УДК 004.42+004.925

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ ШЕЙДЕРОВ

Метелица Д.С., студент группы 881071,
Савенко А.Г., старший преподаватель кафедры ИСиТ

Белорусский государственный университет информатики и радиоэлектроники,
Институт информационных технологий,
г. Минск, Республика Беларусь

Савенко А.Г. – магистр техн. наук

Аннотация. В работе представлено программное средство для визуального программирования шейдеров, предназначенное для автоматизации процесса визуального описания шейдеров техническими художниками и программистами с целью ускорения процесса прототипирования. Использование данного средства техническими художниками позволит уменьшить взаимодействие художник – программист в процессе работы над проектом, т. к. программное средство не требует от пользователя знания программирования или иных специфических навыков. Итогом программирования шейдера станет экспорт в GLSL-код, совместимый с большим количеством игровых движков.

Ключевые слова. Программирование шейдеров, визуальное описание, GLSL-код

Введение. Развитие технологий ведёт не только к упрощению труда современного человека, но и к повышению качества сферы развлечений. Компьютерные игры и графика в кинематографе давно стали привлекательны не только для детей, но и для гораздо более широкой аудитории. Потребитель предъявляет все больше требований к качеству цифрового контента, уровню моделирования изображений, качеству детализации графических изображений. Современная IT-сфера давно вышла за пределы решения сугубо математических задач.

Шейдер – это программа, выполняющаяся на графическом процессоре видеокарты (GPU). На вход шейдера подаются входные данные, содержащие информацию о координатах вершин, полигонах, нормалях, освещении, цвете вершин, UV(текстурных координатах) и т.д. Задача шейдера принять эти данные, обработать, и подать на выход конечный результат [1].

Основная часть. Одним из основных понятий программирования шейдеров является 3D модель (рисунок 1). Оно включает в себя два основных пункта: вершина и текстура. Каждая вершина имеет свои координаты, а также нормаль. Вершины объединены в полигоны при помощи рёбер, а полигоны, в свою очередь, образуют полигональную сетку (mesh). Текстура – это простое изображение, которое расположено на модели в соответствии с UV-координатами.

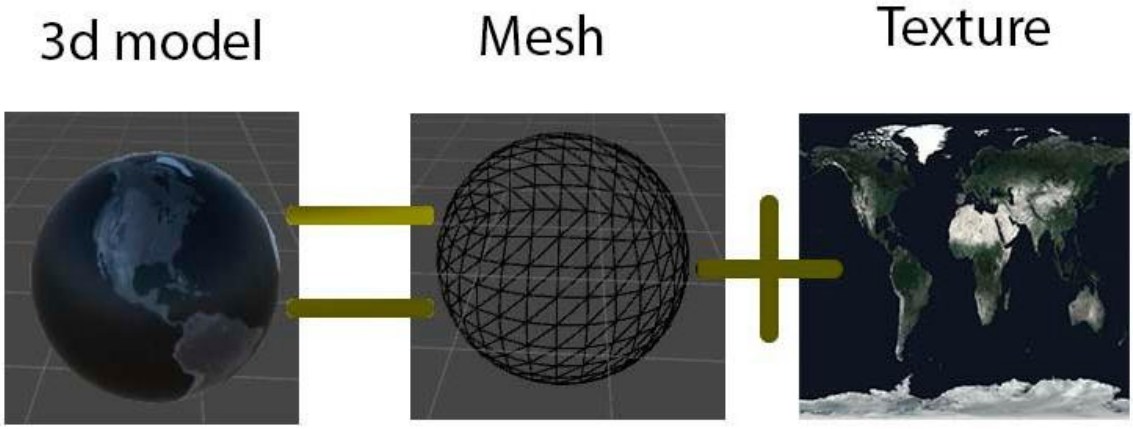

Рисунок 1 – 3D-модель

UV координаты – соответствие между координатами на поверхности трёхмерного объекта (X, Y, Z) и координатами на текстуре (U, V) (рисунок 2). Значения U и V обычно изменяются от **0** до **1**. Т.е. у каждой вершины модели есть свои соответствующие координаты на текстуре.

Шейдер выполняется для каждой отдельной вершины/пикселя отдельно. При этом он имеет информацию только о той вершине/пикселе которую сейчас обрабатывает. Т.е. во время написания шейдера мы не знаем, какого цвета соседняя вершина/пиксель.

Различают пиксельный и вертексный шейдеры.

Вертексный шейдер – шейдер, обрабатывающий данные о вершине и затем передающий их в пиксельный шейдер. Какие данные шейдер передаст, может задать программист. Вертексный шейдер выполняется раньше, чем пиксельный, а затем, из него передаются данные в пиксельный шейдер.

