Инженерная школа информационных технологий и робототехники

Направление 09.03.01 «Информатика и вычислительная техника»

Отделение информационных технологий

Отчёт по теме

**«Разработка игровых приложений на Unity»**

по дисциплине «Творческий проект»

Выполнил:

студент гр. 8В11 \_\_\_\_\_\_\_\_\_\_\_\_\_ Топчеева Д.А.

\_\_\_.\_\_\_.\_\_\_\_\_\_

Проверил:

Ассистент отделения ИТ

оценка (до 30 баллов): \_\_\_\_\_\_\_\_\_\_\_\_\_

к защите допускаю \_\_\_\_\_\_\_\_\_\_\_\_\_ Коровкин В. А.

\_\_\_.\_\_\_.\_\_\_\_\_\_

**Содержание**

[Введение 3](#_Toc138120850)

[1. Анимация в Unity 4](#_Toc138120851)

[1.1. Покадровая анимация 4](#_Toc138120852)

[1.2. Скелетная анимация 5](#_Toc138120853)

[1.3. Шейдеры 6](#_Toc138120854)

[2. Разработка Игрового приложения 10](#_Toc138120855)

[2.1. механика Движения объекта по заданному маршруту 10](#_Toc138120856)

[2.2. Анимация объектов 14](#_Toc138120857)

[Заключение 18](#_Toc138120858)

[Список источников: 19](#_Toc138120859)

[Приложение 1 21](#_Toc138120860)

[Приложение 2 21](#_Toc138120861)

[Приложение 3 24](#_Toc138120862)

[Приложение 4 26](#_Toc138120863)

# Введение

Анимация и движение в целом играют важную роль в восприятии человеком какого-либо объекта. Это делает создаваемый продукт более живым и запоминающимся. Поэтому создание анимации и придание движения объектам важно при разработке игр.

Главной проблемой анимации при разработке игрового приложения является его времязатратность. На отрисовку анимации может уходить много времени, из-за чего может быть недостаточно времени на разработку других составляющих приложения. Для решения этой проблемы иногда используют шейдеры, если есть много объектов с относительно простыми движениями. Это позволяет значительно сократить время на создание анимации. Но для более сложных движений объектов приходится использовать покадровую или скелетную анимацию, поэтому необходимо знать функции Unity позволяющие работать с этим.

В данном семестре планируется добавить в игровое приложение на Unity анимацию и создать шейдер и программу для уменьшения затрат времени на создание движения для простых объектов.

Целью данной работы является:

Изучение методов движения объектов и создания анимации для них в Unity.

Для достижения поставленной цели в творческом проекте решаются следующие задачи:

1. Изучение инструментов Unity для работы с объектами и анимацией.
2. Проектирование и разработка игрового приложения в Unity.

2.1. Создание новых элементов виртуальной окружающей среды.

* 1. Добавление анимации.

# Анимация в Unity

# Покадровая анимация

Покадровая анимация - самый простой в понимании вид анимации. Для ее создания необходимо нарисовать все кадры отдельно с небольшими изменениями, благодаря чему после объединения их всех создается иллюзия движения.

Преимущества:

* Легкий в понимании и реализации
* Низкая нагрузка на процессор
* Движение выглядит более «живым»
* Возможность создавать сложные движения

Недостатки:

* Тратит много памяти, т.к. хранятся данные о каждом кадре
* Занимает много времени на прорисовку каждого кадра

В Unity для создания такого вида анимации нужно к объекту прикрепить Animator, куда вносится Animator Controller, и уже в нем создается анимация. В окно Animation вносятся все нарисованные спрайты в нужном порядке, задается частота кадров в секунду (рис.1), и все готово.

Проблема в использовании данного типа анимации во время разработки приложения это то, что на ее создание уходит очень много времени, из-за чего время, уходящее на приложение, значительно возрастает. Но главное преимущество покадровой анимации перед скелетной это возможность воспроизводить сложные движения.

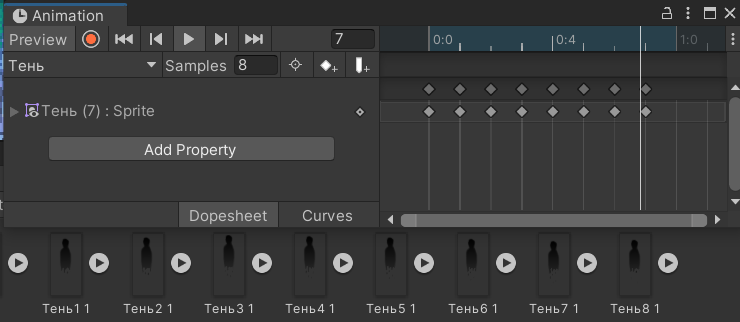


Рисунок 1. Окно Animation с внесенными спрайтами.

# Скелетная анимация

Скелетная анимация — cпособ анимирования объектов, заключающийся в том, что создаётся скелет, представляющий собой как правило древообразную структуру костей, в которой каждая последующая кость «привязана» к предыдущей, то есть повторяет за ней движения и повороты с учётом иерархии в скелете. Далее каждая вершина модели «привязывается» к какой-либо кости скелета. Таким образом, при движении отдельной кости двигаются и все вершины, привязанные к ней. Благодаря этому задача аниматора сильно упрощается, потому что отпадает необходимость анимировать отдельно каждую вершину модели, а достаточно лишь задавать положение и поворот костей скелета.

Также благодаря такому методу сокращается и объём информации, необходимой для анимации. Достаточно хранить информацию о движении костей, а движения вершин высчитываются уже исходя из них.

Преимущества:

* Плавные движения
* Занимает немного времени на создание
* Малое потребление памяти

Недостатки:

* Больше нагрузка на процессор
* Не подходит для сложных движений

Сначала нужно разбить объект на компоненты, что будут крепиться к костям. В Unity к объекту добавляются кости в Skinning Editor. Прикрепляются составляющие объекта к созданным соответствующим им костям. Далее так же, как и для покадровой анимации создается Animator Controller, и в окне Animation появляются компоненты всего скелета объекта. Далее можно настроить, как и за какое время каждая составляющая объекта будет двигаться.

По сравнению с покадровой скелетная анимация занимает меньше времени на создание, что упрощает и ускоряет работу над приложением, но не подходит для сложных движений, например для анимации атаки с использованием магии или других постоянно изменяющихся объектов лучше использовать покадровую анимацию, так как это будет выглядеть более «живым». Но скелетную анимацию удобнее использовать для анимации простых движений, например бега и прыжков.

В данном проекте была использована покадровая анимация, так как самих движений немного, из-за чего это не повлияло на время разработки.

В Unity каждая анимация – это отдельный объект. Ее можно использовать для разных объектов, не занимая много памяти для создания отдельных анимаций каждому из них.

# Шейдеры

Шейдер (рис. 2) — это небольшая программа, которая содержит инструкции для GPU — графического процессора. Они описывают, как рассчитать экранный цвет конкретного материала.

Раньше для этого требовался специальный язык затенения, такой как Cg или HLSL, с соглашениями, немного отличающимися от обычных скриптов игрового процесса.

Unity предоставляет Shader Graph, чтобы упростить написание шейдеров при минимальном кодировании или его отсутствии. Лучше всего то, что Shader Graph позволяет легко начать работу с помощью визуального интерактивного интерфейса.

Такие анимации подойдут для фоновых и декоративных объектов, которые не являются основным фокусом внимания, но добавляют картинке живости и динамики. Используя шейдеры можно анимировать различные простые объекты, особенно если их должно быть много (косяк рыб, стая птиц, толпа крыс).

