УДК 004.92

Создание 3D-модели ландшафта в игровом движке Godot

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс создания 3D модели ландшафт. В процессе работы использовалась программа Godot для создания 3D модели ландшафта и написания языка шейдера. В результате была выведена 3D графика ландшафта.
Ключевые слова: Godot, ландшафт, шейдер.

Creating a 3D model of the landscape in the Godot game engine

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of creating a 3D landscape model. In the process of work, the Godot program was used to create a 3D model of the landscape by writing the shader language. The result was a 3D landscape graphics.
Keywords: Godot, terrain, shader.

1 Введение

1.1 Актуальность исследования

Данная статья описывает возможность написания на языке шейдера моделирования ландшафта и визуализации 3D модели.

1.2 Цель исследования

Целью работы является создания 3D модели путем написания на языке шейдера получения ландшафта.

1.3 Обзор исследований

В работе Х. Мэкелэ Мелланде описывает процесс разработки японской игри риичи маджонг с использованием игрового движка Godot и включением среды .Net Framework, и языка программирования C#, и в игру включает пользовательские интерфейсы, шейдеры, многопользовательский режим с использования сетевой функции gine [1].

Ц. Андерссон предлагает решение системы перемотки времени в игре, которая позволит игроку вернуться в игровое время, чтобы воспроизвести определенный уровень или сеанс, который внедрит в игру включение игрового движка Godot [2].

Е. Беечинг и др. предлагают библиотеку Reinforcement Learning (RL) на игровом движке Godot для обучения и использования агента в игре для 3D и 2D среды, используют фреймворк с универсальными инструментами, который позволяет исследователям и геймдизайнерам создавать среды с дискретными, непрерывными и смешанными пространствами действий [3].

2. Рабочий процесс

В данной статье используется программа Godot версия 3.5.

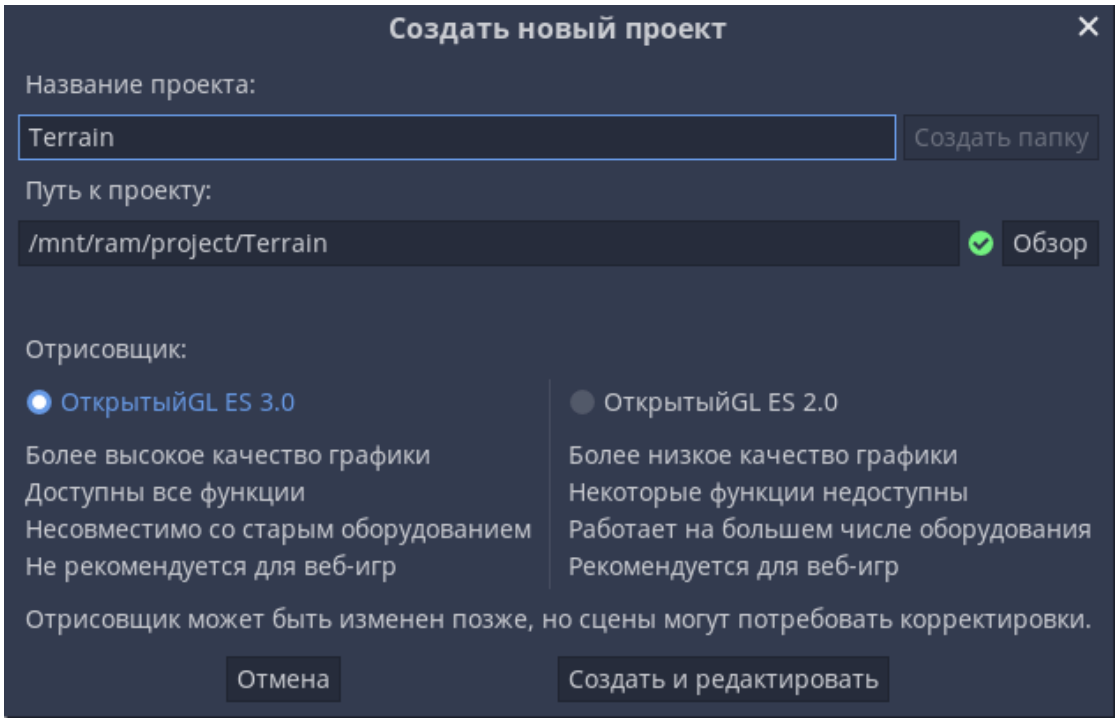

Рисунок 2.1. Начало работы

В работе создали проект и назвали «Terrain», и сцену, выбрав «3D сцена», назвали его «Main» и сохранили файл в путь «Level/Main» и назвали файл «Main».

В сцене создали два объекта «MeshInstance» и назвали «Terrain» и «Water».

«Terrain» - это будет ландшафт. Изначально он будет плоским объектом. Деформация и наложения поверхности будут написаны на языке шейдера.

«Water» - это вода.

В объекте «Terrain» в инспекторе Mesh создали «PlaneMesh» и задали значение Size 200, 200, и Subdivide Width и Depth 200, 200 для увеличения плотность точек внутри объекта.

В «Water» в инспекторе Mesh создали «CubeMesh» и задали Size 200, 8, 200.

В объекте «Terrain» создали материал, выбрав ShaderMaterial, и создали Shader, сохранив в отдельный файл (путь Shader/Terrain) (рис 2.2).