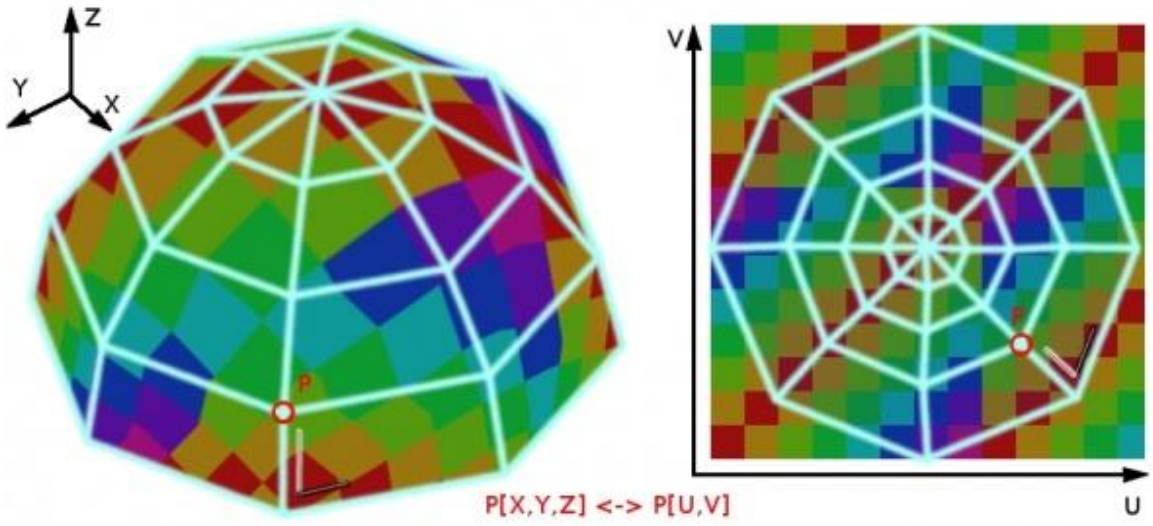

Рисунок 2 – Соответствие координат (x, y, z) и (u,v).

После передачи вертексным шейдером данных о вершине, эти данные интерполируются (сглаживаются) для каждого конкретного пикселя. Т.е. если пиксель стоит ровно между вершинами A (черная) и B (белая), цвет этого пикселя будет высчитан как средний от этих двух вершин(серый), и так для каждого параметра [2].

Пиксельный шейдер – шейдер, принимающий интерполированные данные от вертексного шейдера, и на их основе вычисляющий цвет для каждого отдельного пикселя [2].

Алгоритм работы шейдера:

— шаг1. Прием входных данных.
— шаг2. Обработка их на вертексном шейдере.
— шаг3. Интерполяция данных.
— шаг4. Передача данных пиксельному шейдеру.
— шаг5. Обработка данных на пиксельном шейдере.
— шаг6. Вывод конечного результата(изображения).

Помимо вышеперечисленных, существуют еще два вида шейдеров: геометрические (работают с целыми примитивами модели) и поверхностные (surface – упрощенный вид шейдера в Unity 3D, в основном использующийся для работы с освещением).

На рисунках 3 и 4 показана упрощенная модель графического конвейера, которая условно делится на две части: обработка геометрии и обработка фрагментов (пикселей).

Рисунок 3 – Обработка геометрии (вершинный шейдер)

На первой стадии графического конвейера выполняется преобразование координат вершины, перевод вершины в пространство отсечения, расчет освещения, применение материалов, определение цвета каждой вершины с учетом всех источников света и генерация текстурных координат. После выполнения этих операций наступает компоновка примитива. В этой части конвейера вершины группируются в треугольники и подаются в растеризатор.