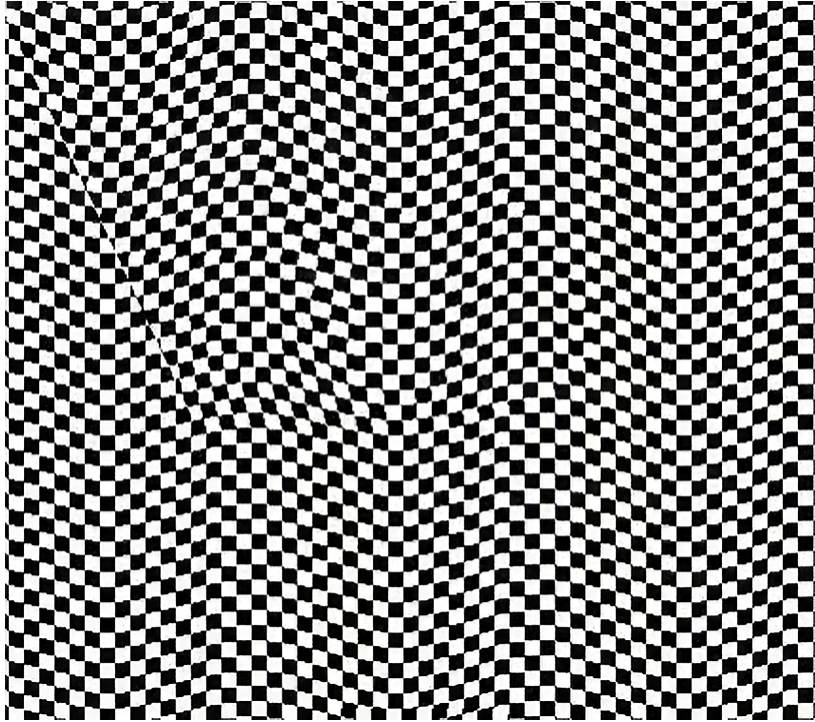


Рисунок 2. Шейдер на текстуре с шашечками.

**Плюсы:**

* Позволяют создавать простые эффекты
* Позволяет создать иллюзию движения объекта
* Упрощает анимацию большого количества одинаковых объектов

**Минусы:**

* Сложно в понимании
* Может потребоваться много времени на разработку.

Чтобы объекты в мире были более подвижны к ним был добавлен шейдер, что имитирует движение листьев на ветру. Шейдер применяется только на некоторых листьях (рис. 4), а не на всем дереве (рис. 3).



Рисунок 3. Статичный спрайт.

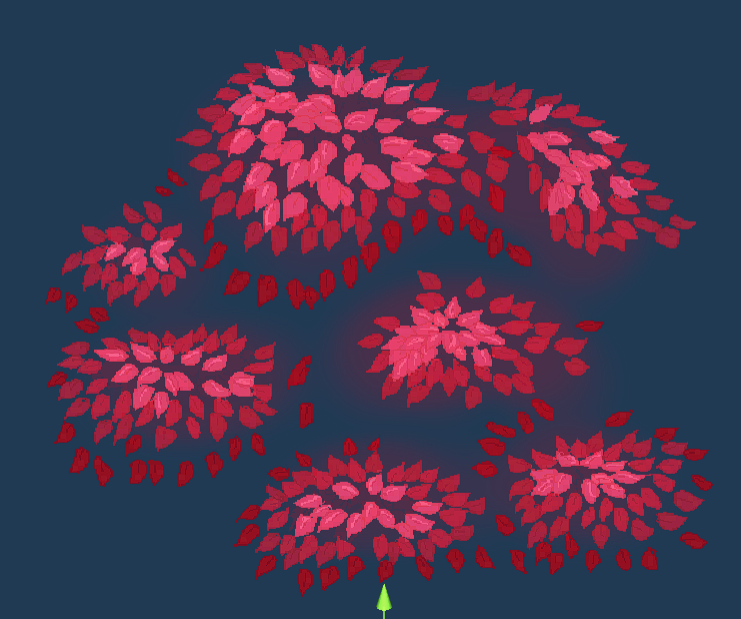


Рисунок 4. Движущийся спрайт с шейдером.

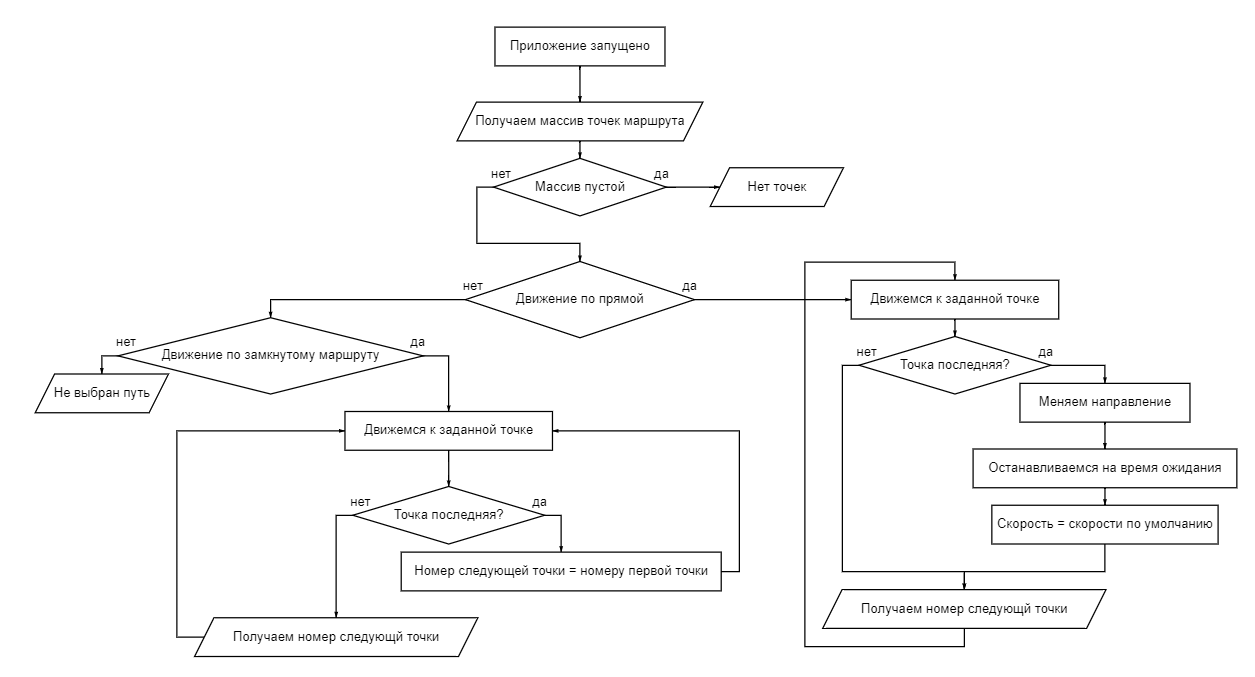
# Разработка Игрового приложения

Основным требованием к проекту в данном семестре было добавить в игровое приложение движение и анимацию, что включало в себя создание движущихся объектов виртуальной окружающей среды, анимация персонажа, а также анимация статичных объектов окружающей среды.

Чтобы выполнить данные требования были использованы такие ресурсы, как инструменты и методы для работы с анимацией и объектами представленные в Unity, так, например, для создания шейдера был использован Shader Graph. Были использованы стандартные функции для работы с объектами, например, изменение положения объекта в пространстве. Также для работы с анимацией были использованы инструменты Animator Controller и Animation.

# механика Движения объекта по заданному маршруту

Во время разработки была реализована механика движения объекта (приложение 1) по заданному маршруту (приложение 2).

