................6
   1.2 Motivation .....................................................................................................................................6
   1.3 Aims ...............................................................................................................................................7
2. Background .........................................................................................................................................8
   2.1 Afterimages .................................................................................................................................8
   2.2 Bloom Shaders .............................................................................................................................9
   2.3 Adaptation ...................................................................................................................................10
   2.4 OpenGL ........................................................................................................................................11
   2.5 Shader programs ..........................................................................................................................14
   2.6 Similar Works / Previous Research ..............................................................................................14
   2.7 Possible uses ...............................................................................................................................15
3. Implementation ...................................................................................................................................17
   3.1 Render to texture ..........................................................................................................................17
   3.2 Transparency ...............................................................................................................................18
   3.3 Choropleth mapping ......................................................................................................................18
   3.4 Masking .......................................................................................................................................19
   3.5 Gaussian Blur / Gaussian Smoothing .............................................................................................20
4. Model ..................................................................................................................................................21
   4.1 Scene ............................................................................................................................................21
   4.2 The choropleth map ......................................................................................................................22
   4.3 Masking .......................................................................................................................................23
   4.4 Blink Model .................................................................................................................................24
   4.5 Afterimage Source .......................................................................................................................25
   4.6 Conditions for Generation ...........................................................................................................26
   4.7 Decay ..........................................................................................................................................26
   4.8 Compound Afterimages ...............................................................................................................27
5. Results ................................................................................................................................................29
   5.1 Verification of results ....................................................................................................................29
   5.2 Runtime Stills ...............................................................................................................................30
6. Conclusions and Further Work ..........................................................................................................33
6.1 Conclusions ................................................................................................................................................. 33
6.2 Limitations.................................................................................................................................................. 33
6.3 Further Work............................................................................................................................................... 34
7. Glossary and Technology used .................................................................................................................. 36
  7.1 Glossary................................................................................................................................................... 36
  7.2 Technology used ...................................................................................................................................... 38
Bibliography .................................................................................................................................................... 40
Appendix A – Files on CD ............................................................................................................................... 42
1. Introduction

1.1 Afterimages

In computer graphics we strive to make virtual scenes as realistic as possible to give the end user a sense of immersion. This can be done in a number of ways. Higher resolution models and textures give objects in the scene a more realistic appearance and allow for large amounts of detail to be expressed. Lighting effects give the scene a more natural appearance by creating shadows and reflections. Eye effects make the user feel more immersed in the world by emulating how the user’s visual system would react if they were in the scene. Afterimages are an example of an eye effect that would increase the user’s sense of immersion in the scene.

An afterimage is a phantom image in your field of view that is created after exposure to a bright light or intense colour. An afterimage takes the shape of the object or light source that caused the afterimage. From the onset the shape of the afterimage is blurred and over time it continues to deform.

1.2 Motivation

Thus far in graphics, afterimages are a relatively new area of interest. However they could have a big impact on the user’s sense of immersion in the scene, primarily in applications that are from a first person perspective. Afterimages present the interesting problem of being very subjective, so coming up with a model to give them a realistic appearance, that is also aesthetically pleasing to the majority of users is quite challenging.

With recent advances in virtual reality hardware, like the Oculus rift, modelling how the eye behaves with computer graphics can have an even bigger impact on how immersed the user feels. Alongside effects like bloom shaders, and adaptation; afterimages could give the user the sense that they are in a real environment, rather than a virtual one.
1.3 Aims

The aim of this project is to produce a model for afterimages in a 3d rendered environment that meets the following criteria:

- Should run in real time.
- The afterimage model, wherever possible, should emulate the behaviour of an eye.
- The model should be easily modifiable and be suitable for either realistic or artistic purposes.
- Should not be dependent on High Dynamic Range (HDR).

---

1 Glossary 7.1.5
2. Background

This chapter includes descriptions of afterimages and existing techniques used to emulate how the eye reacts to various stimuli, as well as a discussion on similar works on the topic of afterimages and what such a system could be used for.