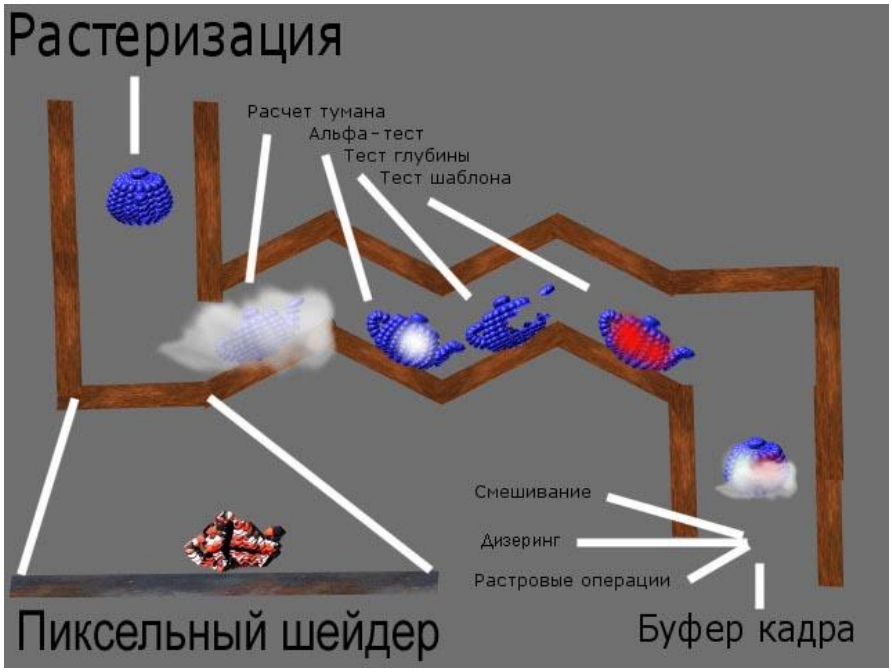

Рисунок 4 – Обработка фрагментов (пиксельный шейдер)

Растеризатор делит треугольник на фрагменты (пиксели), для которых интерполируются текстурные координаты и цвет. Затем для каждого фрагмента происходит выполнение следующих операций: проверка принадлежности пикселя, наложение текстур (заданные для фрагмента координаты текстуры определяют интерполированный цвет из элементов текстурного изображения – текселей, значение этого цвета комбинируется с цветом фрагмента), применение эффектов тумана, альфа-тест, тест шаблона (stencil-test), тест глубины, смешивание, дизеринг и логические операции. После обработки всех этих методов полученный фрагмент помещается в буфер кадра, который впоследствии выводиться на экран. [3]

Существуют такие языки программирования шейдеров:

— GLSL – высокоуровневый язык программирования для OpenGL;
— HLSL – высокоуровневый язык программирования для DirectX;
— CG – высокоуровневый язык программирования, в зависимости от ситуации компилируется под HLSL/GLSL. Используется в Unity.

Любой язык программирования, на котором пишутся шейдеры, является C-подобным, но в то же время более гибким в плане работы с переменными. Его основная задача: вычисления с плавающей запятой, матрицами и векторами. Максимальная эффективность и скорость работы шейдерной программы зависит от ряда условий, а именно:

— программа должна быть простой и короткой;
— чем меньше вычислений, тем быстрее будет работать;
— чем более линейная программа, тем быстрее будет работать шейдер;
— не приветствуется использование нелинейных операторов: циклов и условий.

Типы данных, используемые при построении программ шейдеров, делятся на категории:

— целочисленные и логические (bool, int, half), используются достаточно редко;
— с плавающей запятой (float, double);
— векторы (float2, float3, float4, double3, bool2 и т.п., или vector<float, 3>);
— матрицы (float2x2, float3x3, float4x4, double2x2, int4x3, или matrix<float, 2, 2> и т.п.);

В языке существует оператор «точка». Он позволяет работать с векторами и матрицами.

Все переменные, с которыми работает шейдерная программа, делятся на шесть видов:

1) входные параметры вертексного шейдера;
2) интерполированные входные параметры пиксельного шейдера;
3) текстурные сэмплеры;
4) переменные, которым GPU автоматически присваивает значение (встроенные переменные);
5) временные переменные пользователя;
6) входные параметры, переданные из основной программы.

С учетом вышеизложенного было разработано программное средство для визуального программирования шейдеров, позволяющее уменьшить взаимодействие «художник – программист» в процессе работы над проектом, т. к. программное средство не требует от пользователя знания программирования или иных специфических навыков.

Разработка программного средства осуществлялась на языке C++.

В ходе разработки реализованы следующие модули:

— UI – отвечает за отрисовку, взаимодействие с интерфейсом пользователя и графом, строимым в процессе создания шейдера.

— Generator - генерирует GLSL код по полученному в ходе использования программы графу.

— Render – отрисовывает круг/квадрат с использованием базового или сгенерированного шейдера.

— Serialisation – сохраняет описание графа в файл, считывает из файла описание графа и создает необходимые объекты на основе этого описания.

— Utils – набор утилитных методов и структур.

Схема взаимодействия модулей программного средства представлена на рисунке 5.