Программа работает по следующему принципу:

Для осуществления движения сначала создается маршрут, что представляет собой массив точек задаваемых самостоятельно на сцене (рис. 5). Для удобства точки соединены линиями в том порядке, в каком они находятся в массиве (рис. 6). Если маршрут является циклом, то первая и последняя точка также соединяются. Рисование линий:

for (var i =1; i<PathElements.Length; i++)

{

Gizmos.DrawLine(PathElements[i-1].position, PathElements[i].position);

}

if (pathType == PathTypes.loop)

{

Gizmos.DrawLine(PathElements[0].position, PathElements[PathElements.Length-1].position);

}

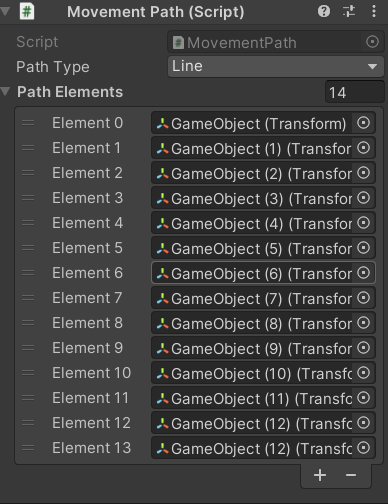
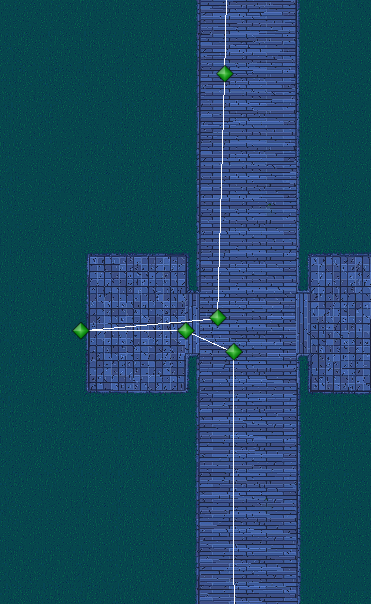


Рисунок 5. Массив точек маршрута.



Рисунки 6. Часть маршрута.

Движение объекта по маршруту осуществляется посредством стремления объекта к точкам маршрута, чьи местоположения берутся из массива. Здесь используются стандартные функции Unity для работы с объектами transform.position = Vector3.MoveTowards (transform.position, MyPath.PathElements[moveTo].position + posdist, Time.deltaTime\*currentspeed);).

Также если маршрут не замкнут, можно установить время остановки объекта в конце маршрута, что придает разнообразие движению:

if (moveTo >= MyPath.PathElements.Length - 1 && Waittime>0)

{

stop = true;

}

else

{

stop = false;

}

if (stop == true)

{

currentspeed= 0;

time += 5f\*Time.deltaTime;

}

if (time > Waittime)

{

currentspeed= speed/2;

time = 0;

stop = false;

moveTo = GetNextPoint(moveTo);

}

Есть опция отклонения от маршрута на определенное расстояние и изменение скорости, чтобы все объекты не двигались ровно по одной и той же линии (рис. 7-8).



Рисунки 7. Объекты без отклонения в маршруте.



Рисунки 8. Объекты с отклонением в маршруте.

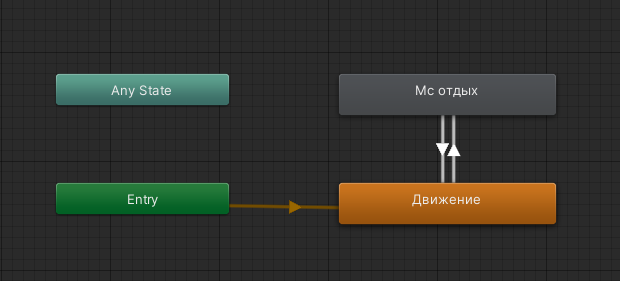
# Анимация объектов

Помимо простого перемещения спрайта по полю, для добавления реалистичности объект должен изменяться сам. Для этого к объектам была добавлена анимация.

Для движущихся «теней» и игрового персонажа была использована покадровая анимация.

Были отрисованы спрайты, что будут использоваться как кадры в анимации.

Для «теней» применяется только зацикленная анимация, состоящая из 8 кадров. А для игрового персонажа создавалось 5 анимаций: для каждого направления движения и когда персонаж не движется. Чтобы связать все движения с состоянием персонажа использовался Animator Controller (рис. 9).



Рисунки 9. Animator Controller персонажа.

В нем было создано Blend Tree, что упрощает выбор анимации в зависимости от того, в каком направлении движется персонаж (рис. 10).

Blend Tree было связано с анимацией покоя переходом с условием. Если скорость равна 0 то запускается анимация покоя, если больше 0 то анимация движения (рис. 11). У анимаций нет времени выхода, т.к. переход из покоя в движение и обратно происходит моментально, время выхода будет полезно при получении урона и использования атаки или взаимодействия с объектами.

Для передачи информации о скорости и направлении движения персонажа в программу движения были добавлены команды для работы с аниматором (приложение 3):

Vector2 lookDirection = new Vector2(1, 0);

Vector2 move;

void Update()

{

move = new Vector2(horizontal, vertical);

if (!Mathf.Approximately(move.x, 0.0f) || !Mathf.Approximately(move.y, 0.0f))

{

lookDirection.Set(move.x, move.y);

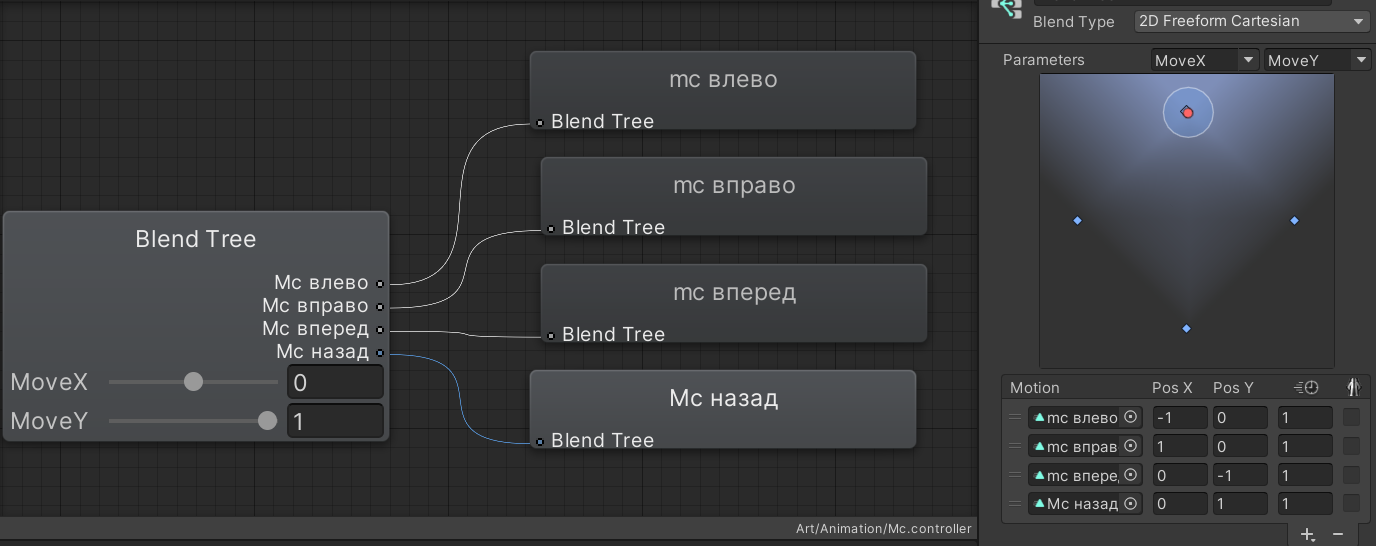
lookDirection.Normalize();

}

animator.SetFloat("MoveX", lookDirection.x);

animator.SetFloat("MoveY", lookDirection.y);

}



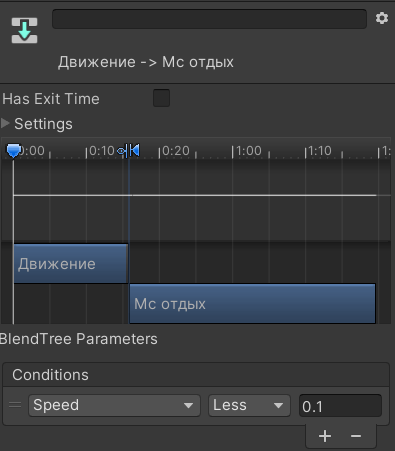
Рисунки 10. Blend Tree.

Рисунок 11. Условие перехода.