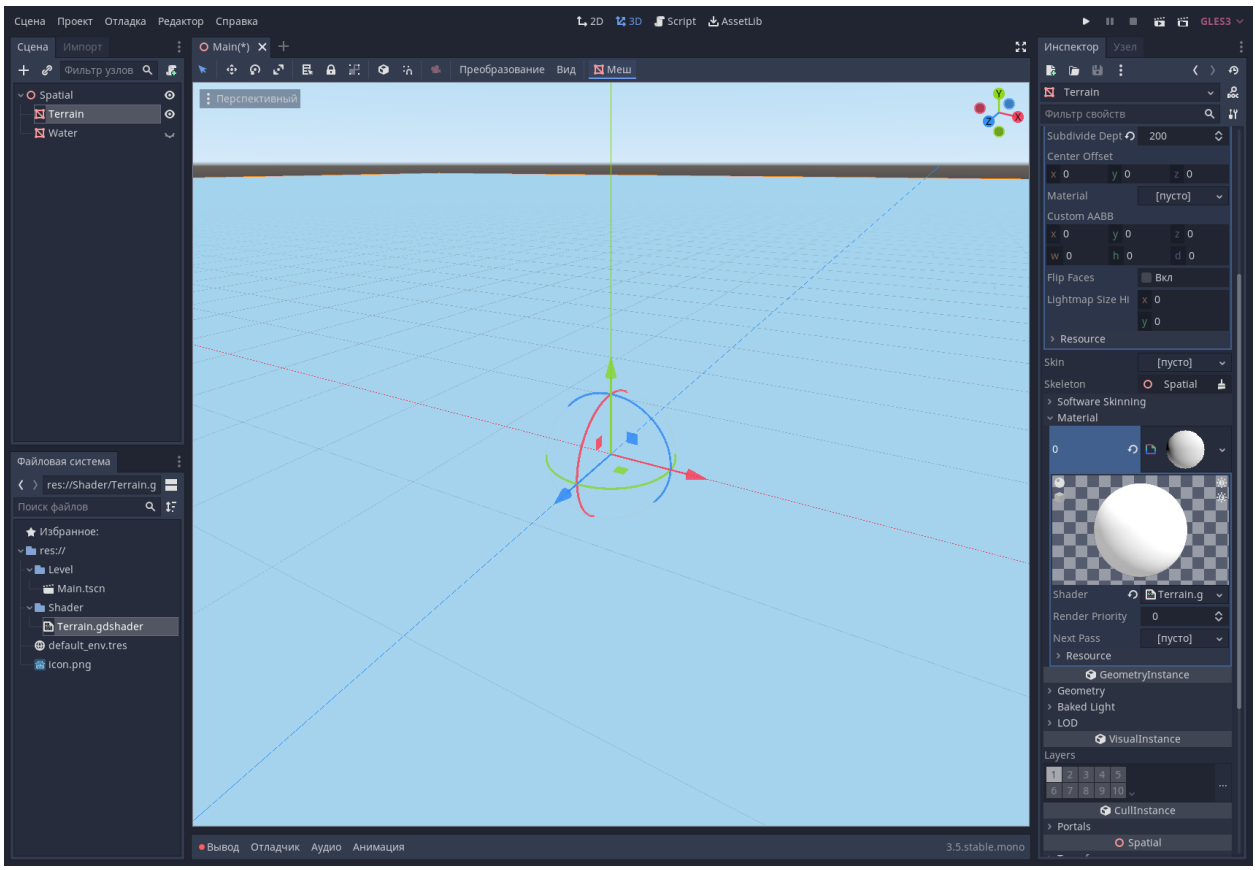

Рисунок 2.2. Созданы 2 объекты и материал

Листинг 4.1. В генерация шума использовали этот код (рис 2.3) (рис 2.4).

```
1  float random(vec2 n) {
2          return fract(sin(dot(n, vec2(12.9898, 4.1414))) * 43758.5453);
3  }
4  float noise(vec2 uv) {
5          vec2 uv_index = floor(uv);
6          vec2 uv_fract = fract(uv);
7
8          float a = random(uv_index);
9          float b = random(uv_index + vec2(1.0, 0.0));
10         float c = random(uv_index + vec2(0.0, 1.0));
11         float d = random(uv_index + vec2(1.0, 1.0));
12
13         vec2 blur = smoothstep(0.0, 1.0, uv_fract);
14         return mix(a, b, blur.x) +
15                 (c - a) * blur.y * (1.0 - blur.x) +
16                 (d - b) * blur.x * blur.y;
17 }
```

Исходный код (листинг 4.1) взят из [4, 5].

