УДК 004

Создание визуального эффекта на игровом движке Godot

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс создание шейдеров для пригодного создание визуальный эффект на игровом движке Godot. В работе использовался язык шейдера для рисования текстур и система частиц в конструкторе игр на движке Godot. В результате работы был создан шейдер, который рисует текстуру и система частиц для получение визуальный эффект на игровом движке Godot.
Ключевые слова: Godot, VFX, Particles system, GDShader.

Creating a character controller using the Godot game engine

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
student

Abstract
This article describes the process of creaters for suitable creating a visual effect on the Godot game engine. The work used the language of Shader for drawing textures and a particle system in the designer of games on the GODOT engine. As a result of the work, a shader was created that draws the texture and particle system for obtaining a visual effect on the Godot game engine.
Keywords: Godot, VFX, Particles system, GDShader.

1 Введение

1.1. Актуальность исследования

Актуальность исследование заключается в том, что язык шейдеров позволяет разработчикам создавать сложные и красочные визуальные эффекты, такие как реалистичные огонь, воду, свет, тени и многое другое. Это обеспечивает гибкость в создании уникальных и качественных визуальных эффектов для игр.

Использование шейдеров также позволяет оптимизировать производительность игры, поскольку они могут быть настроены для эффективного использования ресурсов, что особенно важно для мобильных устройств и слабых компьютеров.

Язык шейдеров также предоставляет разработчикам технические возможности для создания различных визуальных эффектов, таких как динамические и интерактивные анимации, эффекты освещения, деформации

объектов и многое другое, что делает его важным инструментом для разработки VFX.

Таким образом, использование языка шейдеров на игровом движке Godot остается актуальным и важным для создания качественных визуальных эффектов в играх.

1.2. Цель исследования

Целью работы создания шейдеры для рисования текстур для создание визуальных эффект на игровом движоке Godot.

1.3. Обзор исследований

Исследование Дж. Нордберга Основное внимание уделяется значимости визуальных эффектов (VFX) в видеоиграх, особенно в контексте мобильных игр. В исследовании подчеркивается роль визуальных эффектов в поддержке игрового процесса и создания захватывающего внутриигрового мира. Он упоминает примеры визуальных эффектов, таких как взрывы и падающие листья, подчеркивая их важность в улучшении игрового опыта на небольших экранах.

Исследование, вероятно, углубляется в проблемы и методы, связанные с созданием чистых и эффективных визуальных эффектов, специально предназначенных для среды для мобильных игр. Хотя конкретные детали исследования не представлены в результатах поиска, можно сделать вывод, что исследование направлено на решение уникальных соображений и ограничений, связанных с визуальными эффектами для мобильных игр, учитывая ограничения небольших экранов и мобильных аппаратных возможностей.

В целом, работа Дж. Нордберга предлагает ценную информацию о дизайне и реализации визуальных эффектов в мобильных играх, с акцентом на оптимизацию визуального опыта для игроков на небольших экранах [1].

Исследование, проведенное Т. Гуссенкура , Дж. Деллак, П. Бертолино., фокусируется на использовании игрового двигателя в качестве общей платформы для предварительной предварительной учебы в режиме фильма в контексте визуальных эффектов кино. В исследовании, вероятно, рассматриваются преимущества и проблемы использования игрового двигателя в качестве инструмента для визуализации в реальном времени во время производства визуальных эффектов в киноиндустрии.

Хотя конкретные детали исследования недоступны в результатах поиска, можно сделать вывод, что исследование направлено на продемонстрирование того, как можно использовать игровой двигатель для создания визуализаций в реальном времени сложных сцен и эффектов, позволяя кинематографистам принимать обоснованные решения и Корректировки во время производственного процесса. Этот подход может потенциально повысить эффективность и эффективность производства визуальных эффектов в кино.