Для статичных объектов на сцене, например деревьев, был создан шейдер что имитирует ветер. Для деревьев он был использован на листьях, что были заданы отдельным спрайтом.

Шейдер был создан в Shader Graph (рис. 12) (код скомпилированного шейдера - приложение 4), что значительно упростило понимание работы шейдеров и процесс их создания. Шейдер представляет из себя две сложенных синусоидальных волны, одна из которых находится под углом по отношению к другой. Также само изменение объекта происходит только в x и y координатах, так как игровое пространство двумерное.

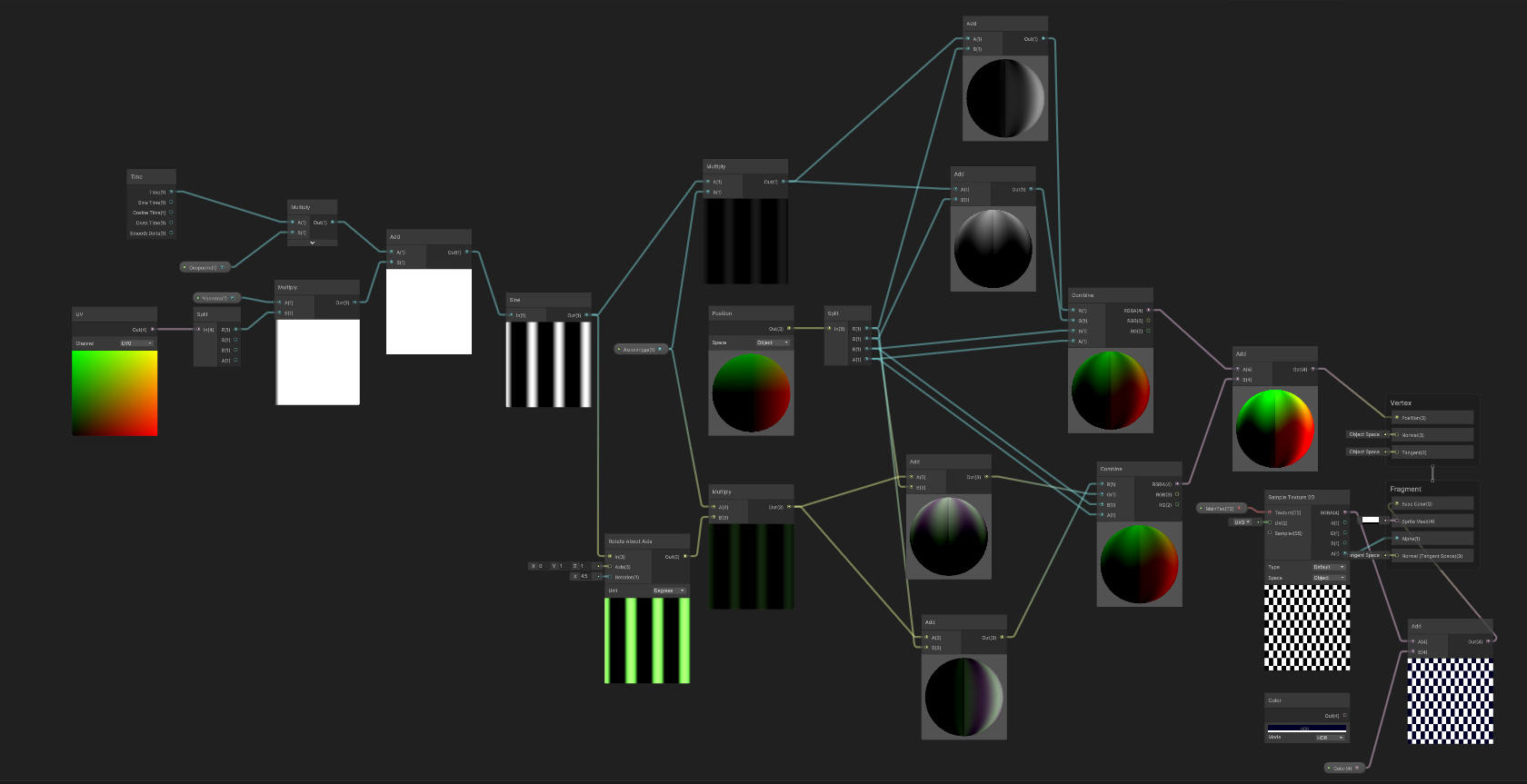


Рисунок 12. Созданный шейдер.

Для настройки шейдера были добавлены такие параметры как амплитуда, частота и скорость. Их можно настроить из самого объекта (рис. 13), на котором лежит шейдер, благодаря чему можно реализовать различные движения для разных объектов, не создавая новых шейдеров.

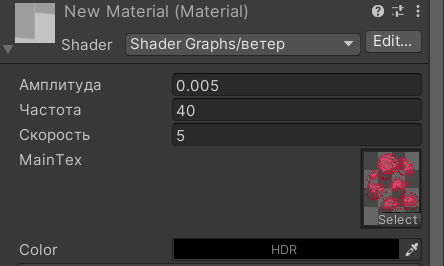


Рисунок 13. Параметры шейдера видимые на объекте.

# Заключение

За данный семестр были изучены методы работы с анимацией и шейдерами в Unity. Были рассмотрены покадровая и скелетная анимации в Unity, а также изучены шейдеры. Было выявлено, что покадровая анимация больше подходит проекту, из-за возможности создавать более сложные и «живые» анимации.

Для разработки приложения был рассмотрен инструмент Shader Graph, который облегчил работу с шейдерами, благодаря котором появилась возможность создавать иллюзию движения для простых объектов. Были использованы стандартные функции Unity для работы с объектами в особенности изменение положения объекта в пространстве.

На следующий семестр планируются такие задачи:

1. Добавление механик взаимодействия персонажа с миром.
2. Добавление новых элементов окружающей среды.
3. Добавление простого пользовательского интерфейса.

Подведя итог, можно сказать, что в следующем семестре планируется больше разработать механику взаимодействия с миром. Таким образом будет продолжена работа над данным проектом в будущем.

# Список источников:

* 1. «Галоп пикселя — часть пятая» — Анимация персонажей. Ходьба // habr.com - Weilard [Электронный ресурс] -Режим доступа: <https://habr.com/ru/articles/441562/> (Дата обращения 14.06.2023)
  2. Под капотом Graveyard Keeper: Как реализованы графические эффекты // vk.com [Электронный ресурс] -Режим доступа: [https://vk.com/ @scienceandlive-pod-kapotom-graveyard-keeper-kak-realizovany-graficheskie-ef](https://vk.com/@scienceandlive-pod-kapotom-graveyard-keeper-kak-realizovany-graficheskie-ef) (Дата обращения 21.05.2023)
  3. Патрулирование 2D при помощи UNITY и C# - легко // youtube.com - [Электронный ресурс] -Режим доступа: <https://www.youtube.com/watch?v=EkmNtwgAkMk> (Дата обращения 25.04.2023)
  4. Создаём простой зомби-шутер на Unity // tproger.ru - OTUS Онлайн-образование [Электронный ресурс] -Режим доступа: <https://tproger.ru/articles/sozdajom-prostoj-zombi-shuter-na-unity/> (Дата обращения 20.04.2023)
  5. Анимация с помощью шейдера в Unity // dtf.ru - Андрей Торчинский [Электронный ресурс] -Режим доступа: <https://dtf.ru/gamedev/152957-animaciya-s-pomoshchyu-sheydera-v-unity> (Дата обращения 04.06.2023)
  6. Преимущества использования уникальных шейдеров в компьютерных играх: личный опыт и наблюдения // devby.io [Электронный ресурс] -Режим доступа: <https://devby.io/news/preimuschestva-ispolzovaniya-unikalnyh-sheyderov-v-kompyuternyh-igrah-lichnyy-opyt-i-nablyudeniya> (Дата обращения 14.06.2023)
  7. Shader Graph в Unity для начинающих // jwinters.ru [Электронный ресурс] -Режим доступа: <https://jwinters.ru/unity3d/shader-graph/> (Дата обращения 08.06.2023)
  8. Форум Unity// unity.com [Электронный ресурс] -Режим доступа: <https://forum.unity.com/> (Дата обращения 10.06.2023)
  9. Движение по пути - гибкий вариант #2 - Unity легко // youtube.com [Электронный ресурс] -Режим доступа: https://www.youtube.com /watch?v=07ziQqKnZUQ (Дата обращения 26.04.2023)
  10. Cистема вопросов и ответов о программировании // stackoverflow.com [Электронный ресурс] -Режим доступа: <https://ru.stackoverflow.com/> (Дата обращения 10.06.2023)
  11. Простейшая скелетная анимация со звуками в Unity на примере игры Uspavanka (с нулевой базой) // dzen.ru [Электронный ресурс] -Режим доступа: <https://dzen.ru/media/thedogsdream/prosteishaia-skeletnaia-animaciia-so-zvukami-v-unity-na-primere-igry-uspavanka-s-nulevoi-bazoi-63492f20607b3c67e6184706?utm_referer=www.google.com> (Дата обращения 29.05.2023)
  12. Основы создания 2D персонажа в Unity 3D 4.3. Часть 1: заготовка персонажа и анимация покоя // habr.com - Charoplet [Электронный ресурс] -Режим доступа: <https://habr.com/ru/articles/211472/> (Дата обращения 13.01.2023)