2.1 Afterimages

There are two types of afterimages. A positive afterimage is caused by a brief exposure to a bright light. Positive afterimages are short lived and the colour of the afterimage is the same as that of the source. A negative afterimage is caused by long term exposure to either a light source or another source of intense colours. Negative afterimages last significantly longer than positive afterimages and the colour of the afterimage is inverted, relative to that of the source (Anstis, et al., 1977). This application focuses on negative afterimages.

The eyes reaction to light is caused by a protein called ospin in the cones of the eye. Cones are split into 3 groups: L, M, and S cones. L cones respond to light of long wavelengths in the visible spectrum. They are most responsive to light with a wavelength \( \approx 580\,\text{nm} \); which we perceive as red. M cones respond to light of a medium wave length and are most responsive to wavelengths \( \approx 540\,\text{nm} \); which we perceive as green. S cones respond to light of a short wave length and are most responsive to wavelengths \( \approx 430\,\text{nm} \); which we perceive as blue (Wyszecki & Stiles, 1982).

![Normalized responses of S, M, and L cones.](image-url)

Figure 2.1 Normalized responses of S, M, and L cones. (Anon., 2007)
As light reaches the cones the level of ospin in the cones is depleted over time. When all of the ospin in the cone is depleted, the cone becomes unresponsive. Over time the level of ospin in the depleted cone starts to regenerate and the cone starts to become responsive again. It is this depletion of ospin and fatigue of cones that causes afterimages (Anstis, et al., 1977). The colour inversion of afterimages is thought to come from the cones adjacent to the unresponsive cones compensating for the depleted cone; however this has yet to be confirmed.

2.2 Bloom Shaders

Bloom shaders are an eye effect that also attempts to model how the eye behaves. Bloom shaders are used to put a glow around objects that are emitting light. This technique is primarily used in a situation where the user is inside a dimly lit room with a window to the outside where it is much brighter. In this scenario, the light coming in from the window creates what could be described as a halo of light around the window, which blurs the edges of the window.

This effect is created by taking a rendering of a scene and using a threshold to remove areas below a specified luminance level. This image is then blurred to soften the edges and to make it bleed over the edges of the object. This new image is then combined with the original rendering of the scene using a linear interpolation to combine them using the luminance of the original scene at that pixel as the weighting factor (Vanhorn, 2009).

![Figure 2.1 Example of the effect of a Bloom shader on a scene. (Vanhorn, 2009)](image)
Linear interpolation takes two textures, and for each pixel computes the weighted sum of the pixels in the same position in the texture to produce the colour of the output pixel. The weighting of each texture is calculated as:

\[ \text{final} = \text{first} \times (\text{weight}) + \text{second} \times (1 - \text{weight}) \]

This technique is also used to smooth the transition between textures (Gerdelan, 2013).

![Figure 2.3 Example of linearly interpolating two textures. (Gerdelan, 2013)](image)

## 2.3 Adaptation

Adaptation is another eye effect that is commonly used to enhance the user’s sense of immersion. Adaptation models how the eye reacts to changes in the luminance of the scene. The human eye is capable of seeing in both bright and dark environments, however when transitioning between the two there is a period of adaptation that reduces the amount of detail that is visible. When in a bright place the cones and rods in the eye become bleached and less responsive because they do not need to be as responsive when they are exposed to large amounts of light. This is caused by the same ospin depletion that causes afterimages, although not as severe. When one goes from a bright place to a dark place, the ospin starts to regenerate, however until the ospin regenerates enough the eye is unable to make out detail in the scene (Anon., 1987). To emulate this in graphics one could implement a linear interpolation, with a black texture as the second texture and a time dependant adaptation coefficient as the weight.
2.4 OpenGL

OpenGL (Open graphics library) is a low level API used for rendering and manipulating 3D and 2D vector graphics. The main benefit of using OpenGL is the access it gives you to the dedicated graphics hardware of the machine on which it is running. OpenGL behaves as a state machine when programming. When drawing, all of the operations are performed on the Graphics Processing Unit (GPU), this is a dedicated, parallelized piece of hardware that has been optimised to be able to manipulate memory quickly and accelerate the creation of images.

The version of OpenGL used in this application (4.4) has a fully programmable pipeline meaning that the programmer has access control over all stages of the drawing.