В целом, Т. Гуссенкура , Дж. Деллак, П. Бертолино. Исследования, вероятно, дают ценную информацию о применении игровых двигателей в качестве платформы для предварительной просугулизации в режиме реального времени в контексте визуальных эффектов кино, предлагая потенциальные преимущества для кинематографистов и Визуальные эффекты художников в отрасли [2].

Исследование, проведенное М. Александров, С. Златанова, Д. Дж. Хеслоп., Вероятно, углубляется в концепцию вариантов вокселизации и управления вокселями в рамках платформы разработки игр Unity3D. Вокселизация включает представление объектов или средств с использованием объемных пикселей (вокселей), а не традиционных полигонов, и имеет различные приложения в разработке игр, включая генерацию местности, создание процедурного контента и визуальные эффекты.

Хотя конкретные детали исследования не доступны в результатах поиска, можно сделать вывод, что исследование направлено на изучение методов, проблем и потенциальных преимуществ вокселизации и управления вокселями в среде Unity3D. Это может включать обсуждения по оптимизации производительности, управление большими наборами данных вокселей и использование рендеринга на основе вокселей для захватывающего визуального опыта в играх, разработанных с использованием Unity3D.

В целом, М. Александров, С. Златанова, Д. Дж. Хеслоп., Вероятно, дает ценную информацию о практической реализации и соображениях, связанных с разработкой и дизайнам [3].

2. Рабочий процесс

Мы создали проект и назвали «vfx_godot» а затем создали 3D сцену и сохранили в файл «res://VFX/Main.tscn», в сцене добавили 3 объекта GPUParticles3D и назвали «Flame», «Sparks», «Smoke» (рис 1).

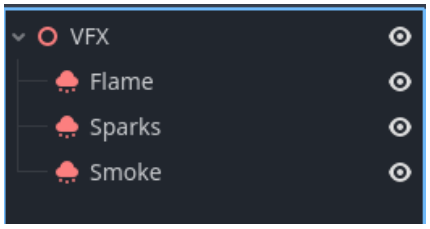

Рисунок 1. Дерево объектов на сцене.

В Инспекторе каждого объекта задали Amount (количество частиц) значение 100 и 100 и 50, а затем задали Time → «Fixed FPS» (частота кадров) на 60. А затем мы задали Explosiveness на 1. Это соотношение времени между каждым выбросом. Если 0, то частицы испускаются непрерывно. Если 1 то все частицы испускаются одновременно [4].

Частота кадров считает количество кадров за одну секунду будет обновляется частицы для визуализации.

В Draw Passes → Pass 1 создали объект PlaneMesh и задали в свойстве «Orientation» на «Face Z» (направление плоскость по осью). В каждом объекте «Flame», «Sparks», «Smoke» имеет один и тоже экземпляр созданный по типу PlaneMesh.

В GeometryInstance3D → Geometry → Material Override, мы создали объект ShaderMaterial и сохранили в файл «res://VFX/VFXNoise.gdshader».