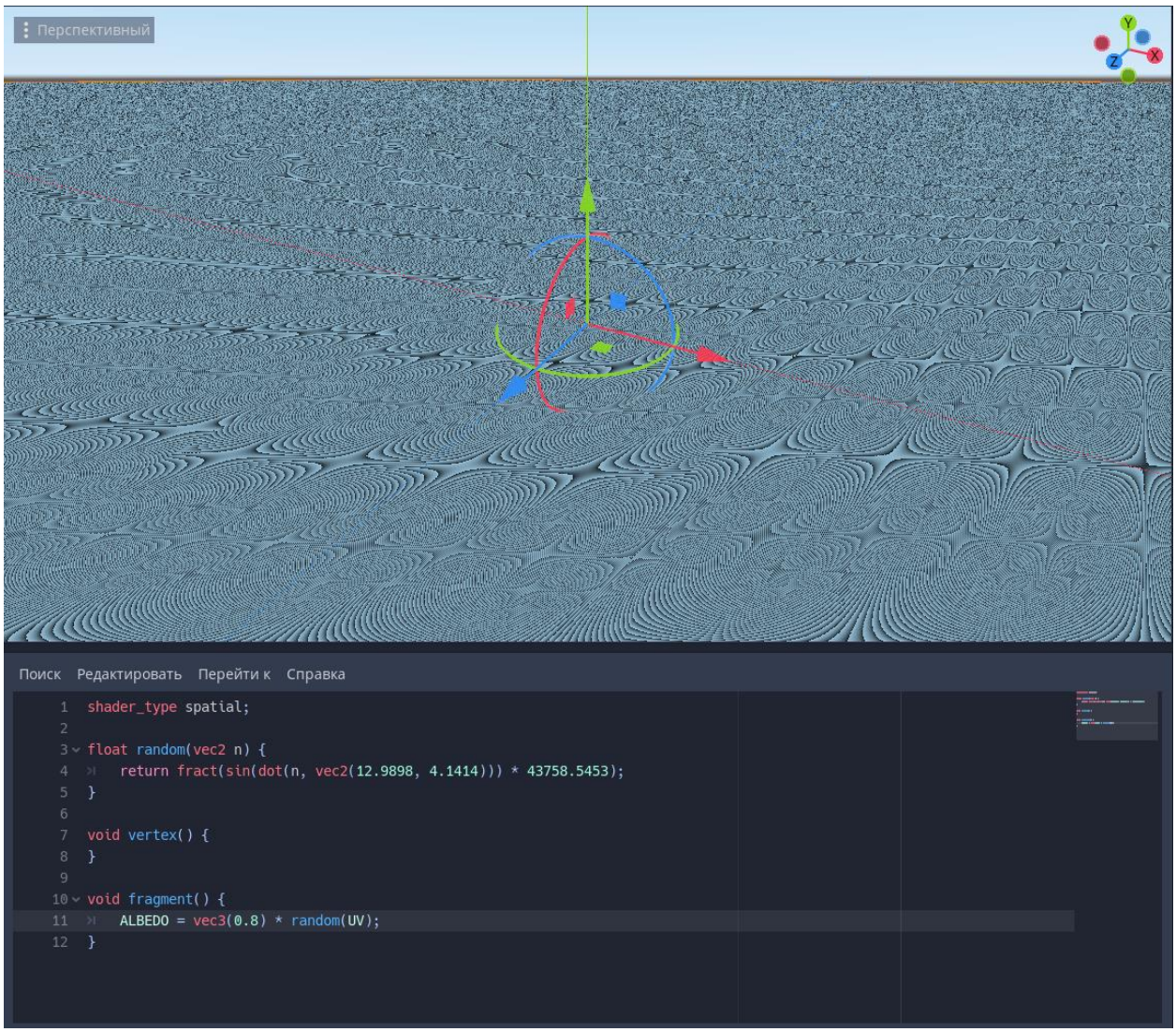

Рисунок 2.3. Использованы генератор шума Random

В листинге 2.1. random (рис 2.3) генерирует чистый шум, а noise генерирует шум (рис 2.4).

Переменная (листинг 2.1) uv_index полученная uv округления вниз, а uv_fract полученная по формуле x — floor(x) [6].

Переменные (листинг 2.1) a, b, c, d получаем из производных изображений [7].

В строке 13 (листинг 2.1) вычисляем интерполяцию Эрмита, получая плавный переход для переменный uv_fract, и получаем результат переменный blue [8].

В строке 14 (листинг 2.1), вычисляем производную изображения.

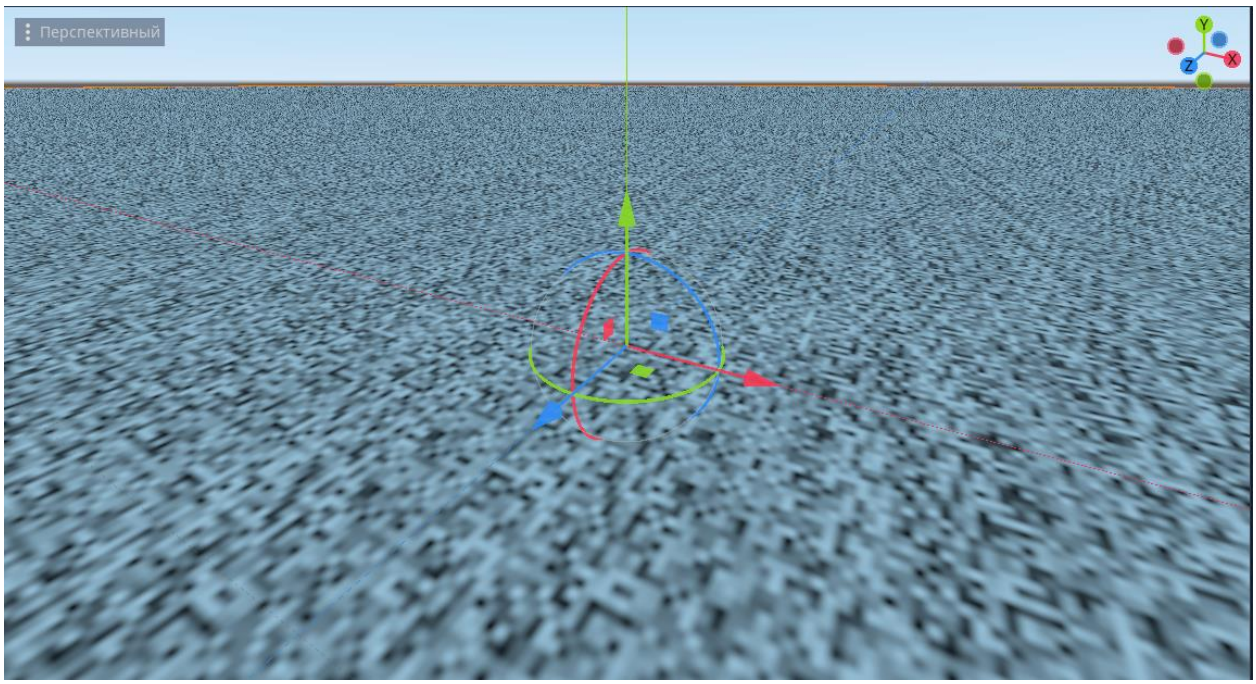

Рисунок 2.4. Использован генератор шума noise.
Координаты умножены на 10000

Листинг 4.2. Генератор фрактала для шума

```
1  float fbm(vec2 uv, float frequency, float amplitude, float value, int octaves) {
2          for(int i = 0; i < octaves; i++) {
3                  value += amplitude * noise(frequency * uv);
4                  amplitude *= 0.5;
5                  frequency *= 2.0;
6          }
7          return value;
8  }
```

fBM (Fractional Brownian Motion) (Фрактальное Броуновское движение) используется для генерации ландшафта, облако, горы, и др. [9].

frequency — частота, плотность горы, высокие значение - наибольшие количество горы и сопки (листинг 2.2).

amplitude — амплитуда, высокие значение - более острые горы, низкие значение - менее острые горы (листинг 2.2).

value — значение, высота ландшафта (листинг 2.2).

octaves — детализация, высокое значение - более детальнее горы (листинг 2.2).

