МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики  
и прикладных информационных технологий

**Технологии программирования компьютерной графики**

Отчёт по Лабораторной работе №2

Направление 01.04.02 Прикладная математика и информатика

Профиль Математические основы  
и программирование компьютерной графики

|  |  |  |  |
| --- | --- | --- | --- |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | И. Б. Рахимов |

Воронеж 2023

# Содержание

Содержание 2

Постановка задачи 3

Результаты 4

Список используемых источников 11

# Постановка задачи

Написать шейдер для учета нормалей из normal map. Использовать обе формулы:

* ;
* .

# Результаты

В этом отчете не буду писать про создание 3D объектов, материалов и шейдеров, так как писал про это в первом отчете.

Добавил новое свойство в Standard Surface Shader:

\_NormalMax ("Normal Map (RGB)", 2D) = "bump" {}

Так же добавил текстуру:

sampler2D \_NormalMax;

В функцию surf добавил:

fixed4 n = tex2D (\_NormalMax, IN.uv\_MainTex);

o.Normal = UnpackNormal(n);

Во входы материала необходимо добавить текстуру и карту нормали для неё, используемые мной текстуры рисунок 1, как добавить рисунок 2.

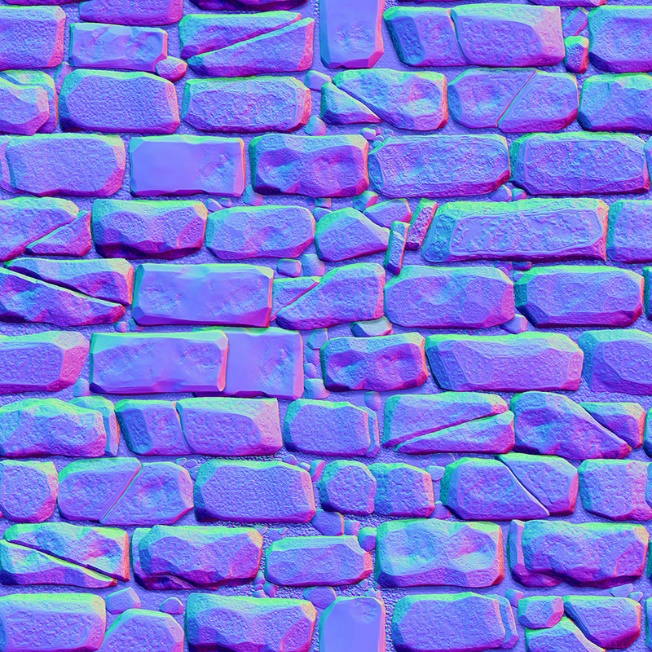


Рис. 1. Текстура и её карта нормали

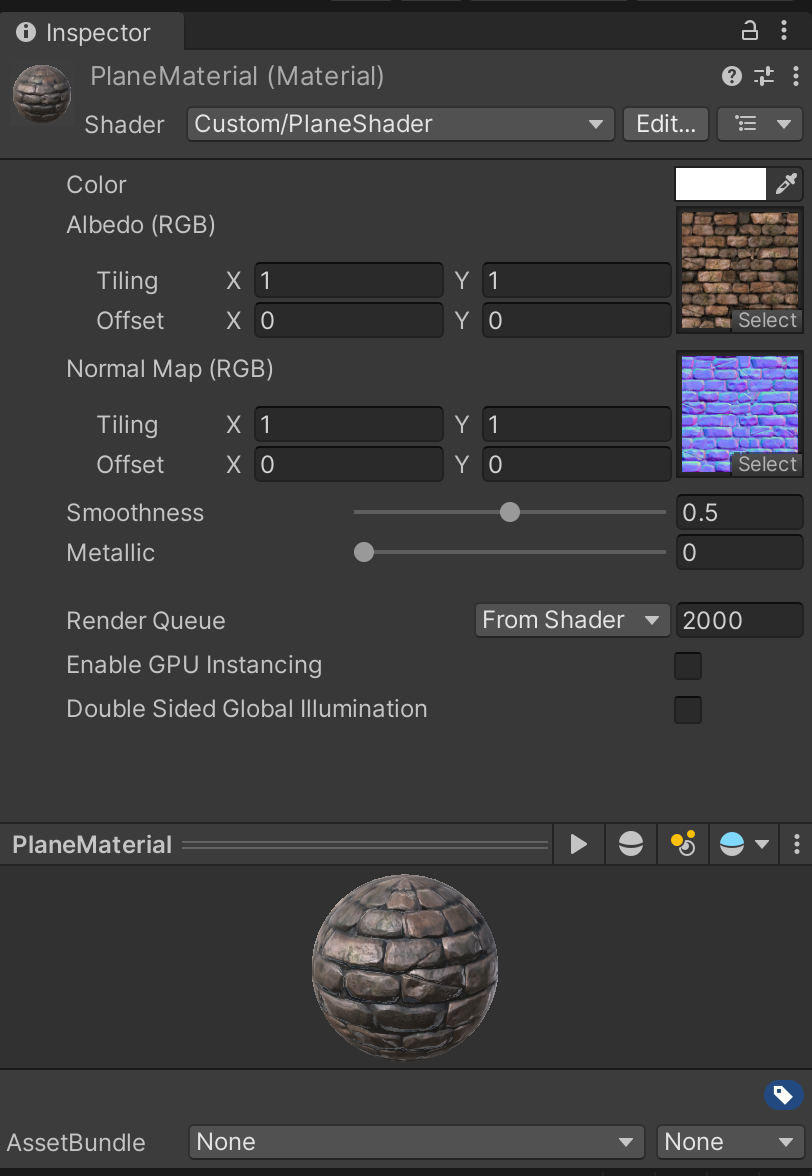


Рис. 2. Материал с установленными текстурой и картой нормали

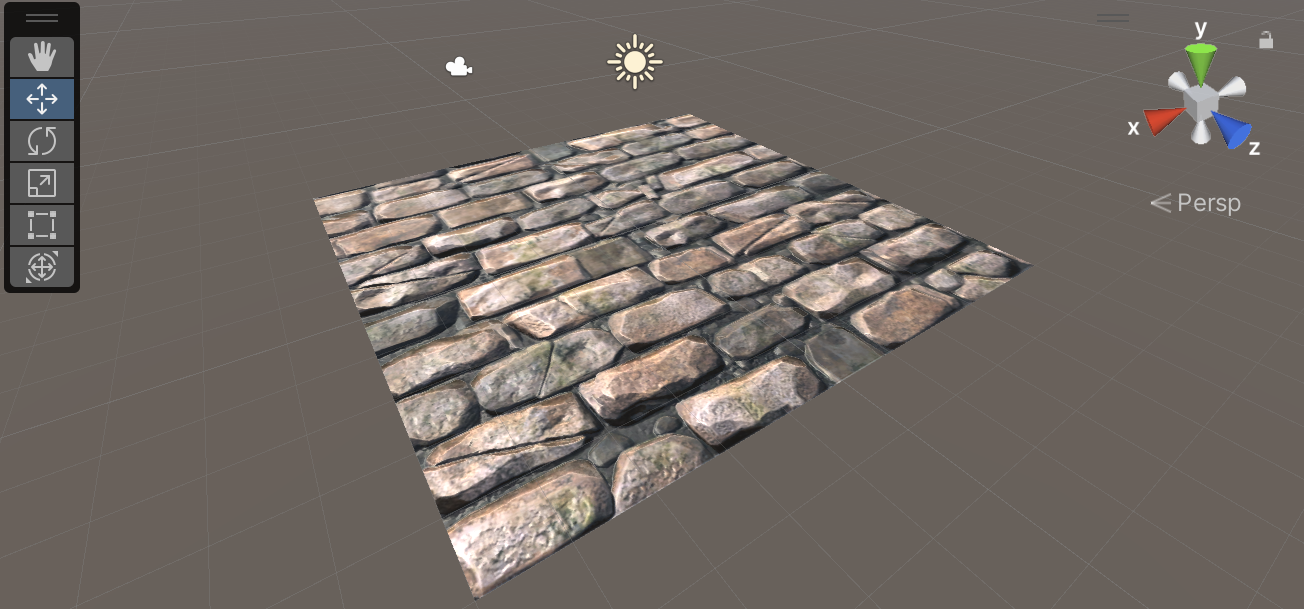


Рис. 3. Полученный результат

Избавимся от некоторых встроенных функций.

Для начала распакуем карту нормалей вручную используя формулы из [1]:

fixed3 unpack\_normal(sampler2D map, fixed2 uv)

{

fixed4 packed = tex2D(map, uv);

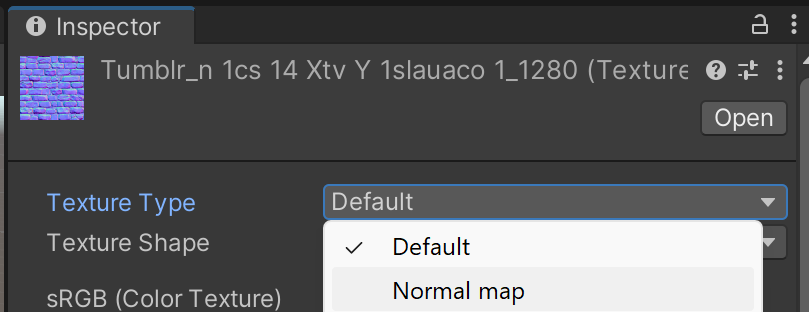
fixed4 unpacked = 2 \* packed - 1;

return normalize(unpacked.xzy);

}

Оси Z и Y меняются местами, потому что в карте нормали ось Z считается осью, направленной вверх, но в Unity ось направленная вверх является ось Y.

Так же для карты нормалей не стал ставить тип Normal map, чтобы Unity не конвертировал jpg в DXT5nm, чтобы не возится с формулами как в [2].



Добавим собственную реализацию освещения, без использования o.Normal. Для того чтобы узнать направление источника света в Unity есть переменная \_WorldSpaceLightPos0. Буду считать, что на сцене есть только стандартный Directional Light.

Следующая функция является реализацией освещения по Фонгу:

