**Matrix Effect**

**Project**

**Student**

Elinor Feller

**Supervisors**

Boaz Sterenfeld

Yaron Honen

# 

# Table of Contents

[**Introduction**](#_ssii9i6ybu0y) **3**

[**Environment**](#_6d4v7hz6jnwu) **4**

[**System overview**](#_tslzmetqgkg5) **5**

[Shader](#_oqvuoq4zeknj) 5

[Unity Script](#_wlem7xquzmdh)s 5

[Unity Scene](#_yj0wtmcnxzma) 5

[**Development Process**](#_alh3gt44f1a7) **6**

[**Results**](#_nee6s7mog4j5) **11**

[**Conclusions**](#_bj100nentvp9) **12**

# 

# Introduction

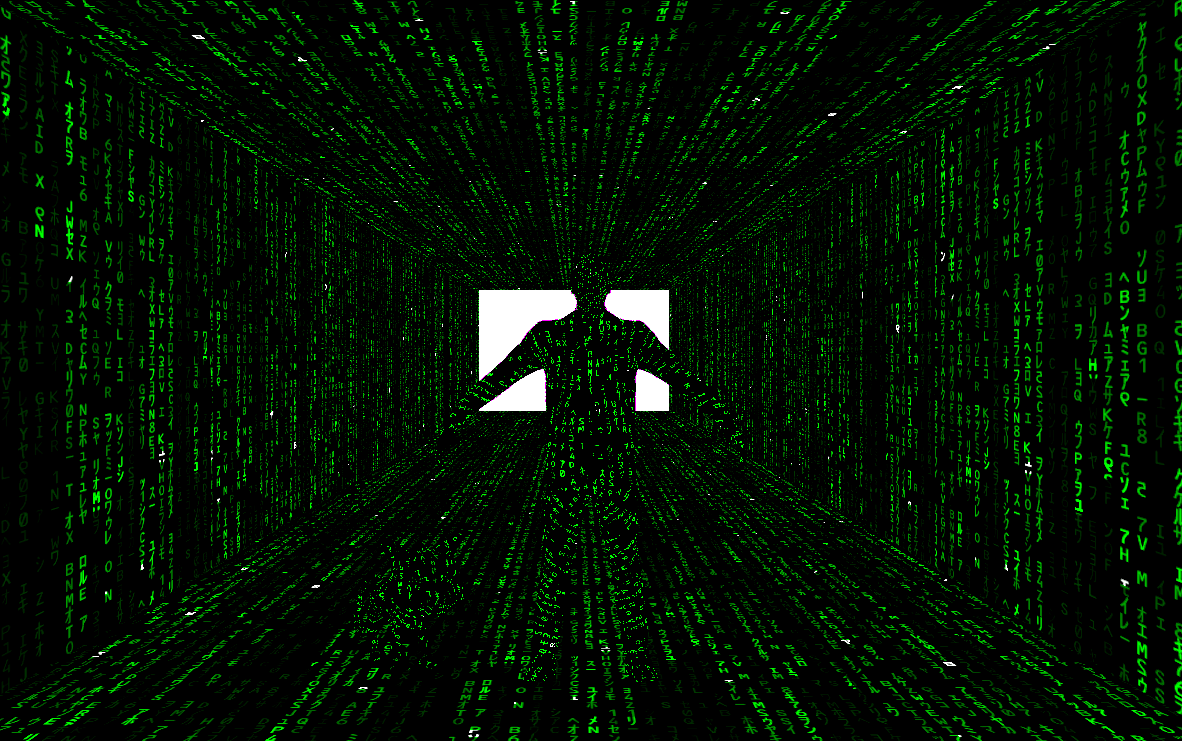
The matrix effect was introduced in *The Matrix* film that was released in 1999.

It was one of many novel new visual effects that were introduced in this film.

In this project, I try to replicate this effect using modern tools such as unity, and more specifically unity shader graph.

In order for the effect to be general, I decided to implement it as a unity shader, so anyone will be able to easily use it on any 3D object. I also added a few configurations such as brightness in order to support complex scenes with many entities, some are more complex than others and thus brighter.

I feel that implementing this shader in a modern environment could help many future projects, and more personally, pointed the ability to create any graphical effect with reasonable effort.



*the resulting effect in an example scene*

# Environment

I created the effect using a few main components.

#### Unity Engine

The entire project was created in the Unity environment.