Исходный код (Листинг 4.2) взят из [10].

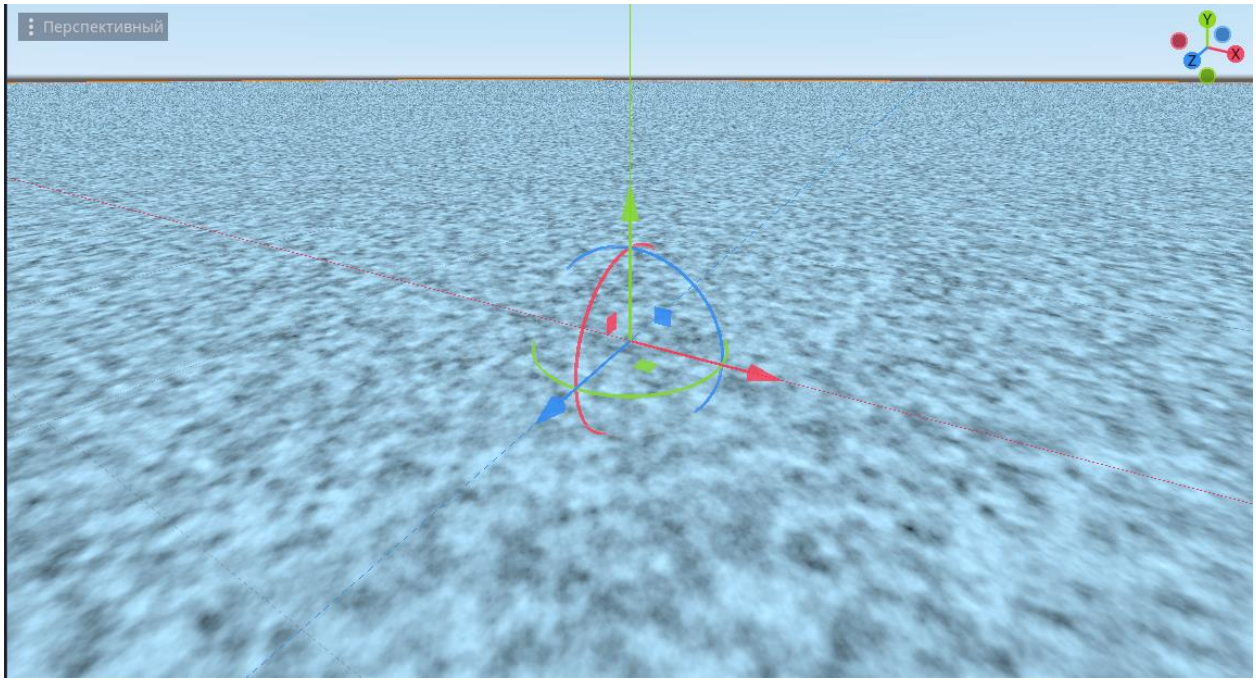

Рисунок 2.5. Использованя функция fBM(UV * 10000.0, 0.5, 1.0, 0.0, 5)

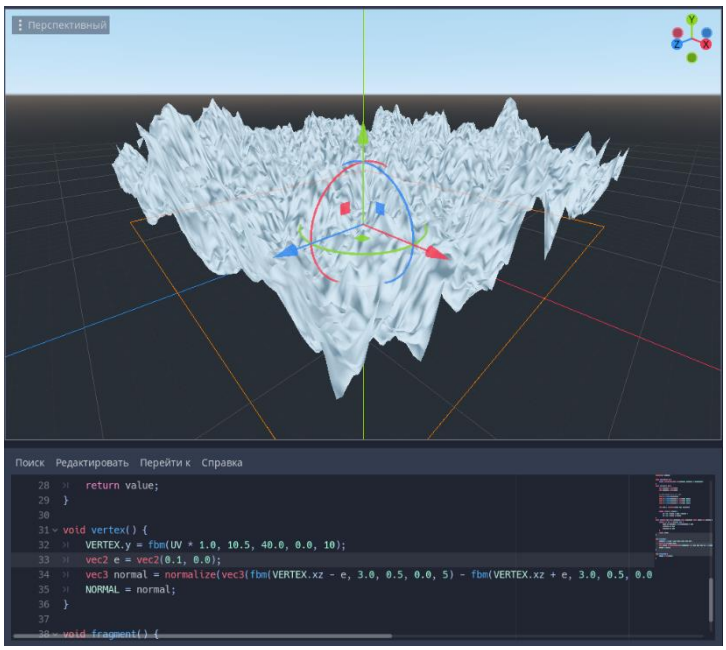

Рисунок 2.6. Использовано функция fBM для вершинный шейдер с добавления нормали и освещения

В сцене добавили еще один объект «DirectionalLight», и повернули на -30 С по оси X.