2.4.1 Application

The first stage is the application stage. In this stage all the vertices and textures for the scene are initialized. The model matrices, which hold all the transforms that are to be
applied to the vertices of the model, are manipulated in this stage as well. All computation done in the application stage is executed on the CPU. The application stage passes all the vertices and textures to the geometry stage. It is also possible to pass variables to the later stages from the application stage.

### 2.4.2 Geometry

The second stage is the geometry stage. The geometry stage handles the per-vertex operations. Models are defined in model space but in the geometry stage they are transformed into GL world coordinates. The position and normal of each vertex and polygon are multiplied by the transformation matrix. Any shading information associated with the vertex, such as colour or texture coordinates are then passed to the rasterization stage. After the vertices have been transformed they are then projected onto the view plane this can be either an orthographic projection or, if a view frustum has been defined in the application stage, a perspective projection. The positioning, shading and projection are all programmable inside a vertex shader program. The geometry stage also handles clipping, this is a process that checks if an object is inside the field of view and skips any calculations on vertices that are not. This makes the rendering process more efficient. Finally everything is mapped to the screen, this is generally a one to one relationship with the projection. However if there are multiple view ports this is not the case.

### 2.4.3 Rasterization

The final stage is the rasterization stage. Rasterization is the conversion from 3D vertices to pixels on the screen. The vertices are grouped into triangles and the space between them is made into a plane that is made of fragments. Fragments are similar to pixels in the sense that they hold the colour information for a small portion of the triangle, however a pixel can be comprised of multiple fragments. If the vertices have a colour associated with them the colours are then interpolated between the vertices. If instead of having a colour the vertices have a texture coordinate associated with them, then the colour information for the fragments is taken from the texture instead. Any changes to the colour from the lighting are also calculated in this stage. How the colour of the fragments is calculated is programmable inside a fragment shader. All the fragment information is then collaborated into the colour
buffer before being drawn to the screen. This step determines the colour for each pixel, and also resolves visibility of objects behind others.

To avoid all these steps being visible they are all done in a back buffer and only when all the stages are complete the back buffer is copied into the front buffer which is what is shown on the screen.

Figure 2.5 OpenGL rendering pipeline. (Anon., 2012)
2.5 Shader programs

Shader programs or shaders are programs written to control parts of the rendering pipeline. Shaders are written in GLSL (OpenGL Shading Language) a high level shading language with a C-like syntax. It is possible to write shaders with assembly language however most of these are hardware specific, whereas GLSL is not. Using shader programs gives much more control over the drawing process and allows for the calculations to be run on the GPU in parallel. The two main types of shader programs are vertex shaders and fragment shaders.

Vertex shaders perform calculations relating to the position of the vertex. This is very useful when dealing with a particle mesh for representing fluids as all the deformation can be done in the shader with a time dependant function, rather than replacing the vertex positions in memory every time they change.

Fragment shaders perform calculations relating to the colour of fragments. Fragment shaders can be used for image processing as well as lighting calculations and other aesthetic effects.

2.6 Similar Works / Previous Research

Very few papers have been published that explore integrating a model for generating afterimages in computer graphics. The most prominent paper models afterimages on the luminance of a given pixel and over all luminance of the scene. This model uses HDR to exaggerate the contrast of the scene making a purely thresholding approach viable (Ritschel & Eismann, 2012).
Outside the field of graphics afterimages have been studied more extensively, particularly in the field of neuroscience. Anstis, Rogers, and Henry studied the perceived colours and duration of afterimages given a known stimulus. However due to the subjective nature of afterimages the results were too varied between experiments to reach a conclusive model for the duration. While the participants in this study agreed on the colour of the afterimages, this is hard to quantify (Anstis, et al., 1977).

2.7 Possible uses

A model for afterimages could have several applications in the entertainment industry; both as an aesthetic component to increase the user’s immersion and as a gameplay mechanic.

An example of an aesthetic use of the model would be for flash-bang grenades in a first-person shooter game. A first person shooter is a game in which the user has the perspective of a person, typically a soldier, in a combat situation, a well-known example of this type of game is Call of Duty. In these types of games a common weapon used to gain an advantage is a flash-bang grenade. This weapon creates a large flash the temporarily blinds and
disorients the other players. The effect of this is very short lived, only two to three seconds, and just makes the screen white, then fades back to the normal view of the screen. The addition of an afterimage in this kind of scenario could have a huge impact on the gameplay by obscuring part of the user’s vision. This would and dramatically change how the users play the game.