Листинг 2.1. GDShader «VFXNoise.gdshader ».

```
shader_type spatial;
render_mode depth_draw_opaque,cull_back,diffuse_burley,specular_schlick_ggx,unshaded;

uniform sampler2D texture_albedo:source_color;
uniform float metallic;
uniform float specular;
uniform vec4 color:source_color = vec4(1.0);
uniform float emission = 1.0;
uniform float roughness:hint_range(0,1);
varying vec3 p_vertex;
varying vec3 p_normal;
uniform bool proximity_fade;

#include "res://Shader/Lib/Classic_Perlin.gdshaderinc"
#include "res://Shader/Lib/Noise.gdshaderinc"

float fbm3dc(vec3 uv, float frequency, float amplitude, float value, int octaves) {
	for(int i = 0; i < octaves; i++) {
		value += amplitude * cnoise(frequency * uv);
		amplitude *= 0.5;
		frequency *= 2.0;
	}
	return value;
}

void vertex() {
	mat4 mat_world = mat4(normalize(INV_VIEW_MATRIX[0])*length(MODEL_MATRIX[0]),normalize(INV_VIEW_MATRIX[1])*length(MODEL_MATRIX[0]),normalize(INV_VIEW_MATRIX[2])*length(MODEL_MATRIX[2]),MODEL_MATRIX[3]);
	mat_world = mat_world * mat4( vec4(cos(INSTANCE_CUSTOM.x),-sin(INSTANCE_CUSTOM.x), 0.0, 0.0), vec4(sin(INSTANCE_CUSTOM.x), cos(INSTANCE_CUSTOM.x), 0.0, 0.0),vec4(0.0, 0.0, 1.0, 0.0),vec4(0.0, 0.0, 0.0, 1.0));
	if (proximity_fade) {
		MODELVIEW_MATRIX = VIEW_MATRIX * mat_world;
	}
	p_vertex = VERTEX;
	p_normal = NORMAL;
}

void fragment() {
	vec3 uv3 = p_vertex;
	uv3 += mod289_3d(vec3(TIME*0.5)).xyz;
	float f = smoothstep(0.5, 1.0, 1.0-length(p_vertex));
	vec4 col = vec4(1.0) * (fbm3dc(uv3, 10.0, 1.0, 0.5, 2) + noise(uv3 * 10.0)) / 2.0 * f * COLOR;
	ALBEDO = col.rgb * color.rgb;
	METALLIC = metallic;
	ROUGHNESS = roughness;
	SPECULAR = specular;
	EMISSION = col.rgb * col.a;
	ALPHA = mix(0.0, col.a, f) * color.a;
}
```

Таблица 1. Параметры и переменный для шейдера (листинг 2.1).

Строка	Тип	Название	Описание
4	float	metallic	Металличность
5	float	specular	Блики
6	vec4	color	Основной цвет
7	float	emission	Излучение света
8	float	roughness	Шероховатость
9	vec3	p_vertex	Переменная полученный от вершинный шейдера, вершины
10	vec3	p_normal	Переменная полученный от вершинный шейдера, нормали
11	bool	proximity_fade	Если истинно то текстура всегда смотрит на камеру

Строка 14 — 15. Включает файл шейдеры (листинг 2.2 и листинг 2.3).

Строка 17 — 24. Генератор текстуры FBM (Фрактальное броуновское движение) [5].

Строка 26 — 34. Вершинный шейдер.

Строка 27 — 31. Вычисляется если proximity_fade установлено истинно то меш объект всегда «смотрит» на камеру.

Строка 32 — 33. Получение вершины и нормали.

Строка 36 — 47. Фрагментный шейдер.

Строка 37 — 38. Задаем вершинный координаты и смещение шума mod289_3d по времени.

Строка 39 - 40. fbm3dc (Фрактальное броуновское движение) сложение на noise (шум) и деленный на 2 и умноженный на f и вершинный цвет.

Строка 41. Поверхностный цвет.

Строка 42. Металличностный цвет.

Строка 43. Шероховатость.

Строка 44. Блики.

Строка 45. Цвет излучение света.

Строка 46. Прозрачность.

Мы создали шейдер и сохранили в файл «res://Shader/Lib/Classic_Perlin.gdshaderinc» (листинг 2.2).

Мы взяли готовый исходный код классический шум Перлина [6] (листинг 2.2).