Листинг 4.3. Использование fBM для создания ландшафта

```
1  float land(vec2 uv) {
2          uv += 10.1;
3          float frequency = 6.0;
4          float amplitude = 0.58;
5          float value = 0.52;
```

```
6       int octaves = 5;
7       float v = fbm(uv * 1.0, frequency, amplitude, value, octaves);
8       vec2 df = vec2(1.0, 1.0);
9       vec2 uvdx = (uv + vec2(df.x, 0.0));
10      vec2 uvdy = (uv + vec2(0.0, df.y));
11      vec2 uvdxdy = (uv + df);
12
13      float vdx = 0.0;
14      float vdy = 0.0;
15      float vdxdy = 0.0;
16      for (int i = 0; i < 4; i++) {
17              uvdx = (uv + vec2(df.x, 0.0));
18              uvdy = (uv + vec2(0.0, df.y));
19              uvdxdy = (uv + df);
20              vdx = fbm(uvdx, frequency, amplitude, value, octaves);
21              vdy = fbm(uvdy, frequency, amplitude, value, octaves);
22              vdxdy = fbm(uvdxdy, frequency, amplitude, value, octaves);
23              v = (v + vdx + vdy + vdxdy) / 4.0;
24      }
25      return v;
26  }
```

В строке 2 (листинг 2.3) координаты uv сдвинуты на 10.1 обоих осей.
В строке 3 — 6, значения для аргумента функция fBM.
В строке 7 строим ландшафт.
В строке 8 — 15 определяем переменную производные изображения.
В строке 16 — 24 вычисляем производная изображения.
В строке 25 возвращаем результат, полученный из ландшафта.

Листинг 4.4. Вершинный шейдер, моделирование ландшафта.

```
1   void vertex() {
2       float v = land(UV);
3       float h = 80.0;
4       VERTEX.y = v * h - h;
5       vec2 e = vec2(0.95, 0.0);
6       vec3 normal = normalize(
7               vec3(
8                       fbm(VERTEX.xz - e, 1.0, 0.5, 0.0, 5) - fbm(VERTEX.xz +
    e, 1.0, 0.5, 0.0, 5),
9                       2.0 * e.x,
10                      fbm(VERTEX.xz - e.yx, 1.0, 0.5, 0.0, 5) - fbm(VERTEX.xz
11  + e.yx, 1.0, 0.5, 0.0, 5)
12                      )
13              );
```

```
14          NORMAL = normal;
15  }
```

В строке 2 (листинг 2.4) получаем переммену ландшафт.

В строке 3 Высота ландшафта.

В строке 4 Деформация вершин поверхность в умноженную на ландшафт на высоту и вычитается высота.

В строке 5 задаем детали для для затемнения.

В строке 7 — 13 создаем ландшафт для направления света (нормаль).

В строке 14 изменяем нормаль полученный результат ландшафт.

Листинг 4.5. Фрагментный шейдер, рисования ландшафта

```
1   void fragment() {
2           float v = land(UV);
3           vec3 col = vec3(0.0);
4           float n = v / 2.0;
5           vec3 col_sand = vec3(1.0, 0.684, 0.162);
6           vec3 col_land = vec3(0.15, 0.9, 0.0);
7           col_land = col_land * smoothstep(-0.4, 0.9, noise(vec2(sin(UV.x),
    cos(UV.y)) * 40. + noise(UV * 40.)));
8           float sand_h = 500.0;
9           float sand_noise = fbm(UV * sand_h, 8.0, 0.5, -0.5, 5) + fbm(UV +
    vec2(1.0, 0.0) * sand_h, 8.0, 0.5, -0.5, 5) + fbm(UV + vec2(0.0, 1.0) * sand_h,
    8.0, 0.5, -0.5, 5);
10          col_sand = mix(col_sand, vec3(0.287, 0.174, 0.076) * 0.0, sand_noise /
    2.0);
11          col = mix(col_sand, col_land, smoothstep(0.492, 0.50, n));
12          vec3 col_snow = vec3(0.85, 0.98, 1.0);
13          col_snow = col_snow * smoothstep(-0.5, 0.6, noise(UV * 20.0));
14          col = mix(col, col_snow, smoothstep(0.57, 0.591, n));
15          ALBEDO = col / 1.4;
16  }
```

В строке 2 (листинг 2.5) получаем ландшафт.

В строке 3 определяем переменную цвет.

В строке 4 определяем переменную для высоты слой ландшафта.

В строке 5 определяем цвет песок.

В строке 6 определяем цвет травы.

В строке 7 создаем эффект тени травы.

В строке 8 задаем затемнения ландшафта.

В строке 9 — 10 задаем детали песка.

В строке 11 добавляем слой песок и травой.

В строке 12 определяем цвет снега.

В строке 13 добавляем тени снега.

В строке 14 смешиваем слой снег в основной слой.
В строке 15 выводим цвет.
В результате работы получили вид рис 2.7.