Another example of how afterimages could be used to affect how the user interacts with a game is in a survival horror scenario. Examples of this genre are Silent Hill and Resident Evil. Typically survival horror games are played in a first person perspective in dark environments. In most games players are also given a torch. With an afterimage model in the game things like turning on a torch or other source of light, that would normally have a positive effect would now come with a negative side effect. This would affect how the user plays the game as it would make light a resource that should only be used when it is completely necessary, rather than something you can turn on and off at will with no repercussions.
3. Implementation

This chapter describes the libraries and techniques used in the model for generating afterimages and how they are implemented.

3.1 Render to texture

Render to texture is a procedure where instead of rendering a scene directly to the screen you instead render it to an alternate frame buffer that is off screen, and from there into a texture\(^2\). To do this, a frame buffer must first be created. This involves designating a section of memory to be the new frame buffer. The size of the frame buffer is dynamic and changes with the size of the screen.

Then an empty texture is created. The empty texture is essentially a dimensionless array until dimensions are assigned to it or something is put into it, at which point it takes on those dimensions. When dimensions are assigned a section of memory is assigned to the texture, equal to the dimensions of the texture times the size of the colour information for the texture. To avoid having to copy the texture from the frame buffer to the texture object, it is possible to bind the texture object to the frame buffer so that what gets written to the frame buffer also gets written to the texture.

As well as binding a texture, a render buffer must also be bound to the frame buffer so that the relative depths of the objects are also preserved. Without the render buffer objects would just be drawn over each other in the order in which they are drawn.

![Render to texture process](image)

**Figure 3.1 Render to texture process**

\(^2\) Glossary 7.1.4
3.2 Transparency

In fragment shaders a RGBA colour representation system is used. This system is normalized to 1 instead of 255; however in the frame buffer they are stored as a uchar. R,G and B stand for red, green and blue respectively, varying amounts of these 3 colours can be used to make up thousands of colours.

The A stands for alpha, this is a measure of the transparency of the fragment. If a fragment has an alpha value of 0 it is fully transparent. Alpha culling is a process that sets the alpha value of a fragment to 0. Having non-opaque fragments is not enabled by default in OpenGL as it more costly when computing what should be occluded. Calculating the occlusion with no-opaque fragments is called Alpha Blending.

3.3 Choropleth mapping

A choropleth map is an image that uses colours to denote information. In this program it was used to determine whether an object should generate an afterimage. To do this a shader program was used. The vertex shader calculates the position of the objects as they appear in the scene. The fragment shader takes a colour instead of texture coordinates and draws the objects with this colour. This colour denotes whether an object should generate an afterimage. Any objects that should generate afterimages are given an alpha value of 0, however this is still visible because alpha blending is not enabled for this section of the program. Alone the choropleth map is not enough to generate a realistic afterimage but it is required for the later stages of generating the afterimage.

Choropleth maps are used in graphics for a number of other applications. One example of this is for selecting items. A choropleth map is generated with each object in the scene having a unique colour code. Then when an object is clicked on or hovered over, action can be taken based on the colour of the pixel clicked in the choropleth map.
3.4 Masking

Masking is a process where a detailed image is drawn and then a second image is drawn over the top to obscure some of the detail of the first image. To do this a shader program was used. For masking a very simple vertex shader was used. The object being drawn is a rectangle that fills the viewport so there is no need to calculate translations or perspective projections. The fragment shader used is very simplistic as it has no lighting calculations to do, and only has to texture the rectangle.

![Figure 3.2 Using a choropleth map for selecting objects. (Gerdelan, 2013)](image)

![Figure 3.3 Masking process](image)
3.5 Gaussian Blur / Gaussian Smoothing

Gaussian smoothing is a technique used to smooth and blur an image. It is used extensively in computer vision to reduce noise in images. However it also has many uses in computer graphics.

