

Описание предполагаемого способа решения

Перед тем, как перейти к описанию методов, нужно рассказать об общей структуре используемого шейдера т.к. все решения будут опираться именно на его структуру.

```
Shader "MyShaderName"
{
    Properties
    {
        // свойства материала
    }
    SubShader // сабшейдер для определённого железа (можно определить
    директивой компиляции)
    {
        Pass
        {
            CGPROGRAM

            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            // проход шейдера
            struct v2f {
                float4 pos : SV_POSITION;
                fixed3 color : COLOR0;
            };

            v2f vert (appdata_base v)
            {
                v2f o;
                o.pos = UnityObjectToClipPos(v.vertex);
                o.color = v.normal * 0.5 + 0.5;
                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                return fixed4 (i.color, 1);
            }
            ENDCG
            // для некоторых эффектов может понадобится несколько проходов
        }
    }
}
```

Такие директивы компиляции как

#pragma	vertex	vert
#pragma fragment frag		

определяют какие функции шейдера компилировать в качестве вершинного и фрагментного шейдера соответственно.

Функция *UnityObjectToClipPos* — это вспомогательная функция Unity (из файла *UnityCG.cginc*), которая переводит вертексы объекта в позицию,

связанную с камерой. Без неё объект, при попадании в зону видимости камеры, будет рисоваться в координатах экрана вне зависимости от положения трансформы. Так как первоначально позиции вертексов представлены в координатах объекта.

```
struct v2f {  
float4 pos : SV_POSITION;  
fixed3 color : COLOR0;  
};
```

Это определение структуры, которая будет обрабатываться в вертексной части и передаваться в фрагментную. В данном случае в ней определено, чтобы из меша забирались два параметра — позиция вершины и цвет вершины.

Далее будут рассмотрены два метода создания эффекта капель дождя на окне.

Simple Raindrops

Основная идея данного шейдера заключается в циклическом уменьшении/увеличении небольшой псевдослучайной области изображения с последующей рефракцией данной области. Таким образом, создается иллюзия того, что на окне появляется капля, а затем плавно уменьшается и исчезает с окна. Данный эффект описывается функцией зависимости размера капли от времени:

$$\text{SizeDrop} = \max(0.0, 0.5 - \text{frac}(\text{intensity} * t))$$

Где команда `frac()`, возвращает дробную часть от исходных данных.

Rain Window

Идея этого шейдера состоит в том, чтобы создать динамическую сетку, которая будет содержать в себе набор капель. Каждая капля будет менять свою позицию в пределах своей сетки согласно некоторой функции движения, тем самым создавая ощущение стекания по окну. Для разделения на отдельные зоны для капель используется ранее упомянутая команда `frac(uv)`, которая возвращает дробную часть от исходных `uv`-координат.

Далее с помощью команды smoothstep создается сама капля: данная операция производит гладкую интерполяцию Эрмита между двумя значениями, передаваемые в качестве параметров.

Теперь требуется, чтобы полученные капли могли двигаться в пределах своей “коробки”. Более того, их движение должно быть более-менее реалистичным и цикличным. Для этой цели используется функция зависимости положения на оси ординат от времени:

$$y = -\sin(x + \sin(x + \sin(x) \cdot .5))$$

График такой функции представлен на рисунке 2.

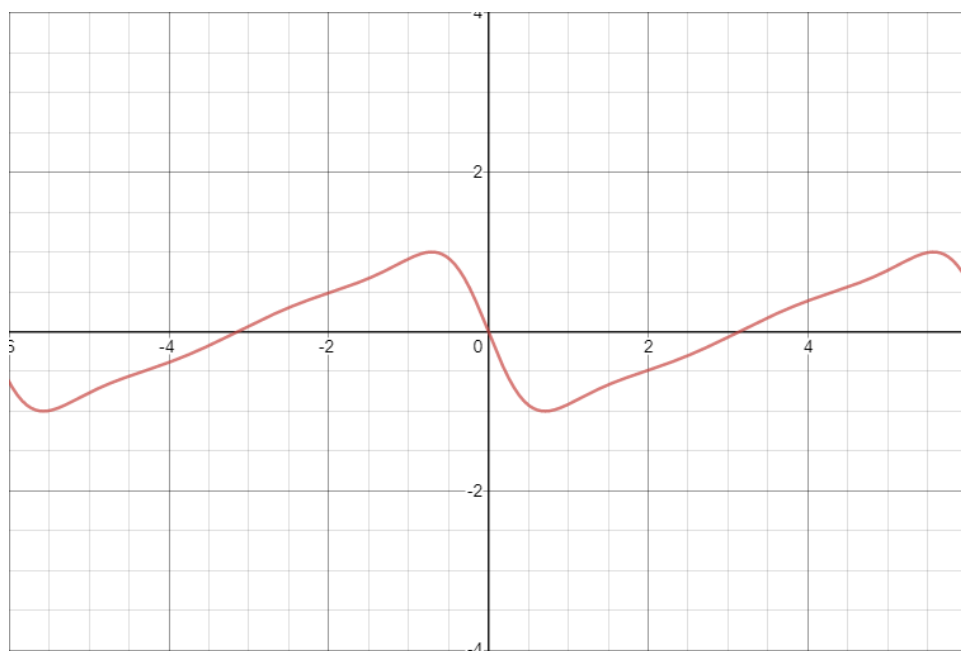


Рис. 2

Исходя из приведенного графика, движение капли можно описать следующим образом: капля быстро стекает вниз, а затем медленно поднимается вверх.

Теперь осталось добавить скорость смещения самой сетки по оси ординат. Для обеспечения правдоподобности, скорость смещения должна быть равна скорости капли в момент её подъема наверх, это создаст эффект прилипания капли к окну.

Помимо самой капли, можно сделать “хвост”, который следует за её движениями. С помощью ранее описанных команд `smoothstep` и `frac()` создаются несколько маленьких капель позади основной.

Сравнение методов

Краткое сравнение предложенных методов создания визуального эффекта: (таблица)

Вид шейдера	Дополнительные улучшения	Основная идея алгоритма	Причина использования
Simple Raindrops	Нет	Циклическое сжатие/расширение капли	Простота использования
Rain Window	Размытие окна	Циклическое движение капли вместе с содержащей её сеткой	Реалистичность эффекта

Таблица 1

Как видно из Таблицы 1, у методов есть как преимущества, так и недостатки. В первом случае, метод достаточно легко реализуем и не требует особых познаний, однако сам результат нельзя будет назвать достаточно реалистичным. Второй метод, наоборот, требует более глубоких знаний в работе с шейдерами, но при этом создаваемый им визуальный эффект будет выглядеть более правдоподобно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Hohn Kessenich, Dave Baldwin, and Randi Rost. 2014. The OpenGL® Shading Language (Version 4.50). <https://www.opengl.org/registry/doc/GLSLangSpec.4.50.pdf>. (дата обращения: 10.12.2019).
2. Khronos Group, Inc. 2009. ARB_shader_subroutine. https://www.opengl.org/registry/specs/ARB/shader_subroutine.txt (дата обращения: 10.12.2019).
3. Michael D. McCool, Zheng Qin, and Tiberiu S. Popa. 2002. Shader Metaprogramming, 2010.
4. Natalya Tatarchuk. 2006. Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows. In Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games (I3D '06). ACM, New York, NY, USA, 63–69
5. Ivan Selesnick .Least Squares with Examples in Signal Processing// NYU-Poly, 7 марта, 2013.