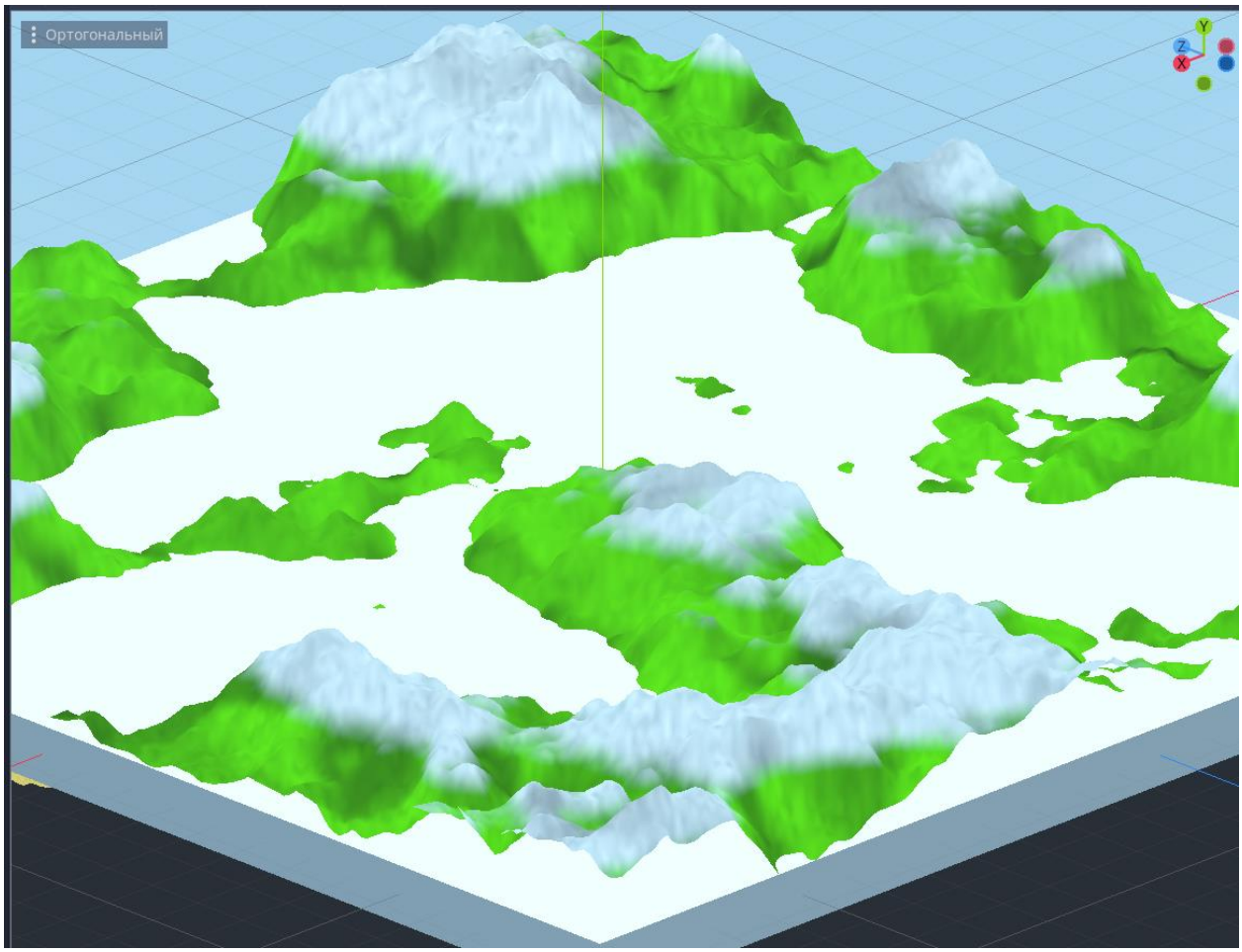

Рисунок 2.7. Ландшафт в ортогональный вид

В объекте «Water» (рис 2.7, белый цвет плоскость), создали материал по типу «SpatialMaterial» задали следующие свойство:

Названия	Значения	Вкладка	Описания
Cull Mode	Disabled	Parameters	Задняя отрисовка сторона объекта
Color	#008dff	Albedo	Цвет поверхности
Metallic	1	Metallic	Коэффициент отражение
Specular	1	Metallic	Коэффициент блеска
Texture	NoiseTexture	Metallic	Текстура отражения
Roughness	0	Roughness	Коэффициент шероховатости

Enable	1	Proximity Fade	Включает эффект затухания зависимости от расстояния.
Mode	PixelAlpha	Distance Fade	
Max Distance	8	Distance Fade	Расстояние эффекта затухания

В NoiseTexture создали Noise (OpenSimplexNoise) и задали следующие значение:

Названия	Значения	Вкладка	Описания
Seed	58	Noise	Зерно, для случайных значений
Octaves	5	Noise	Детализация
Period	6.2	Noise	Периоды

В вкладке «Spatial» в «Transform» задали значение -6 по осью Y.