A Gaussian filter uses a Gaussian function to compute the value of a pixel or fragment based on the colour of its neighbours. To do this in graphics a 2 dimensional Gaussian equation is used to make a kernel. A kernel is a matrix used for image processing that assumes each entry in the matrix corresponds to a pixel. For a Gaussian kernel it is assumed that the centre of the matrix will be the pixel for which we are computing the value. The other entries in the matrix are the weights of the neighbouring pixels. The new value for the centre pixel is the weighted sum of all the pixels covered by the kernel.

![Example of Gaussian blur applied to an image.](image)

*Figure 3.4 Example of Gaussian blur applied to an image. (Anon., 2012)*
4. Model

This section describes the model that was developed to create afterimages in a 3D rendered environment.

![Diagram of afterimage generation procedure](image)

Figure 4.1 Graphical representation of afterimage generation procedure

4.1 Scene

The scene is fully rendered with textures and using a phong shader into framebuffer0 which is the frame buffer that gets drawn to the screen. It is not possible to bind a texture to this frame buffer. This means that the data inside the frame buffer has to be copied into a texture to be able to use it later when making the masked image.

Phong shading is a lighting technique that softens the transition between the triangles in the model, this is a per fragment shader. In a Phong shader, everything is first coloured with an ambient light. The fragment’s diffuse intensity is then calculated, this is the intensity of the light times the dot product of the surface normal with the vector from the surface to the light position as a scaling factor. The reflection of the light off the surface to the user is then calculated. This is the intensity of the light shining on the surface, scaled buy the surface reflectance. This is then scaled by the dot product of the reflected ray from the light source through the surface normal, and the position of the viewer relative to the object. These 3 factors are summed to give the light intensity and then multiplied by the colour of the fragment to give the final output.
4.2 The choropleth map

The choropleth map allows for simplification of a lot of the issues presented in the model. The afterimage generation is largely an image processing problem that is dependent on what the user is looking at in the scene. The choropleth map uses the transformations used to arrange the scene when drawing the models, so it is an exact replica of the scene. The map is drawn off screen into a frame buffer and the texture bound to the frame buffer is used to access the map.

To do this without a choropleth map would require having a ray tracing\textsuperscript{3} system that would have to check what objects are in front of the user and then determine whether they have been designated as an object that should generate an afterimage. By sampling the choropleth map it is possible to determine whether an object should generate an afterimage. This process is also faster than a ray tracing method because it can very easily check the points corresponding to objects in the choropleth map and fully utilise the GPU to do this in parallel.

The other benefit of having the choropleth map is it can be used to limit what generates an afterimage. If a purely threshold based method was used then it would be possible for objects that would not normally generate afterimages to generate afterimages eg. White

\textsuperscript{3} Glossary 7.1.6
walls. This additional level of control allows the designer to use this method for artistic purposes as well as for realism.

![Figure 4.3 Choropleth map output](image)

### 4.3 Masking

The masking process allows for the afterimages to contain detail from the object that is causing them. To create the mask a rectangle the size of the screen is used as a surface to render both the full scene and the choropleth map to. For this step of the process, alpha blending is enabled. To allow for the rectangle to fill the screen, an orthographic projection is used. In this case the order of the texture being used is important as the rectangle is not being scaled or translated so the rasterizer will draw objects that are drawn later over ones which have already been drawn. The result is a texture of what is in the user’s field of view where only the objects that should generate an afterimage have detail and the rest is black.
4.4 Blink Model

As afterimages are an eye effect they are affected by blinking. When one blinks while perceiving an afterimage, the afterimage appears stronger. As this is one of the more noticeable things about afterimages a blink was added to the model.

To model the blink a frame counter was implemented to set the frame rate to 60Hz so that counting the number of frames would be a reliable way of measuring time. When the cycle reaches a blink frame, a translucent black texture is drawn to a rectangle that fills the screen and the afterimage texture is made fully opaque. The result is a dimmed version of the scene with the afterimage very prominent. After the blink frame the transparency of the afterimage is increased, extending its duration and making it appear stronger.