**Unity** is a cross-platform game engine developed by Unity Technologies. As of 2018, the engine has been supporting 27 platforms. The engine can be used to create both three-dimensional and two-dimensional games as well as simulations for its many platforms.

#### Visual Studio

In order to create more complex scenes I used Visual Studio in order to code C# scripts.

**Microsoft Visual Studio** is an integrated development environment (IDE) from Microsoft.

# System overview

The system created is a complex scene including three main parts - GameObjects, Shader and Unity scripts. I will now go over each one of them in more details.

### Shader

A **shader** is simply a program that runs in the graphics hardware and tells the computer how to render each pixel. These programs are called shaders because they're often used to control lighting and shading effects, but they also can be used to create other special effects.

In this project, I created the **Matrix Effect Shader**. When one wishes to deploy the effect on a specific object, it can just use it with no other assumptions.

As can be seen, this is a very general approach which can be utilized to many future projects.

In order to further improve the generalization of the results, I also created 2 exposed properties for the shader: (Can be changed from the Unity interface)

1. **Tilling** - How many times to duplicate the falling code. In practice, this controls the size and proportions of the effect on a given object. This allows for the effect to look good on many different objects in many setups.
2. **Brightness** - How bright will the effect be. This allows us to create complex scenes in which different objects with different levels of complexity are with different brightness as seen on the original effect in the movie.

### Unity Script

Unity allows us to create our own Components using **scripts**. These allow us to trigger game events, modify Component properties over time and respond to user input in any way we like.

I created 3 scripts:

1. **ShaderTiming** - This script modifies the object shader or material. It does it only after several seconds in which the objects are shown with the standard shader. This is similar to the way the effect is shown in the movie.
2. **MoveFarward** - This script moves the character forward in the direction of the camera.
3. **MoveRight**  - Same as MoveFarward, in a different direction.

### Unity Scene

I created an example scene, in order to show the power of the created shader.

It also functions as a reference implementation for the more complex effect on various GameObjects, including starting it only after few seconds.

# Development Process

Starting the project, I had no prior experience with Unity or any knowledge regarding it.

After watching basic tutorials about Unity and understanding the basic ideas of it, I tried to think on what will be the best way to create and present the matrix effect using the Unity environment.

After exploring the Unity’s Particle System and trying to combine it with animations from sprite sheets, I figured out it wasn’t suitable to overlaying over objects, I started to realize that I will probably need to create a shader with the Matrix effect. I wasn’t familiar with what exactly shader is and started to learn about it and about related concepts like working with **textures**, **UV mapping**.

I found out about VideoPlayer. VideoPlayer allows us to use video as a **movie texture** in standard shaders.



*Hallway built using plane objects with VideoShader.*

This gave a result, which looked great at first glance, but when I started to build more complex scenes with many objects, it had performance issues and kept getting stuck.

Another issue was the lack of flexibility.

The effect quality was affected deeply from the shape and size of the object it covered.

After exploring the unity resources for a more general solution, I decided the only way to get a satisfying result is to build my own shader.

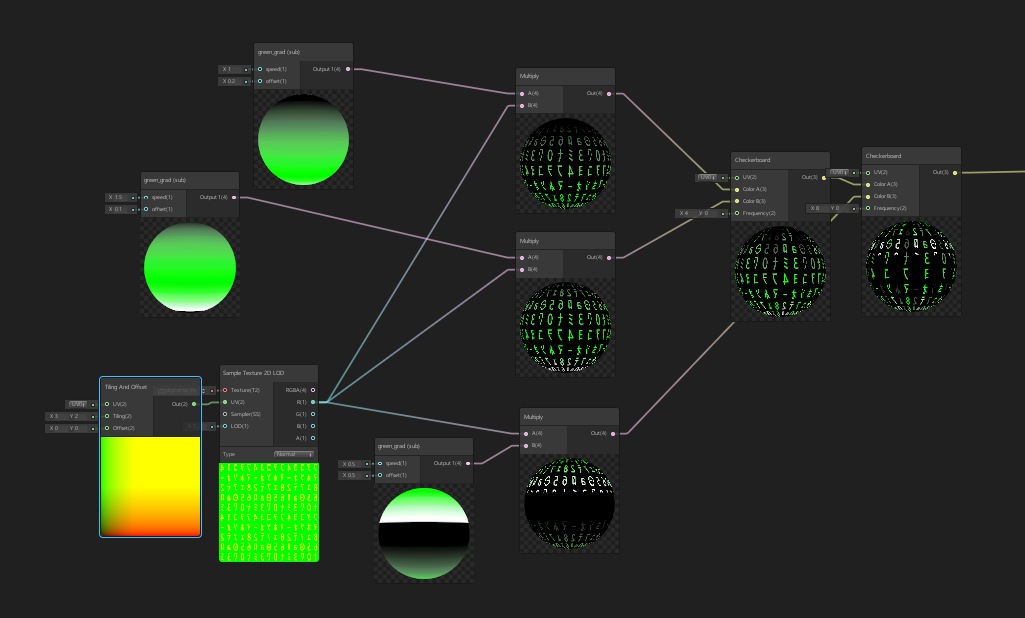
I discovered that there is a new solution for that need which is currently in beta version, it is called **Shader Graph**.