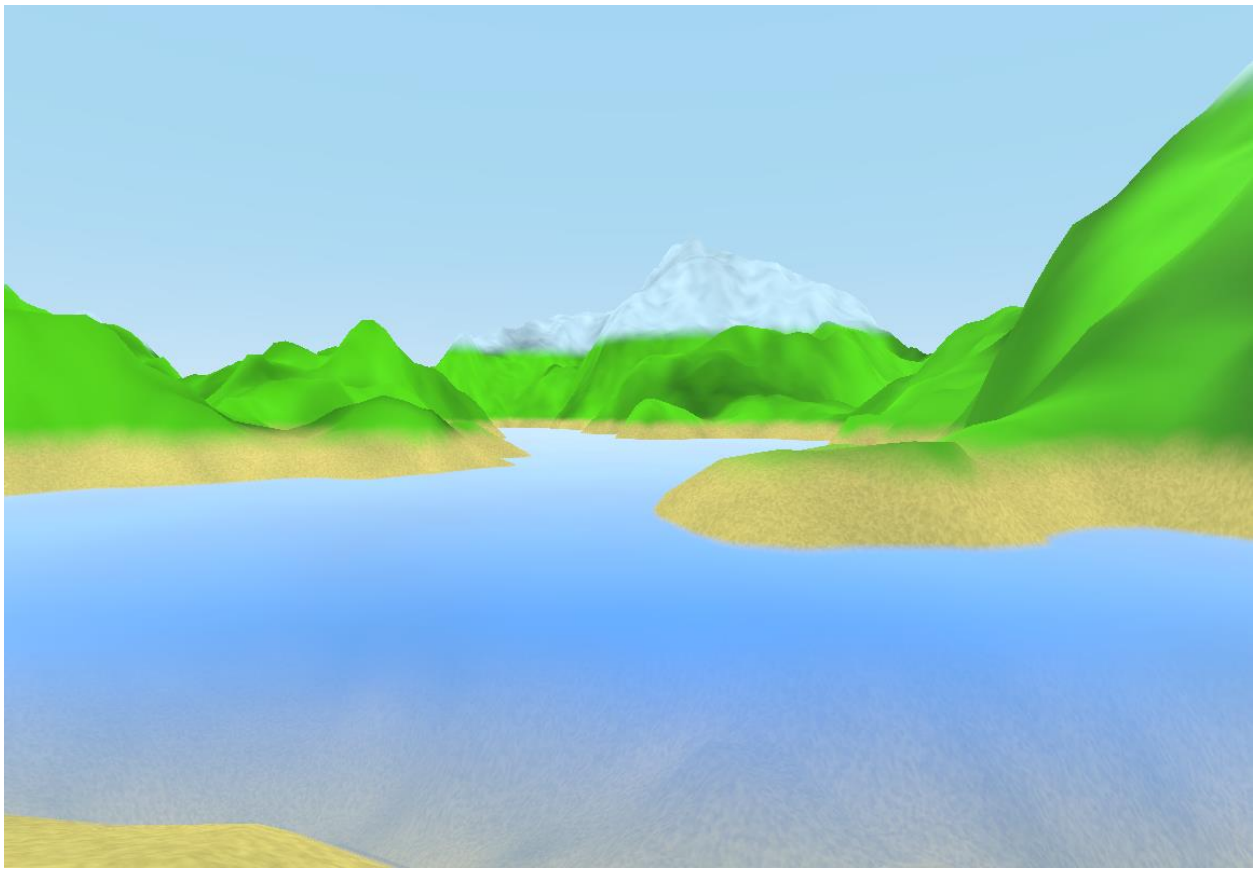

Рисунок 2.8. Готовый ландшафт

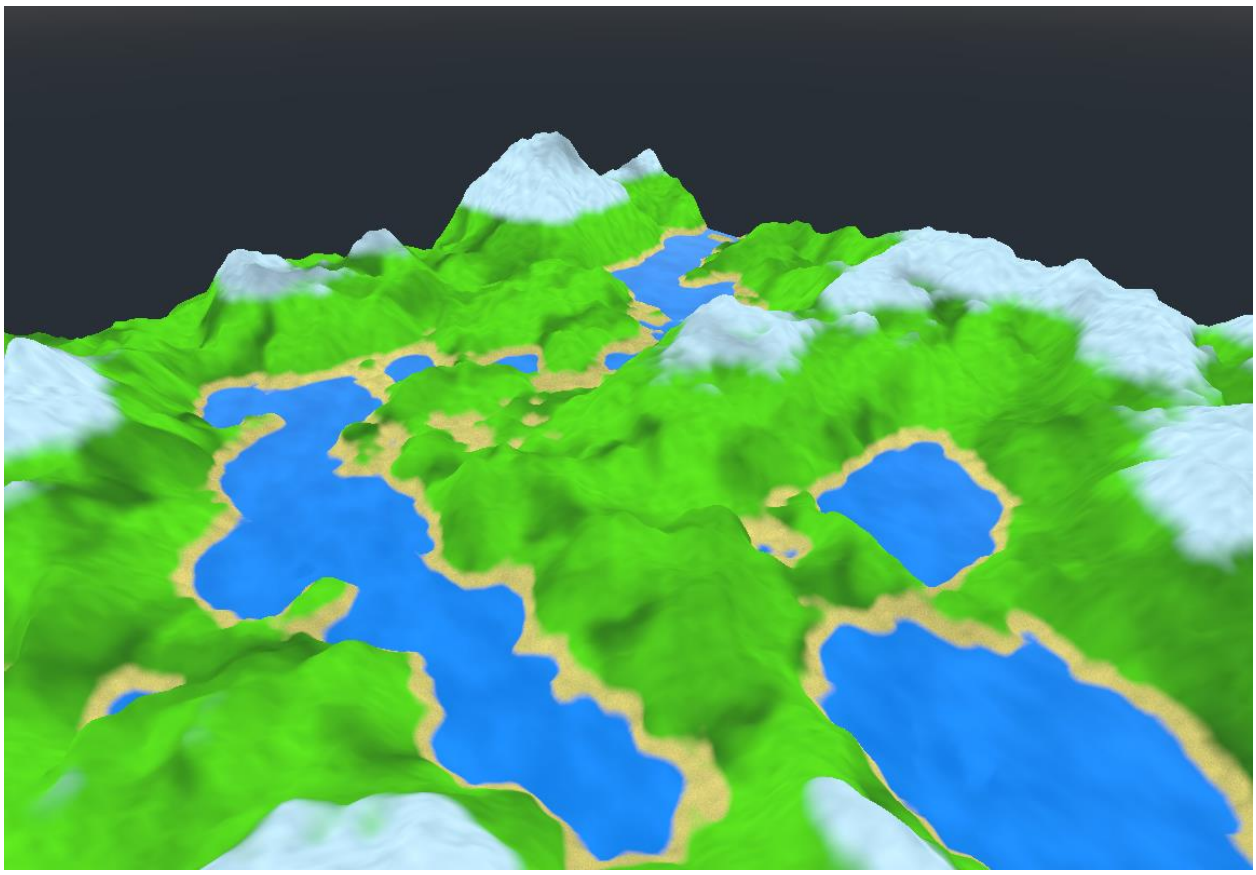

Рисунок 2.9. Готовый ландшафт в другой ракусе

3 Выводы

В результате работы использовалась программа Godot для создания модели и написания на языке шейдера получения ландшафта, использованы функция fBM и генератор шума для получения ландшафта, в результате получили 3D модель ландшафт.