The rate of blinking was modelled on the average rate at which people blink, 2.8s (Anon., 2008), with a randomised variance to make it seem more natural. The duration of the blink was partially modelled on the average duration of a blink for people. The average duration of a blink is between 0.1ms and 0.4ms (Schiffman, 2000), however this duration is from eyes open to open. While blinking the photoreceptors in the eye continue to send signals depicting the last thing seen, which is why we do not notice blinks as more than a brief...
flutter if at all (Gawne & Martin, 2000). It was this barely perceivable flutter that was emulated in this model.

![Figure 4.5 Blink frame output.](image)

### 4.5 Afterimage Source

The base afterimage is the output from the masking process; because of this the same frame buffer is used for both. The processed afterimage is stored in a texture, separate from the one that is bound to the frame buffer that contains the masked texture. This means it persists when the masked texture is updated.

This section of the process takes advantage of OpenGL acting as a state machine. The frame buffer that the afterimage processing is written to is the same frame buffer that the masked image is written to. This means that the process does not need to be altered for the first time an afterimage is generated. Every time an afterimage is generated the afterimage texture is drawn over the contents of the frame buffer, however if it is the first time an afterimage is being generated, this texture is empty and does not change the contents of the frame buffer. The contents of the frame buffer then get copied into the afterimage texture.
4.6 Conditions for Generation

As this application is focused on negative afterimages the trigger for generating an afterimage occurs when the user stops moving for a period of time. While the user is not moving the afterimage texture gets passed through the post processing shader repeatedly. This shader blurs the afterimage using a Gaussian blur, with a 5x5 kernel and $\sigma = 0.8$. This shader also uses alpha culling to remove any black areas of the afterimage texture. While the user is not moving this post processed texture is copied into the afterimage texture. This also starts the afterimage being drawn to the screen.

The transparency of the afterimage is governed by a variable that increases over time while the user is not moving. This models spin depletion in the eye. The rate at which the variable is increased depends on the maximum intensity of the afterimage but will always take 15s to build up to a full afterimage in this model. The intensity of the afterimage is the maximum opacity of the afterimage as this is what will make it appear stronger. The 15s build up time was taken from the research paper Interactions between Simultaneous Contrast and Coloured Afterimages (Anstis et al.) as this is how long the subjects were exposed to the stimulus.

While the user is not moving the afterimage starts to build up on the screen and become more noticeable as the opacity of the afterimage is increased. As the afterimage is drawn to the screen it is passed through the same post processing filter and is blurred again with the same Gaussian kernel.

4.7 Decay

When the user starts to move the afterimage opacity starts to be reduced. The decay is primarily linear however when a blink frame occurs the opacity is increased making it a decaying saw tooth function (Fig 4.6). This is done because as one blinks the afterimage becomes more prominent (Suzuki & Grabowecky, 2003). When the user starts to move the colour of the afterimage becomes inverted. This decision was made to increase the aesthetics of the model, as the inversion is not noticeable while still looking at the stimulus.
When the afterimage has fully decayed the texture remains unchanged, however it is not visible as it is being drawn fully transparent. The afterimage texture only gets reset when a new afterimage is generated. While the user is moving the masked image in the frame buffer is deleted.

### 4.8 Compound Afterimages

The above method describes how a single afterimage is generated and only allows for one afterimage to be generated at any time. However this is not what happens in reality. It is possible to experience a second afterimage while already experiencing one. In this scenario the old afterimage continues to decay while the new one becomes stronger.

Not clearing the frame buffer with the masked image and drawing the current afterimage texture on top allows for this to be done. These two textures together give a base texture to work from to replicate the effect of one afterimage decaying while the other is generated.

To have one afterimage decaying as another is being generated it is essential to be able to tell which parts of the texture are which. To do this the choropleth map was used. By sampling from both the afterimage texture and the choropleth map at the same time it is possible to determine which parts of the afterimage texture correspond to the new afterimage. Areas of the afterimage texture that correspond to a black area of the
choropleth map are the old afterimage. The new afterimage is handled in the same way as it would be if it was not a compound afterimage.

As the old afterimage decays, it eventually disappears. To replicate this, the afterimage texture is changed to the masked texture. This results in any old afterimages being culled from the texture.