It allows us to create shader by building a graph which described the shader.

I decided to learn how the shader graph works and use it, in order for it to be easy for future use, and easily expanded.

I started learning and thinking of ideas on how can I implement the Matrix effect using the available nodes and Graphics principles.

My first approach was to try and create letters picture and combine few version of it which runs in different speeds.

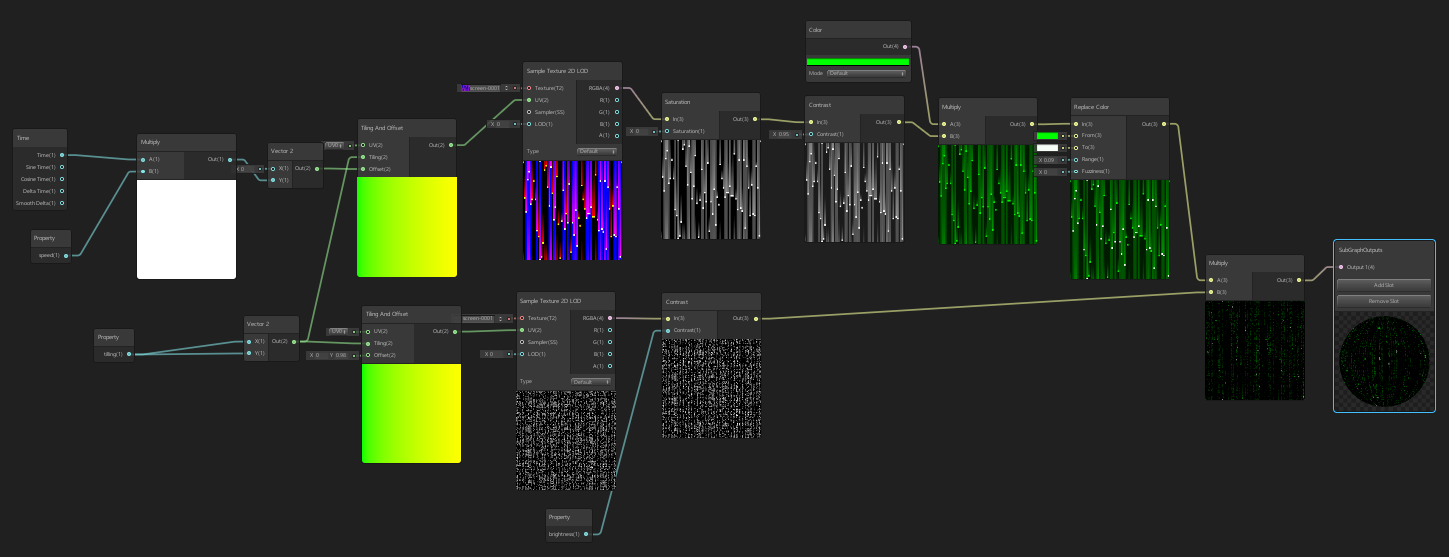


*The resulting graph of this approach*

The only problem was that it didn’t look good enough as you can see:

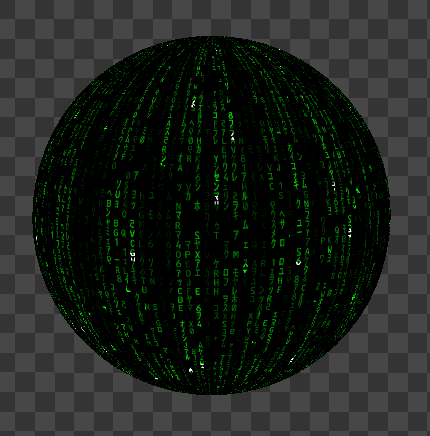
I tried to look on the internet for any similar projects to get ideas until I found a blog post about a similar effect that was created by a shader artist in Unreal (<https://medium.com/swinginc/typography-meets-shaders-making-matrix-rain-shader-with-processing-and-unreal-70097aa5991f> ).