# Приложение 1

Программа, отвечающая за создание точек маршрута для объектов:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MovementPath : MonoBehaviour

{

public enum PathTypes

{

line,

loop

}

public PathTypes pathType;

public Transform[] PathElements;

public void OnDrawGizmos()

{

if(PathElements == null || PathElements.Length<2)

{

return;

}

for (var i =1; i<PathElements.Length; i++)

{

Gizmos.DrawLine(PathElements[i-1].position, PathElements[i].position);

}

if (pathType == PathTypes.loop)

{

Gizmos.DrawLine(PathElements[0].position, PathElements[PathElements.Length-1].position);

}

}

}

# Приложение 2

Программа, отвечающая за движение объекта по маршруту:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using static MovementPath;

using UnityEngine.EventSystems;

public class FollowPath : MonoBehaviour

{

public enum MoveType

{

Move,

Loop

}

public MoveType Type = MoveType.Move;

public MovementPath MyPath;

public float Waittime;

public float time = 0;

public float speed;

public float currentspeed;

public float Maxdistance = .1f;

public float distance = 0;

public bool stop;

Vector3 posdist;

public float xdist, ydist=0;

public int moveDirection = 1;

public int moveTo = 0;

public IEnumerator<Transform> pointInPath;

public int GetNextPoint( int moveTo)

{

if (MyPath.PathElements == null || MyPath.PathElements.Length < 1)

{

return 0;

}

while (true)

{

if (MyPath.PathElements.Length == 1)

{

continue;

}

if (Type == MoveType.Move)

{

if (moveTo <= 0)

{

moveDirection = 1;

}

else if (moveTo >= MyPath.PathElements.Length - 1)

{

moveDirection = -1;

}

}

moveTo += moveDirection;

if (Type == MoveType.Loop)

{

if (moveTo >= MyPath.PathElements.Length)

{

moveTo = 0;

}

if (moveTo < 0)

{

moveTo = MyPath.PathElements.Length - 1;

}

}

return moveTo;

}

}

// Start is called before the first frame update

void Start()

{

posdist = new Vector3(0.5f\*distance, 0.1f\*distance, 0);

Vector3 dist = new Vector3(xdist, 2\*ydist, 0);

if (MyPath == null)

{

Debug.Log("Примени путь");

return;

}

currentspeed = speed/2;

if (MyPath.PathElements[0] == null)

{

Debug.Log("Нужны точки");

return ;

}

transform.position = MyPath.PathElements[moveTo].position +posdist+2\*dist;

stop = false;

moveTo = GetNextPoint(moveTo);

}

// Update is called once per frame

void Update()

{

if (MyPath.PathElements == null || MyPath.PathElements[0] == null)

{

return ;

}

if (Type == MoveType.Move && stop == false)

{

transform.position =Vector3.MoveTowards(transform.position, MyPath.PathElements[moveTo].position +posdist, Time.deltaTime\*currentspeed);

}

else if (Type == MoveType.Loop)

{

transform.position = Vector3.Lerp(transform.position, MyPath.PathElements[moveTo].position + posdist, Time.deltaTime \* currentspeed);

}

var distanceSquare = (transform.position - (MyPath.PathElements[moveTo].position + posdist)).sqrMagnitude;

if (distanceSquare < Maxdistance\*Maxdistance && stop ==false)

{

moveTo=GetNextPoint(moveTo);

}

if (moveTo >= MyPath.PathElements.Length - 1 && Waittime>0)

{

stop = true;

}

else

{

stop = false;

}

if (stop == true)

{

currentspeed= 0;

time += 5f\*Time.deltaTime;

}

if (time > Waittime)

{

currentspeed= speed/2;

time = 0;

stop = false;

moveTo = GetNextPoint(moveTo);

}

}

}

# Приложение 3

Измененная программа, отвечающая за движение персонажа:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MCcontroller : MonoBehaviour

{

public int maxSpeed = 100;

public int maxMana = 20;

public int maxHealth = 5;

Animator animator;

Vector2 lookDirection = new Vector2(1, 0);

Vector2 move;

int currentHealth;

public int health { get { return currentHealth; } }

float currentSpeed;

int currentMana;

Rigidbody2D rigidbody2d;

float horizontal;

float vertical;

float speed;

bool Running;

// Start is called before the first frame update

void Start()

{

animator = GetComponent<Animator>();

rigidbody2d = GetComponent<Rigidbody2D>();

currentHealth = maxHealth;

currentSpeed = maxSpeed;

speed = 1.5f;

Running = false;

}

// Update is called once per frame

void Update()

{

horizontal = Input.GetAxis("Horizontal");

vertical = Input.GetAxis("Vertical");

move = new Vector2(horizontal, vertical);

if ((Input.GetKeyDown(KeyCode.LeftShift) || Input.GetKeyDown(KeyCode.RightShift)) && (horizontal != 0 || vertical != 0))

{

speed = 3.0f;

Running = true;

}

if (Input.GetKeyUp(KeyCode.LeftShift) || Input.GetKeyUp(KeyCode.RightShift))

{

speed = 1.5f;

Running = false;

}

if ((Running == false)) //Если ускорения нет то выносливость восстанавливается на один за единицу времени, если есть то тратиться на 2 за единицу времени

{

ChangeSpeed(1 \* Time.deltaTime);

speed = 1.5f;

}

else

{

ChangeSpeed(-2 \* Time.deltaTime);

speed = 3.0f;

}

if (!Mathf.Approximately(move.x, 0.0f) || !Mathf.Approximately(move.y, 0.0f))

{

lookDirection.Set(move.x, move.y);

lookDirection.Normalize();

}

if (!Mathf.Approximately(move.x, 0.0f) || !Mathf.Approximately(move.y, 0.0f))

{

lookDirection.Set(move.x, move.y);

lookDirection.Normalize();

}

animator.SetFloat("MoveX", lookDirection.x);

animator.SetFloat("MoveY", lookDirection.y);

}

void FixedUpdate()

{

Vector2 position = rigidbody2d.position;

animator.SetFloat("MoveX", lookDirection.x);

animator.SetFloat("MoveY", lookDirection.y);

if ((position.x == (position.x + horizontal \* Time.deltaTime \* speed)) && (position.y == position.y + vertical \* Time.deltaTime \* speed))

{

Running = false;

speed = 0f;

}

else if (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift))

{

Running = true;

}

position.x = position.x + horizontal \* Time.deltaTime \* speed;

position.y = position.y + vertical \* Time.deltaTime \* speed;

animator.SetFloat("Speed", speed);

rigidbody2d.MovePosition(position);

}

public void ChangeHealth(int amount)