Библиографический список

1. Mäkelä H. Development of a 3D mahjong video game in Godot Engine. 2021. URL: https://www.theseus.fi/handle/10024/502957
2. Andersson C., Mellander T. A 2D Game Rewind-System Using Godot Game Engine, Performance Comparison and Analysis. 2021.
3. Beeching E. et al. Godot Reinforcement Learning Agents // arXiv preprint arXiv:2112.03636. 2021.
4. GLSL Noise Algorithms GitHub // GitHub Gist URL: https://gist.github.com/patriciogonzalezvivo/670c22f3966e662d2f83 (дата обращения: 2022-09-03).
5. Perlin K. Improving noise // Proceedings of the 29th annual conference on Computer graphics and interactive techniques. 2002. С. 681-682.
6. fract - OpenGL 4 Reference Pages // Khronos URL: https://registry.khronos.org/OpenGL-Refpages/gl4/html/fract.xhtml (дата обращения: 2022-09-03).

7. An introduction to shader derivative functions | A Clockwork Berry // A Clockwork Berry URL: http://www.aclockworkberry.com/shader-derivative-functions/ (дата обращения: 2022-09-03).
8. smoothstep - OpenGL 4 Reference Pages // Khronos URL: https://registry.khronos.org/OpenGL-Refpages/gl4/html/smoothstep.xhtml (дата обращения: 2022-09-03).
9. Inigo Quilez :: computer graphics, mathematics, shaders, fractals, demoscene and more // Inigo Quilez URL: https://iquilezles.org/articles/fbm/ (дата обращения: 2022-09-03).
10. Fractal Brownian Motion (fBM) - Godot Shaders // Godot Shaders URL: https://godotshaders.com/snippet/fractal-brownian-motion-fbm/ (дата обращения: 2022-09-03).

4. Приложения

Листинг 4.1. Исходный код шейдера «Terrain»

```
shader_type spatial;

float random(vec2 n) {
	return fract(sin(dot(n, vec2(12.9898, 4.1414))) * 43758.5453);
}
float noise(vec2 uv) {
	vec2 uv_index = floor(uv);
	vec2 uv_fract = fract(uv);

	// Four corners in 2D of a tile
	float a = random(uv_index);
	float b = random(uv_index + vec2(1.0, 0.0));
	float c = random(uv_index + vec2(0.0, 1.0));
	float d = random(uv_index + vec2(1.0, 1.0));

	vec2 blur = smoothstep(0.0, 1.0, uv_fract);

	return mix(a, b, blur.x) +
		(c - a) * blur.y * (1.0 - blur.x) +
		(d - b) * blur.x * blur.y;
}
float fbm(vec2 uv, float frequency, float amplitude, float value, int octaves) {
	for(int i = 0; i < octaves; i++) {
		value += amplitude * noise(frequency * uv);
		amplitude *= 0.5;
		frequency *= 2.0;
	}
	return value;
}

float land(vec2 uv) {
	uv += 10.1;
	float frequency = 6.0;
	float amplitude = 0.58;
	float value = 0.52;
	int octaves = 5;
	float v = fbm(uv * 1.0, frequency, amplitude, value, octaves);
	vec2 df = vec2(1.0, 1.0);
	vec2 uvdx = (uv + vec2(df.x, 0.0));
	vec2 uvdy = (uv + vec2(0.0, df.y));
	vec2 uvdxdy = (uv + df);

	float vdx = 0.0;
	float vdy = 0.0;
	float vdxdy = 0.0;
	for (int i = 0; i < 4; i++) {
		uvdx = (uv + vec2(df.x, 0.0));
		uvdy = (uv + vec2(0.0, df.y));
		uvdxdy = (uv + df);
		vdx = fbm(uvdx, frequency, amplitude, value, octaves);
		vdy = fbm(uvdy, frequency, amplitude, value, octaves);
```

```
52                              vdxdy = fbm(uvdxdy, frequency, amplitude, value, octaves);
53                              v = (v + vdx + vdy + vdxdy) / 4.0;
54              }
55              return v;
56      }
57
58      void vertex() {
59              float v = land(UV);
60              float h = 80.0;
61              VERTEX.y = v * h - h;
62              vec2 e = vec2(0.95, 0.0);
63              vec3 normal = normalize(
64                      vec3(
65                              fbm(VERTEX.xz - e, 1.0, 0.5, 0.0, 5) - fbm(VERTEX.xz + e, 1.0, 0.5, 0.0, 5),
66                              2.0 * e.x,
67                              fbm(VERTEX.xz - e.yx, 1.0, 0.5, 0.0, 5) - fbm(VERTEX.xz + e.yx, 1.0, 0.5, 0.0, 5)
68                              )
69                      );
70              NORMAL = normal;
71      }
72
73      void fragment() {
74              float v = land(UV);
75              vec3 col = vec3(0.0);
76              float n = v / 2.0;
77              vec3 col_sand = vec3(1.0, 0.684, 0.162);
78              vec3 col_land = vec3(0.15, 0.9, 0.0);
79              col_land = col_land * smoothstep(-0.4, 0.9, noise(vec2(sin(UV.x), cos(UV.y)) * 40. + noise(UV * 40.)));
80              float sand_h = 500.0;
81              float sand_noise = fbm(UV * sand_h, 8.0, 0.5, -0.5, 5) + fbm(UV + vec2(1.0, 0.0) * sand_h, 8.0, 0.5, -0.5, 5) + fbm(UV +
        vec2(0.0, 1.0) * sand_h, 8.0, 0.5, -0.5, 5);
82              col_sand = mix(col_sand, vec3(0.287, 0.174, 0.076) * 0.0, sand_noise / 2.0);
83              col = mix(col_sand, col_land, smoothstep(0.492, 0.50, n));
84              vec3 col_snow = vec3(0.85, 0.98, 1.0);
85              col_snow = col_snow * smoothstep(-0.5, 0.6, noise(UV * 20.0));
86              col = mix(col, col_snow, smoothstep(0.57, 0.591, n));
87              ALBEDO = col / 1.4;
88      }
```