fixed4 phong(Input IN)

{

// light

fixed4 light\_color = \_LightColor0;

fixed3 light\_dir = normalize(\_WorldSpaceLightPos0);

// ambient

fixed4 ambient = \_ambient\_strength \* light\_color;

// diffuse

fixed3 n = unpack\_normal(\_NormalMap, IN.uv\_NormalMap);

float d = max(dot(n, light\_dir), 0.0);

fixed4 diffuse = d \* light\_color;

// specular

fixed3 view\_dir = world\_space\_view\_dir(IN.worldPos);

fixed3 reflect\_dir = reflect(-light\_dir, n);

float s = pow(max(dot(view\_dir, reflect\_dir), 0.0), 32);

fixed4 specular = \_specular\_strength \* s \* light\_color;

// object color

fixed4 albedo = tex2D(\_MainTex, IN.uv\_MainTex) \* \_Color;

return (ambient + specular + diffuse) \* albedo;

}

Так же для её правильной работы необходимо знать направление взгляда, его можно посчитать следующим образом:

fixed3 world\_space\_view\_dir(fixed3 object\_pos)

{

fixed3 camera\_pos = \_WorldSpaceCameraPos.xyz;

return normalize(camera\_pos - object\_pos);

}

И положение текущей точки в мировом пространстве для этого в структуру Input необходимо добавить следующее поле:

fixed3 worldPos;

Unity сам подставит в него правильное значение.