Листинг 2.2. GDShader «Classic_Perlin.gdshaderinc» Классический шум Перлина.

```
1  //          Classic Perlin 3D Noise
2  //          by Stefan Gustavson
3  //
4  vec4 permute(vec4 x){return mod(((x*34.0)+1.0)*x, 289.0);}
```

```
vec4 taylorInvSqrt(vec4 r){return 1.79284291400159 - 0.85373472095314 * r;}
vec3 fade(vec3 t) {return t*t*t*(t*(t*6.0-15.0)+10.0);}

float cnoise(vec3 P) {
	vec3 Pi0 = floor(P); // Integer part for indexing
	vec3 Pi1 = Pi0 + vec3(1.0); // Integer part + 1
	Pi0 = mod(Pi0, 289.0);
	Pi1 = mod(Pi1, 289.0);
	vec3 Pf0 = fract(P); // Fractional part for interpolation
	vec3 Pf1 = Pf0 - vec3(1.0); // Fractional part - 1.0
	vec4 ix = vec4(Pi0.x, Pi1.x, Pi0.x, Pi1.x);
	vec4 iy = vec4(Pi0.yy, Pi1.yy);
	vec4 iz0 = Pi0.zzzz;
	vec4 iz1 = Pi1.zzzz;

	vec4 ixy = permute(permute(ix) + iy);
	vec4 ixy0 = permute(ixy + iz0);
	vec4 ixy1 = permute(ixy + iz1);

	vec4 gx0 = ixy0 / 7.0;
	vec4 gy0 = fract(floor(gx0) / 7.0) - 0.5;
	gx0 = fract(gx0);
	vec4 gz0 = vec4(0.5) - abs(gx0) - abs(gy0);
	vec4 sz0 = step(gz0, vec4(0.0));
	gx0 -= sz0 * (step(0.0, gx0) - 0.5);
	gy0 -= sz0 * (step(0.0, gy0) - 0.5);

	vec4 gx1 = ixy1 / 7.0;
	vec4 gy1 = fract(floor(gx1) / 7.0) - 0.5;
	gx1 = fract(gx1);
	vec4 gz1 = vec4(0.5) - abs(gx1) - abs(gy1);
	vec4 sz1 = step(gz1, vec4(0.0));
	gx1 -= sz1 * (step(0.0, gx1) - 0.5);
	gy1 -= sz1 * (step(0.0, gy1) - 0.5);

	vec3 g000 = vec3(gx0.x,gy0.x,gz0.x);
	vec3 g100 = vec3(gx0.y,gy0.y,gz0.y);
	vec3 g010 = vec3(gx0.z,gy0.z,gz0.z);
	vec3 g110 = vec3(gx0.w,gy0.w,gz0.w);
	vec3 g001 = vec3(gx1.x,gy1.x,gz1.x);
	vec3 g101 = vec3(gx1.y,gy1.y,gz1.y);
	vec3 g011 = vec3(gx1.z,gy1.z,gz1.z);
	vec3 g111 = vec3(gx1.w,gy1.w,gz1.w);

	vec4 norm0 = taylorInvSqrt(vec4(dot(g000, g000), dot(g010, g010), dot(g100, g100), dot(g110, g110)));
	g000 *= norm0.x;
	g010 *= norm0.y;
	g100 *= norm0.z;
	g110 *= norm0.w;
	vec4 norm1 = taylorInvSqrt(vec4(dot(g001, g001), dot(g011, g011), dot(g101, g101), dot(g111, g111)));
	g001 *= norm1.x;
	g011 *= norm1.y;
	g101 *= norm1.z;
	g111 *= norm1.w;

	float n000 = dot(g000, Pf0);
	float n100 = dot(g100, vec3(Pf1.x, Pf0.yz));
	float n010 = dot(g010, vec3(Pf0.x, Pf1.y, Pf0.z));
	float n110 = dot(g110, vec3(Pf1.xy, Pf0.z));
	float n001 = dot(g001, vec3(Pf0.xy, Pf1.z));
```

```
65          float n101 = dot(g101, vec3(Pf1.x, Pf0.y, Pf1.z));
66          float n011 = dot(g011, vec3(Pf0.x, Pf1.yz));
67          float n111 = dot(g111, Pf1);
68
69          vec3 fade_xyz = fade(Pf0);
70          vec4 n_z = mix(vec4(n000, n100, n010, n110), vec4(n001, n101, n011, n111), fade_xyz.z);
71          vec2 n_yz = mix(n_z.xy, n_z.zw, fade_xyz.y);
72          float n_xyz = mix(n_yz.x, n_yz.y, fade_xyz.x);
73          return 2.2 * n_xyz;
74  }
```

А затем создали еще один шейдер и сохранили в файл «res://Shader/Lib/Noise.gdshaderinc» (листинг 2.2), и взяли готовый исходный код шум [7] (листинг 2.3).