{

currentHealth = Mathf.Clamp(currentHealth + amount, 0, maxHealth);

Debug.Log(currentHealth + "/" + maxHealth);

}

void ChangeSpeed(float amount)

{

currentSpeed = Mathf.Clamp(currentSpeed + amount, 0, maxSpeed);

if (currentSpeed <= 0) //Если наша выносливость закончилась, ускорения не будет

{

speed = 1.5f;

Running = false;

}

}

void ChangeMana(int amount)

{

currentMana = Mathf.Clamp(currentMana + amount, 0, maxMana);

Debug.Log(currentMana + "/" + maxMana);

}

}

# Приложение 4

Программа, написанного шейдера для движения листьев:

Shader "Shader Graphs/ветер" {

Properties {

\_ ("Амплитуда", Float) = 0.100000

\_1 ("Частота", Float) = 2.000000

\_2 ("Скорость", Float) = 1.000000

[NoScaleOffset] \_MainTex ("MainTex", 2D) = "white" { }

[HDR] \_Color ("Color", Color) = (0.000000,0.000000,0.019608,1.000000)

[HideInInspector] [NoScaleOffset] unity\_Lightmaps ("unity\_Lightmaps", 2DArray) = "" { }

[HideInInspector] [NoScaleOffset] unity\_LightmapsInd ("unity\_LightmapsInd", 2DArray) = "" { }

[HideInInspector] [NoScaleOffset] unity\_ShadowMasks ("unity\_ShadowMasks", 2DArray) = "" { }

}

SubShader {

Tags { "QUEUE"="Transparent" "RenderType"="Transparent" "RenderPipeline"="UniversalPipeline" "UniversalMaterialType"="Lit" "ShaderGraphShader"="true" "ShaderGraphTargetId"="" }

Pass {

Name "Sprite Lit"

Tags { "LIGHTMODE"="Universal2D" "QUEUE"="Transparent" "RenderType"="Transparent" "RenderPipeline"="UniversalPipeline" "UniversalMaterialType"="Lit" "ShaderGraphShader"="true" "ShaderGraphTargetId"="" }

ZWrite Off

Cull Off

Blend SrcAlpha OneMinusSrcAlpha, One OneMinusSrcAlpha

Keywords: <none>

-- Hardware tier variant: Tier 1

-- Vertex shader for "d3d11":

Uses vertex data channel "Vertex"

Uses vertex data channel "TexCoord0"

Uses vertex data channel "TexCoord3"

Uses vertex data channel "Color"

Constant Buffer "$Globals" (2016 bytes) on slot 0 {

Matrix4x4 unity\_MatrixVP at 1168

Vector4 \_TimeParameters at 240

Vector4 \_ProjectionParams at 272

Vector4 \_RendererColor at 2000

}

Constant Buffer "UnityPerDraw" (656 bytes) on slot 1 {

Matrix4x4 unity\_ObjectToWorld at 0

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 2 {

Float \_ at 0

Float \_1 at 4

Float \_2 at 8

}

Shader Disassembly:

vs\_4\_0

dcl\_constantbuffer CB0[126], immediateIndexed

dcl\_constantbuffer CB1[4], immediateIndexed

dcl\_constantbuffer CB2[1], immediateIndexed

dcl\_input v0.xyz

dcl\_input v3.xyzw

dcl\_input v4.xyzw

dcl\_input v5.xyzw

dcl\_output\_siv o0.xyzw, position

dcl\_output o1.xyz

dcl\_output o2.xyzw

dcl\_output o3.xyzw

dcl\_output o4.xyzw

dcl\_output o5.xyzw

dcl\_temps 2

0: mul r0.x, v3.x, cb2[0].y

1: mad r0.x, cb0[15].x, cb2[0].z, r0.x

2: sincos r0.x, null, r0.x

3: dp3 r0.y, l(0.853553, 0.146447, 0.500000, 0.000000), r0.xxxx

4: mad r1.xy, r0.xxxx, cb2[0].xxxx, v0.xyxx

5: mad r0.xy, cb2[0].xxxx, r0.yyyy, v0.xyxx

6: mov r1.z, v0.z

7: mov r0.z, v0.z

8: add r0.xyz, r0.xyzx, r1.xyzx

9: mul r1.xyz, r0.yyyy, cb1[1].xyzx

10: mad r0.xyw, cb1[0].xyxz, r0.xxxx, r1.xyxz

11: mad r0.xyz, cb1[2].xyzx, r0.zzzz, r0.xywx

12: add r0.xyz, r0.xyzx, cb1[3].xyzx

13: mul r1.xyzw, r0.yyyy, cb0[74].xyzw

14: mad r1.xyzw, cb0[73].xyzw, r0.xxxx, r1.xyzw

15: mad r1.xyzw, cb0[75].xyzw, r0.zzzz, r1.xyzw

16: mov o1.xyz, r0.xyzx

17: add r0.xyzw, r1.xyzw, cb0[76].xyzw

18: mov o0.xyzw, r0.xyzw

19: mov o2.xyzw, v3.xyzw

20: mov o3.xyzw, v4.xyzw

21: mul o4.xyzw, v5.xyzw, cb0[125].xyzw

22: mul r0.y, r0.y, cb0[17].x

23: mul r1.xzw, r0.xxwy, l(0.500000, 0.000000, 0.500000, 0.500000)

24: mov o5.zw, r0.zzzw

25: add o5.xy, r1.zzzz, r1.xwxx

26: ret

-- Hardware tier variant: Tier 1

-- Fragment shader for "d3d11":

Set 2D Texture "\_MainTex" to slot 0

Constant Buffer "$Globals" (2016 bytes) on slot 0 {

Vector2 \_GlobalMipBias at 304

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 1 {

Vector4 \_Color at 32

}

Shader Disassembly:

ps\_4\_0

dcl\_constantbuffer CB0[20], immediateIndexed

dcl\_constantbuffer CB1[3], immediateIndexed

dcl\_sampler s0, mode\_default

dcl\_resource\_texture2d (float,float,float,float) t0

dcl\_input\_ps linear v3.xy

dcl\_input\_ps linear v4.xyzw

dcl\_output o0.xyzw

dcl\_temps 4

0: sample\_b r0.xyzw, v3.xyxx, t0.xyzw, s0, cb0[19].x

1: eq r1.x, r0.w, l(0.000000)

2: discard\_nz r1.x

3: mul r1.xyz, cb1[2].xyzx, l(12.920000, 12.920000, 12.920000, 0.000000)

4: log r2.xyz, |cb1[2].xyzx|

5: mul r2.xyz, r2.xyzx, l(0.416667, 0.416667, 0.416667, 0.000000)

6: exp r2.xyz, r2.xyzx

7: mad r2.xyz, r2.xyzx, l(1.055000, 1.055000, 1.055000, 0.000000), l(-0.055000, -0.055000, -0.055000, 0.000000)

8: ge r3.xyz, l(0.003131, 0.003131, 0.003131, 0.000000), cb1[2].xyzx

9: movc r1.xyz, r3.xyzx, r1.xyzx, r2.xyzx

10: add r0.xyz, r0.xyzx, r1.xyzx

11: mul o0.xyzw, r0.xyzw, v4.xyzw

12: ret

Keywords: DEBUG\_DISPLAY

-- Vertex shader for "d3d11":

-- Hardware tier variant: Tier 1

-- Fragment shader for "d3d11":

Set 2D Texture "\_MainTex" to slot 0

Constant Buffer "$Globals" (2160 bytes) on slot 0 {

Vector2 \_GlobalMipBias at 304

ScalarInt \_DebugMaterialMode at 1984

ScalarInt \_DebugSceneOverrideMode at 2000

Vector4 \_DebugColor at 2064

Vector4 \_DebugColorInvalidMode at 2080

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 1 {

Vector4 \_Color at 32

}

Shader Disassembly:

ps\_4\_0

dcl\_constantbuffer CB0[131], immediateIndexed

dcl\_constantbuffer CB1[3], immediateIndexed

dcl\_sampler s0, mode\_default

dcl\_resource\_texture2d (float,float,float,float) t0

dcl\_input\_ps linear v3.xy

dcl\_input\_ps linear v4.xyzw

dcl\_output o0.xyzw

dcl\_temps 5

0: sample\_b r0.xyzw, v3.xyxx, t0.wxyz, s0, cb0[19].x

1: eq r1.x, r0.x, l(0.000000)