Так же для функции phong необходимы следующие свойства:

\_ambient\_strength ("ambient\_strength", Range(0,1)) = 0.0

\_specular\_strength ("specular\_strength", Range(0,1)) = 0.0

и переменные:

half \_ambient\_strength;

half \_specular\_strength;

Освещение практически работает, но так как нормали читаются из карты нормалей на освещение (diffuse, specular), использующее эти нормали, никак не влияют трансформации объекта. Чтобы это исправить используется TBN матрица, векторы для её столбцов можно получить из вершинного шейдера следующим образом:

void vert(inout appdata\_full v, out Input o)

{

UNITY\_INITIALIZE\_OUTPUT(Input, o);

o.normal = normalize(UnityObjectToWorldNormal(v.normal));

o.tangent = normalize(UnityObjectToWorldDir(v.tangent.xyz));

o.binormal = normalize(binormal(o.normal, o.tangent.xyz));

}

Для перевода нормали и тангента в мировые координаты используются стандартные функции Unity, про их реализацию можно почитать в [1-2] (матрица нормали и всё такое), самому писать было лень.

Для хранения этих данных в структуру Input необходимо добавить следующие поля:

fixed3 normal;

fixed3 tangent;

fixed3 binormal;

Для расчёта бинормали используется следующая функция:

fixed3 binormal(fixed3 normal, fixed3 tangent)

{

return cross(normal, tangent);

}

Так же Unity необходимо указать что функцию vert будет использоваться как вершинный шейдер, это делается следующим образом:

#pragma surface surf Standard fullforwardshadows vertex:vert

Так же в какой-то момент, кажется, в этот, Unity не мог передать такое количество данных из шейдера в шейдер, пришлось обновить версию шейдера, подробнее [3]:

#pragma target 3.5

Так как ориентация TBN матриц совпадает с ориентацией нормали в карте нормалей обмен оси Z и Y в функции unpack\_normal можно удалить. То есть она примет вид:

fixed3 unpack\_normal(sampler2D map, fixed2 uv)

{

fixed4 packed = tex2D(map, uv);

fixed4 unpacked = 2 \* packed - 1;

return normalize(unpacked.xyz);

}

Функция для получения TBN матрицы:

fixed3x3 getTbn(Input IN)

{

return fixed3x3(IN.tangent.x, IN.binormal.x, IN.normal.x,

IN.tangent.y, IN.binormal.y, IN.normal.y,

IN.tangent.z, IN.binormal.z, IN.normal.z);

}

Её можно использовать двумя способами:

1. Умножить нормаль из карты нормали на TBN матрицу, перед использование в освещении:

fixed3x3 tbn = getTbn(IN);

n = mul(tbn, n);

1. Умножить направление взгляда и освещения на обратную TBN матрицу, перед использование в освещении:

fixed3x3 tbn = getTbn(IN);

fixed3x3 tbn\_inv = transpose(tbn);

fixed3 light\_dir = normalize(\_WorldSpaceLightPos0);

light\_dir = mul(tbn\_inv, light\_dir);

fixed3 view\_dir = world\_space\_view\_dir(IN.pos);

view\_dir = mul(tbn\_inv, view\_dir);

Применена функция transpose() вместо inverse(). Такая подстановка справедлива, поскольку для ортогональных матриц (где все оси представлены единичными взаимно перпендикулярными векторами) получение обратной матрицы дает результат идентичный транспонированию. И это весьма кстати, поскольку, в общем случае, вычисление обратной матрицы куда более вычислительно затратное дело по сравнению с транспонированием [1].

Итог можно дополнительно посмотреть здесь: <https://github.com/Ihtier-0/UniversityAssignments/tree/main/Unity/lab2>

# Список используемых источников

1. <https://habr.com/ru/articles/415579/>
2. <https://catlikecoding.com/unity/tutorials/rendering/part-6/>
3. <https://stackoverflow.com/questions/52827701/too-many-texture-interpolators-would-be-used-shader-with-rotation>