Листинг 2.3. GDShader «Noise.gdshaderinc» шум.

```
1   vec3 mod289_3d(vec3 x) {
2           return x - floor(x * (1.0 / 289.0)) * 289.0;
3   }
4
5   float mod289(float x){return x - floor(x * (1.0 / 289.0)) * 289.0;}
6   vec4 mod289_4d(vec4 x){return x - floor(x * (1.0 / 289.0)) * 289.0;}
7   vec4 perm(vec4 x){return mod289_4d(((x * 34.0) + 1.0) * x);}
8
9   float noise(vec3 p){
10          vec3 a = floor(p);
11          vec3 d = p - a;
12          d = d * d * (3.0 - 2.0 * d);
13
14          vec4 b = a.xxyy + vec4(0.0, 1.0, 0.0, 1.0);
15          vec4 k1 = perm(b.xyxy);
16          vec4 k2 = perm(k1.xyxy + b.zzww);
17
18          vec4 c = k2 + a.zzzz;
19          vec4 k3 = perm(c);
20          vec4 k4 = perm(c + 1.0);
21
22          vec4 o1 = fract(k3 * (1.0 / 41.0));
23          vec4 o2 = fract(k4 * (1.0 / 41.0));
24
25          vec4 o3 = o2 * d.z + o1 * (1.0 - d.z);
26          vec2 o4 = o3.yw * d.x + o3.xz * (1.0 - d.x);
27
28          return o4.y * d.y + o4.x * (1.0 - d.y);
29  }
```

Для объектов «Flame» и «Sparks» и «Smoke» в инспекторе свойства GeometryInstance3D → Geometry → Material Override, мы прилепили шейдер «VFXNoise.gdshader» (листинг 2.1). И в Shader Parameters задали Emission для «Flame» и «Sparks» на 5, а «Smoke» на 0.

3 Выводы

В данное статьи была создана 3 объекта система частиц с прилепленный на шейдеры и плоскость. В результате работе было написано на языке GDScript который рисует текстуру для пригодного использование система частиц визуальных эффектов (VFX).

Библиографический список

1. Nordberg J. Visual effects for mobile games: creating a clean visual effect for small screens. 2020.
2. De Goussencourt T., Dellac J., Bertolino P. A game engine as a generic platform for real-time previz-on-set in cinema visual effects //Advanced Concepts for Intelligent Vision Systems: 16th International Conference, ACIVS 2015, Catania, Italy, October 26-29, 2015. Proceedings 16. Springer International Publishing, 2015. С. 883-894.
3. Aleksandrov M., Zlatanova S., Heslop D. J. Voxelisation and Voxel Management Options in UNITY3D //ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2022. Т. 10. С. 13-20.
4. ParticleProcessMaterial — Godot Engine (stable) documentation in English GitHub URL: https://docs.godotengine.org/en/stable/classes/class_gpuparticles3d.html (дата обращения: 2024-01-27).
5. Fractal Brownian Motion (fBM) - Godot Shaders // Godot Shaders URL: https://godotshaders.com/snippet/fractal-brownian-motion-fbm/ (дата обращения: 2024-01-27).
6. Classic Perlin 3D Noise in glsl · GitHub URL: https://gist.github.com/moonraker22/bf2f62b8a720839289f29d5bb2884f16 (дата обращения: 2024-01-27).
7. GLSL Noise Algorithms GitHub URL: https://gist.github.com/patriciogonzalezvivo/670c22f3966e662d2f83 (дата обращения: 2024-01-27).