Combining the ideas derive from this tutorial, with many other resources I found online, and many hours of experimenting led me to a more complex graph, but one with much better results.



*The more complex and better-resulting graph*

For further improve the results, I created this as a sub-graph, added property called speed to control the falling speed and combined few of this sub-graphs to get the final result.

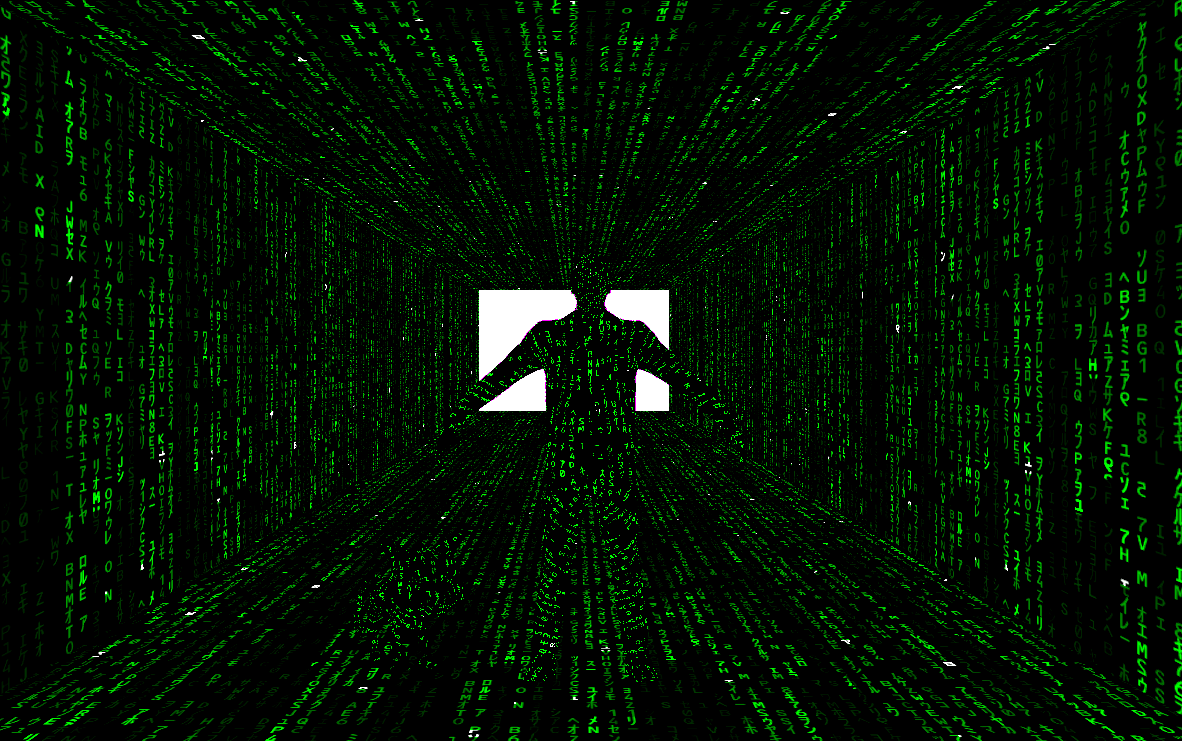


*The final shader result*

Now I wanted to create a more complex scene with a few objects in different sizes and complexity. In order to replicate the effect from the movie in which more smart objects are brighter, I exposed 2 properties - tilling and brightness.

For the brightness and tiling control, I had to change the graph but it resulted in much more flexible and general shader.

After many tunings and changes the results finally looks good:



*the final shader result*

Now that I have shader from which I am satisfied, I tried to create a scene which starts regular (without the matrix effect) and then the matrix effect starts, the same as in the movie.

I couldn’t find a way to do it from the unity interface, which led me to create BehaviroulScript.

Those scripts can be coded both in JavaScript and in C#. I’m not familiar with any of those languages, but from the examples I saw, it seemed more common to use C# so I chose to try and implement it in C#.

As one can imagine, building script in an environment I’m not familiar with, in a programming language I don’t know, was not easy nor fast. It took many tries and dead ends to get to a final working version.