2: discard\_nz r1.x

3: mul r1.xyz, cb1[2].xyzx, l(12.920000, 12.920000, 12.920000, 0.000000)

4: log r2.xyz, |cb1[2].xyzx|

5: mul r2.xyz, r2.xyzx, l(0.416667, 0.416667, 0.416667, 0.000000)

6: exp r2.xyz, r2.xyzx

7: mad r2.xyz, r2.xyzx, l(1.055000, 1.055000, 1.055000, 0.000000), l(-0.055000, -0.055000, -0.055000, 0.000000)

8: ge r3.xyz, l(0.003131, 0.003131, 0.003131, 0.000000), cb1[2].xyzx

9: movc r1.xyz, r3.xyzx, r1.xyzx, r2.xyzx

10: add r1.xyz, r0.yzwy, r1.xyzx

11: switch cb0[124].x

12: case l(0)

13: mov r2.xyzw, l(0,0,0,0)

14: mov r3.x, l(0)

15: break

16: case l(1)

17: mov r1.w, l(1.000000)

18: mov r2.xyzw, r1.xyzw

19: mov r3.x, l(-1)

20: break

21: case l(3)

22: mov r0.w, l(1.000000)

23: mov r2.xyzw, r0.xxxw

24: mov r3.x, l(-1)

25: break

26: case l(11)

27: mov r2.xyzw, l(1.000000,1.000000,1.000000,1.000000)

28: mov r3.x, l(-1)

29: break

30: case l(8)

31: case l(7)

32: mov r2.xyzw, l(0,0,1.000000,1.000000)

33: mov r3.x, l(-1)

34: break

35: default

36: mov r2.xyzw, cb0[130].xyzw

37: mov r3.x, l(-1)

38: break

39: endswitch

40: ine r1.w, cb0[124].x, l(11)

41: movc r1.w, r3.x, r1.w, l(-1)

42: if\_nz r1.w

43: movc r4.xyzw, cb0[125].xxxx, cb0[129].xyzw, cb0[130].xyzw

44: movc o0.xyzw, r3.xxxx, r2.xyzw, r4.xyzw

45: ret

46: endif

47: mov r0.yzw, r1.xxyz

48: mul o0.xyzw, r0.yzwx, v4.xyzw

49: ret

}

// Stats for Vertex shader:

// d3d11: 22 math

// Stats for Fragment shader:

// d3d11: 1 math

Pass {

Name "Sprite Normal"

Tags { "LIGHTMODE"="NormalsRendering" "QUEUE"="Transparent" "RenderType"="Transparent" "RenderPipeline"="UniversalPipeline" "UniversalMaterialType"="Lit" "ShaderGraphShader"="true" "ShaderGraphTargetId"="" }

ZWrite Off

Cull Off

Blend SrcAlpha OneMinusSrcAlpha, One OneMinusSrcAlpha

Keywords: <none>

-- Hardware tier variant: Tier 1

-- Vertex shader for "d3d11":

// Stats: 22 math, 2 temp registers

Uses vertex data channel "Vertex"

Uses vertex data channel "Tangent"

Uses vertex data channel "TexCoord0"

Uses vertex data channel "TexCoord3"

Constant Buffer "$Globals" (2000 bytes) on slot 0 {

Matrix4x4 unity\_MatrixV at 976

Matrix4x4 unity\_MatrixVP at 1168

Vector4 \_TimeParameters at 240

}

Constant Buffer "UnityPerDraw" (656 bytes) on slot 1 {

Matrix4x4 unity\_ObjectToWorld at 0

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 2 {

Float \_ at 0

Float \_1 at 4

Float \_2 at 8

}

Shader Disassembly:

vs\_4\_0

dcl\_constantbuffer CB0[77], immediateIndexed

dcl\_constantbuffer CB1[4], immediateIndexed

dcl\_constantbuffer CB2[1], immediateIndexed

dcl\_input v0.xyz

dcl\_input v2.xyzw

dcl\_input v3.x

dcl\_input v4.xyzw

dcl\_output\_siv o0.xyzw, position

dcl\_output o1.xyz

dcl\_output o2.xyzw

dcl\_output o3.xyzw

dcl\_temps 2

0: mul r0.x, v3.x, cb2[0].y

1: mad r0.x, cb0[15].x, cb2[0].z, r0.x

2: sincos r0.x, null, r0.x

3: dp3 r0.y, l(0.853553, 0.146447, 0.500000, 0.000000), r0.xxxx

4: mad r1.xy, r0.xxxx, cb2[0].xxxx, v0.xyxx

5: mad r0.xy, cb2[0].xxxx, r0.yyyy, v0.xyxx

6: mov r1.z, v0.z

7: mov r0.z, v0.z

8: add r0.xyz, r0.xyzx, r1.xyzx

9: mul r1.xyz, r0.yyyy, cb1[1].xyzx

10: mad r0.xyw, cb1[0].xyxz, r0.xxxx, r1.xyxz

11: mad r0.xyz, cb1[2].xyzx, r0.zzzz, r0.xywx

12: add r0.xyz, r0.xyzx, cb1[3].xyzx

13: mul r1.xyzw, r0.yyyy, cb0[74].xyzw

14: mad r1.xyzw, cb0[73].xyzw, r0.xxxx, r1.xyzw

15: mad r0.xyzw, cb0[75].xyzw, r0.zzzz, r1.xyzw

16: add o0.xyzw, r0.xyzw, cb0[76].xyzw

17: mov o1.x, cb0[61].z

18: mov o1.y, cb0[62].z

19: mov o1.z, cb0[63].z

20: mul r0.xyz, v2.yyyy, cb1[1].xyzx

21: mad r0.xyz, cb1[0].xyzx, v2.xxxx, r0.xyzx

22: mad r0.xyz, cb1[2].xyzx, v2.zzzz, r0.xyzx

23: dp3 r0.w, r0.xyzx, r0.xyzx

24: max r0.w, r0.w, l(0.000000)

25: rsq r0.w, r0.w

26: mul o2.xyz, r0.wwww, r0.xyzx

27: mov o2.w, v2.w

28: mov o3.xyzw, v4.xyzw

29: ret

-- Hardware tier variant: Tier 1

-- Fragment shader for "d3d11":

Set 2D Texture "\_MainTex" to slot 0

Constant Buffer "$Globals" (2000 bytes) on slot 0 {

Vector2 \_GlobalMipBias at 304

}

Shader Disassembly:

ps\_4\_0

dcl\_constantbuffer CB0[20], immediateIndexed

dcl\_sampler s0, mode\_default

dcl\_resource\_texture2d (float,float,float,float) t0

dcl\_input\_ps linear v1.xyz

dcl\_input\_ps linear v3.xy

dcl\_output o0.xyzw

dcl\_temps 1

0: sample\_b r0.xyzw, v3.xyxx, t0.xyzw, s0, cb0[19].x

1: mov o0.w, r0.w

2: mad o0.xyz, v1.xyzx, l(0.500000, 0.500000, 0.500000, 0.000000), l(0.500000, 0.500000, 0.500000, 0.000000)

3: ret

}

Pass {

Name "Sprite Forward"

Tags { "LIGHTMODE"="UniversalForward" "QUEUE"="Transparent" "RenderType"="Transparent" "RenderPipeline"="UniversalPipeline" "UniversalMaterialType"="Lit" "ShaderGraphShader"="true" "ShaderGraphTargetId"="" }

ZWrite Off

Cull Off

Blend SrcAlpha OneMinusSrcAlpha, One OneMinusSrcAlpha

Keywords: <none>

-- Hardware tier variant: Tier 1

-- Vertex shader for "d3d11":