The afterimage texture is drawn to a rectangle that fills the screen; this is drawn over the fully rendered scene. This could have been done using linear interpolation, as is done with bloom shaders. However, given that afterimages exist only in the eye, blending the afterimage with the scene would detract from the impact of the afterimage.
5. Results

The resulting program has achieved the goals outlined earlier in the report:

- The model runs in real time.
- The afterimage model, wherever possible, emulates the behaviour of an eye.
- The model is easily modifiable and would be suitable for either realistic or artistic purposes.
- The model is not dependent on HDR.

5.1 Verification of results

5.1.1 Running in real time

In this application the frame rate was set to 60Hz for timing various elements of the model. With the frame rate limiter removed the average frame rate was slightly over 60Hz, approx. 62Hz. However it is worth noting that this was with writing the time between frames to a file for every frame.

![Frame rate over 800 frames](image)
5.1.2 HDR dependency

HDR images would be essential for a model that relies on thresholding to generate afterimages. However the choropleth map used in this application removes the need for HDR images. As such none of the images used in the application were HDR images.

5.2 Runtime Stills

The following images show afterimages generated by the program in various states.

Figure 5.2 Afterimage being generated.

Figure 5.2 shows an afterimage being generated with a lamp as the stimulus. The lamp shade has become brighter and the afterimage has started to bleed over edges of lamp.
Figure 5.3 Detailed afterimage of the painting on the wall.

Figure 5.3 shows an afterimage shortly after being generated. As show the afterimage keeps the detail of the stimulus.

Figure 5.4 Inverted afterimage of the painting on the wall.

Figure 5.4 highlights the inversion aspect of the afterimages in this model. As shown the blue is inverted to yellow, the red to cyan and the green to magenta.
Figure 5.5 shows a compound afterimage being generated. The sections of the afterimage that are over the wall and window frame are decaying while the areas over the window panes are becoming stronger.

Figure 5.6 shows a resulting compound afterimage almost fully decayed. In this case both afterimages were generated in quick succession; however the lower section has decayed more than the top.
6. Conclusions and Further Work

6.1 Conclusions

The afterimage model developed in this program successfully emulates how the eye reacts to stimuli that cause afterimages. In this model the frame rate was capped at 60Hz, which it easily achieved, meaning this could be integrated into a more comprehensive system without impeding the performance. The use of the choropleth map adds an additional level of control over model that would not be possible in a purely threshold based model.

6.2 Limitations

6.2.1 Subjectivity

Given the highly subjective nature of afterimages a comprehensive model that perfectly emulates afterimages is impossible. The intensity and duration of an afterimage, as well as the time it takes to generate an afterimage is dependent on too many things for a fully realistic, comprehensive model to be feasible.

Despite this an aesthetic model that emulates responses in the eye that cause afterimages has been achieved. While modelling the actual time it takes to generate an afterimage is not practical, it is possible to make an approximation to it. The maximum intensity of the afterimage also has an effect on the amount of time it takes for the afterimage to be fully generated. Again this is not entirely practical to measure. To overcome this, the maximum intensity of the afterimage has been made variable so that depending on the end use of the application the maximum intensity can be raised or lowered to suit.

The time taken for an afterimage to decay is something that can easily be measured. However the fade time is dependent on the stimulus and is also subjective. A library of stimuli and environmental conditions would be required to model all possible stimuli accurately; this would still have the problem of being subjective however. Alternatively an approximate universal model could be used instead.
6.2.2 Perception of Afterimages

As an afterimage is a phenomenon that occurs in the eye coming up with a model that accurately portrays this effect on a screen is challenging. This is because we are used to seeing these effects directly in front of our eyes, and seeing it on a screen that is in front of us looks unusual. One of the things done to try to make it feel less a part of the scene was not mixing the afterimage texture with one of the scene. This gives it a sense of being separate from the rest of the scene as well as being closer to the user.

Integrating the model on different hardware could be another solution to this problem. The most obvious example of this would be the Oculus Rift headset because it is a head mounted display. This would remove the unusual feeling of seeing the afterimages on a screen in front of the user.