In my final script version, I extended the **MonoBehaviour** class. I predefined specific material for the human and the cat with different properties. I then put those materials in the Resources folder so I will be able to load it during runtime.

For all the other objects I created new material in runtime and assigned it the matrix shader I created.

Now I decided to add some animation to move the human and pet in the scene. This will empathize the matrix shader and will show its full power when it is rendered on moving object.

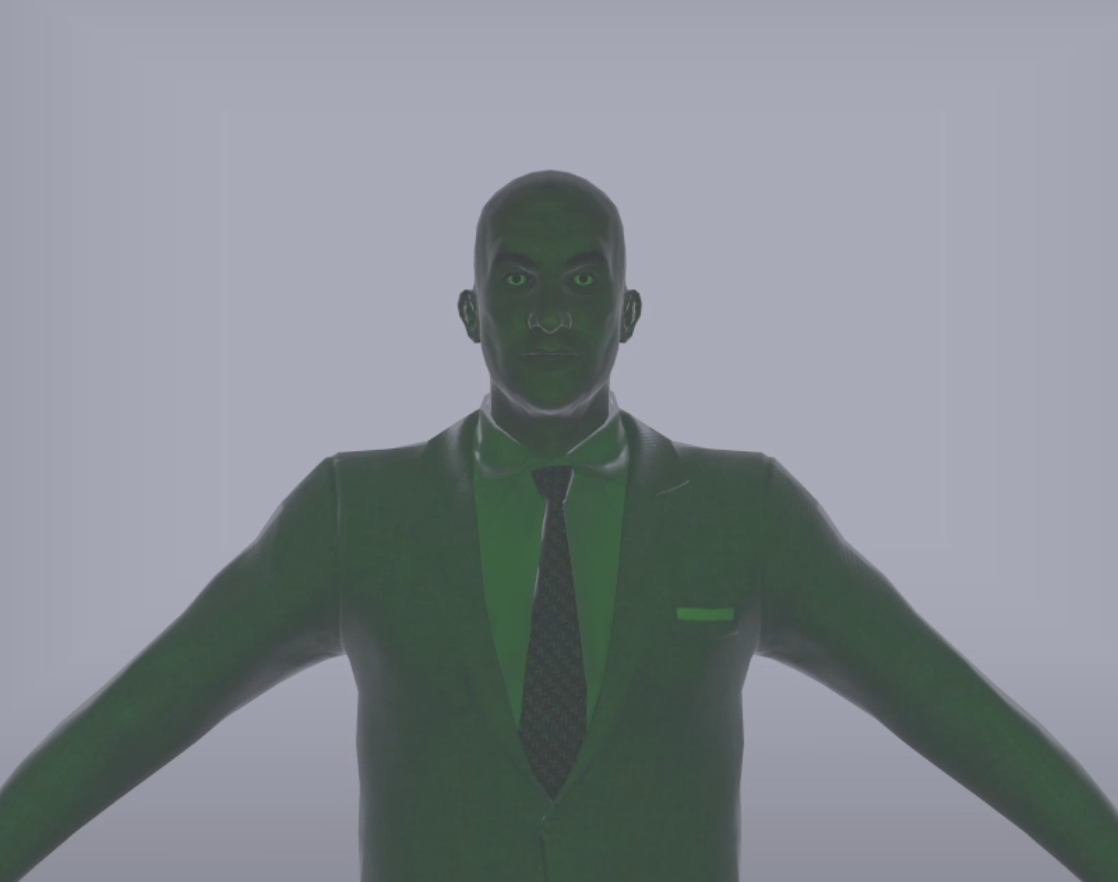
I changed one of the characters and again built a script for moving the objects along with activating the animation.

# Results

The results in an example scene in which the objects start with a normal shader, and after a few seconds, the matrix effect starts.

This scene contains a general shader which can be applied in many scenes and scenarios. It is also extendable and easy to understand, as it was built as a graph.

The effect can be seen at full in this [video](https://youtu.be/uuKwhQNq98M), and below there are before and after pictures.



another example:



# Conclusions

This project has exposed me to the world of graphical modeling in general, and unity in specific, with its great power and flexibility to create complex 3D objects and effects.  
  
Also, I’m glad that I had the option to learn about the world of shaders, that looks very complicated at first glance, and to use a new feature that was just recently added to Unity, to make this world more user-friendly.

# Website

<https://elinorfe.wixsite.com/matrixshader>