Uses vertex data channel "Vertex"

Uses vertex data channel "TexCoord0"

Uses vertex data channel "TexCoord3"

Uses vertex data channel "Color"

Constant Buffer "$Globals" (2016 bytes) on slot 0 {

Matrix4x4 unity\_MatrixVP at 1168

Vector4 \_TimeParameters at 240

Vector4 \_RendererColor at 2000

}

Constant Buffer "UnityPerDraw" (656 bytes) on slot 1 {

Matrix4x4 unity\_ObjectToWorld at 0

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 2 {

Float \_ at 0

Float \_1 at 4

Float \_2 at 8

}

Shader Disassembly:

vs\_4\_0

dcl\_constantbuffer CB0[126], immediateIndexed

dcl\_constantbuffer CB1[4], immediateIndexed

dcl\_constantbuffer CB2[1], immediateIndexed

dcl\_input v0.xyz

dcl\_input v3.xyzw

dcl\_input v4.xyzw

dcl\_input v5.xyzw

dcl\_output\_siv o0.xyzw, position

dcl\_output o1.xyz

dcl\_output o2.xyzw

dcl\_output o3.xyzw

dcl\_output o4.xyzw

dcl\_temps 2

0: mul r0.x, v3.x, cb2[0].y

1: mad r0.x, cb0[15].x, cb2[0].z, r0.x

2: sincos r0.x, null, r0.x

3: dp3 r0.y, l(0.853553, 0.146447, 0.500000, 0.000000), r0.xxxx

4: mad r1.xy, r0.xxxx, cb2[0].xxxx, v0.xyxx

5: mad r0.xy, cb2[0].xxxx, r0.yyyy, v0.xyxx

6: mov r1.z, v0.z

7: mov r0.z, v0.z

8: add r0.xyz, r0.xyzx, r1.xyzx

9: mul r1.xyz, r0.yyyy, cb1[1].xyzx

10: mad r0.xyw, cb1[0].xyxz, r0.xxxx, r1.xyxz

11: mad r0.xyz, cb1[2].xyzx, r0.zzzz, r0.xywx

12: add r0.xyz, r0.xyzx, cb1[3].xyzx

13: mul r1.xyzw, r0.yyyy, cb0[74].xyzw

14: mad r1.xyzw, cb0[73].xyzw, r0.xxxx, r1.xyzw

15: mad r1.xyzw, cb0[75].xyzw, r0.zzzz, r1.xyzw

16: mov o1.xyz, r0.xyzx

17: add o0.xyzw, r1.xyzw, cb0[76].xyzw

18: mov o2.xyzw, v3.xyzw

19: mov o3.xyzw, v4.xyzw

20: mul o4.xyzw, v5.xyzw, cb0[125].xyzw

21: ret

-- Hardware tier variant: Tier 1

-- Fragment shader for "d3d11":

Set 2D Texture "\_MainTex" to slot 0

Constant Buffer "$Globals" (2016 bytes) on slot 0 {

Vector2 \_GlobalMipBias at 304

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 1 {

Vector4 \_Color at 32

}

Shader Disassembly:

ps\_4\_0

dcl\_constantbuffer CB0[20], immediateIndexed

dcl\_constantbuffer CB1[3], immediateIndexed

dcl\_sampler s0, mode\_default

dcl\_resource\_texture2d (float,float,float,float) t0

dcl\_input\_ps linear v3.xy

dcl\_input\_ps linear v4.xyzw

dcl\_output o0.xyzw

dcl\_temps 3

0: log r0.xyz, |cb1[2].xyzx|

1: mul r0.xyz, r0.xyzx, l(0.416667, 0.416667, 0.416667, 0.000000)

2: exp r0.xyz, r0.xyzx

3: mad r0.xyz, r0.xyzx, l(1.055000, 1.055000, 1.055000, 0.000000), l(-0.055000, -0.055000, -0.055000, 0.000000)

4: mul r1.xyz, cb1[2].xyzx, l(12.920000, 12.920000, 12.920000, 0.000000)

5: ge r2.xyz, l(0.003131, 0.003131, 0.003131, 0.000000), cb1[2].xyzx

6: movc r0.xyz, r2.xyzx, r1.xyzx, r0.xyzx

7: sample\_b r1.xyzw, v3.xyxx, t0.xyzw, s0, cb0[19].x

8: add r1.xyz, r0.xyzx, r1.xyzx

9: mul o0.xyzw, r1.xyzw, v4.xyzw

10: ret

Keywords: DEBUG\_DISPLAY

-- Vertex shader for "d3d11":

-- Hardware tier variant: Tier 1

-- Fragment shader for "d3d11":

Set 2D Texture "\_MainTex" to slot 0

Constant Buffer "$Globals" (2160 bytes) on slot 0 {

Vector2 \_GlobalMipBias at 304

ScalarInt \_DebugMaterialMode at 1984

ScalarInt \_DebugSceneOverrideMode at 2000

Vector4 \_DebugColor at 2064

Vector4 \_DebugColorInvalidMode at 2080

}

Constant Buffer "UnityPerMaterial" (48 bytes) on slot 1 {

Vector4 \_Color at 32

}

Shader Disassembly:

ps\_4\_0

dcl\_constantbuffer CB0[131], immediateIndexed

dcl\_constantbuffer CB1[3], immediateIndexed

dcl\_sampler s0, mode\_default

dcl\_resource\_texture2d (float,float,float,float) t0

dcl\_input\_ps linear v3.xy

dcl\_input\_ps linear v4.xyzw

dcl\_output o0.xyzw

dcl\_temps 5

0: sample\_b r0.xyzw, v3.xyxx, t0.wxyz, s0, cb0[19].x

1: mul r1.xyz, cb1[2].xyzx, l(12.920000, 12.920000, 12.920000, 0.000000)

2: log r2.xyz, |cb1[2].xyzx|

3: mul r2.xyz, r2.xyzx, l(0.416667, 0.416667, 0.416667, 0.000000)

4: exp r2.xyz, r2.xyzx

5: mad r2.xyz, r2.xyzx, l(1.055000, 1.055000, 1.055000, 0.000000), l(-0.055000, -0.055000, -0.055000, 0.000000)

6: ge r3.xyz, l(0.003131, 0.003131, 0.003131, 0.000000), cb1[2].xyzx

7: movc r1.xyz, r3.xyzx, r1.xyzx, r2.xyzx

8: add r1.xyz, r0.yzwy, r1.xyzx

9: switch cb0[124].x

10: case l(0)

11: mov r2.xyzw, l(0,0,0,0)

12: mov r3.x, l(0)

13: break

14: case l(1)

15: mov r1.w, l(1.000000)

16: mov r2.xyzw, r1.xyzw

17: mov r3.x, l(-1)

18: break

19: case l(3)

20: mov r0.w, l(1.000000)

21: mov r2.xyzw, r0.xxxw

22: mov r3.x, l(-1)

23: break

24: case l(11)

25: mov r2.xyzw, l(1.000000,1.000000,1.000000,1.000000)

26: mov r3.x, l(-1)

27: break

28: case l(8)

29: case l(7)

30: mov r2.xyzw, l(0,0,1.000000,1.000000)

31: mov r3.x, l(-1)

32: break

33: default

34: mov r2.xyzw, cb0[130].xyzw

35: mov r3.x, l(-1)

36: break

37: endswitch

38: ine r1.w, cb0[124].x, l(11)

39: movc r1.w, r3.x, r1.w, l(-1)

40: if\_nz r1.w

41: movc r4.xyzw, cb0[125].xxxx, cb0[129].xyzw, cb0[130].xyzw

42: movc o0.xyzw, r3.xxxx, r2.xyzw, r4.xyzw

43: ret

44: endif

45: mov r0.yzw, r1.xxyz

46: mul o0.xyzw, r0.yzwx, v4.xyzw

47: ret

}

}

CustomEditor "UnityEditor.ShaderGraph.GenericShaderGraphMaterialGUI"

Fallback "Hidden/Shader Graph/FallbackError"

}