6.3 Further Work

6.3.1 User Experiment

Ideally an end user experiment would be done to evaluate the model. In order to accurately test this model a photorealistic scene, with high definition models and textures, would be required. A game environment would be ideal for testing this model; however scenes of this nature are made by a team of artists over a number of months. As such a scene of this quality was beyond the scope of this project.

A suggested experiment would be to change the subjective parameters of the model and have the users evaluate how realistic the model appeared. This would first be done by fixing the time to generate an afterimage and the decay time while varying the maximum intensity. This would then be repeated for both the generation time and the decay time, using the participant’s preferred values from the earlier stages of the experiment as the fixed value in the later stages. These results would then be tabulated and a universal model would be derived from the results. As a further step this universal model could then be shown to another group of participants to evaluate the new universal model.
6.3.2 Focusing Model

The addition of a system that took into account where in the scene the user is focusing, to restrict what objects should be included as a stimulus, could benefit the model. A subsection of the screen could be used to do this. Alternatively an eye tracking system could be added. However computer vision systems are computationally expensive and this type of solution could have a negative impact on the performance of the application containing the afterimage model.

6.3.2 Randomised Deformation

In the real world afterimages deform randomly over time. This model continues to deform the afterimage; however the deformation is linear. A model that includes a randomized blur could increase the realism of the model.

6.3.4 Oculus Rift

Integrating the model with the Oculus Rift could potentially solve the issue of the afterimages appearing on a screen some distance away from the user. As the Oculus Rift does not naturally cause afterimages, this model would be ideal for increasing the sense of immersion when using this hardware.
7. Glossary and Technology used

7.1 Glossary

7.1.1 Vertex

A vertex is a three dimensional vector that is used to describe the location of points in a model in three dimensional space.

Vertices are defined in three dimensions, however to be able to manipulate the vertices with transformation matrices we need to use the homogeneous representation of the vertices as the transformation matrices are 4X4 matrices.

7.1.2 Model

A model, also known as a mesh, is a collection of vertices that define an object. To give the model a surface rather than being a collection of points, polygons are interpolated between the vertices. These polygons are usually triangles.

Figure 7.1 Model with a vertex selected
7.1.3 Rendering

Rendering is the process of generating a two dimensional representation of three dimensional environment or model to be displayed on a screen.

7.1.4 Texture

A texture is a data structure used to hold images that are used to add detail to models. Textures are made by unravelling the model to a two dimensional collection of polygons, a texture is then drawn over these polygons. This is how the texture will look when applied to the three dimensional model. The positions of the vertices in the texture image are known as texture coordinates. Texture coordinates are used in the fragment shader as a reference to determine which areas of the texture should be applied to each polygon.

![Figure 7.2 Example of a texture with vertices overlay](image)

7.1.5 HDR

HDR stands for high dynamic range. HDR images are images that emulate having a much higher dynamic range, 50,000 : 1 contrast ratio, than is able to be displayed on a monitor, approx. 750 : 1 contrast ratio.
7.1.6 Ray Tracing

Ray tracing is an illumination technique that follows the path of a ray of light through a scene and simulates the rays interactions with objects in the scene. This is used to cast reflections and shadows. Ray tracing is a very computationally expensive process.

![Ray Tracing Diagram](image)

Figure 7.4 Ray tracing example (Anon., 2008)

7.2 Technology used

1.4.1 C++ programing language.

C++ is used almost exclusively for multi-platform graphics applications because of its speed versatility. This made it the obvious choice for this application.

1.4.2 Visual studio 2010
Visual studio was the IDE of choice because of its powerful debugging tools and past experience with using it.

1.4.3 Open Graphics Library

OpenGL is the most widely used graphics library because of its fully programmable pipeline, multi-platform nature and graphics hardware acceleration.

This program uses OpenGL 4.4 and GLSL 4.0

1.4.4 GLUT

The OpenGL Utility Toolkit is an API that manages window creation as well as external hardware input.

1.4.5 maths_funcs library

This open library was written by Dr. Anton Gerdelan and provides functions for creating and manipulating vectors and matrices. This library is particularly useful as it uses a similar naming convention to GLSL.
Bibliography


Appendix A – Files on CD

1. Afterimage Model.zip

Contains the complete source code of the afterimage modelling program, including shader files and assets used.