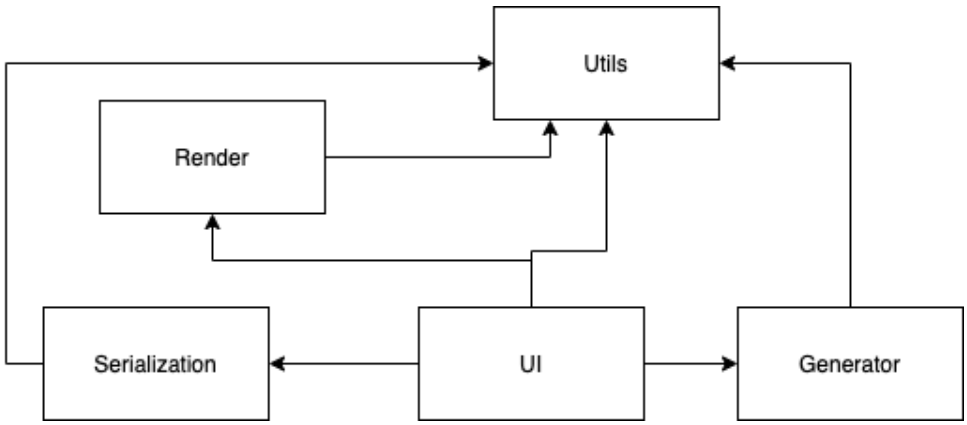

Рисунок 5 – Схема взаимодействия модулей программного средства

В основе разработки шейдеров лежат математические алгоритмы обработки графов. Граф шейдера представляет собой графический интерфейс на основе узлов, который позволяет дизайнерам и художникам добавлять и подключать узлы для создания шейдера без необходимости писать какой-либо код. Основная идея заключается в следующем: граф шейдера – это графическое представление фрагментного шейдера (в данном случае GLSL). Каждый узел в этом графе представляет собой текстовый блок в коде GLSL. Например, узел Multiply принимает два числа с плавающей запятой либо вектора и выдает результат их перемножения. В этом случае используется простой текстовый формат для указания узлов, вместо описания каждого узла классом. Такой подход имеет преимущества: узлы могут быть реализованы быстро, и узлы могут быть изменены или добавлены без перекомпиляции движка. Узлы и ссылки строимого в процессе создания шейдера графа – это данные. Для их описания используются структуры. На рисунке 6 приведён псевдокод алгоритма обхода нодов шейдера.

```
function resolve_node(node):
let input_code = ""
let this_code = node.code
store_outputs(node) // create variables for each output
for i in node.inputs:
if i.connected_node is resolved:
replace(this_code, i, i.connected_output)
else:
input_code += resolve_node(i.connected_node)
replace(this_code, i, i.connected_output)
return input_code + this_code
let output = resolve_node(master_node)
```

Рисунок 6 – Схема взаимодействия модулей программного средства

Для генерации кода необходима «входная» точка, начиная с которой будет обходиться граф. По линии связи определяется нод, с которым она связана и для того нода так же найти с котором связан тот (т.е. обходим смежные вершины графа последовательно). Таким образом, изначально доходим до нода без связей на входе записываем его код в строку, и так пока не вернемся обратно к ноде вывода цвета, и, собственно, получаем код, описанный графом.

Интерфейс программного средства похож на известные и распространенные редакторы шейдеров. Рабочая область представляет собой сетку, слева – область предпросмотра, вверху – главное меню. Добавление узлов графа шейдера осуществляется посредством контекстного меню (нажатие правой кнопкой мыши и выбор вида узла) (рисунок 7).

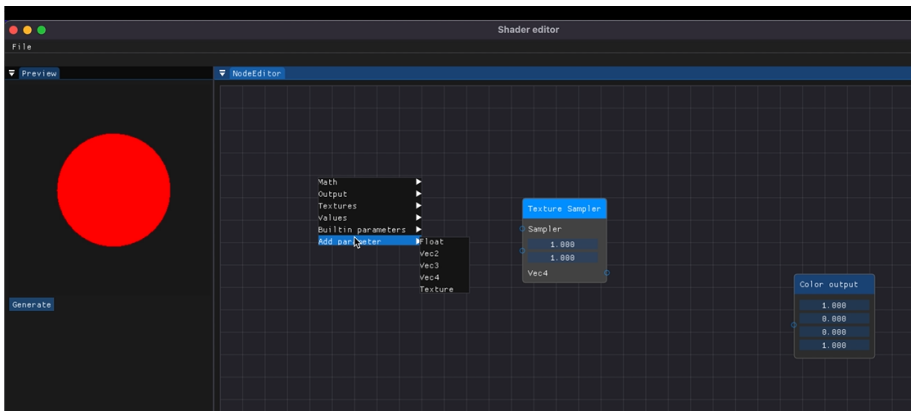

Рисунок 7 – Добавление узлов

При добавлении текстур в граф используется диалоговое окно, позволяющее загружать текстуры из файлов. Каждый пиксель геометрического объекта окрашивается соответственным цветом текстуры. На выходе получается эффект объемной текстуры, корректно воспринимаемый зрителем. На рисунке 8 показано, как текстура применена к геометрии объекта.

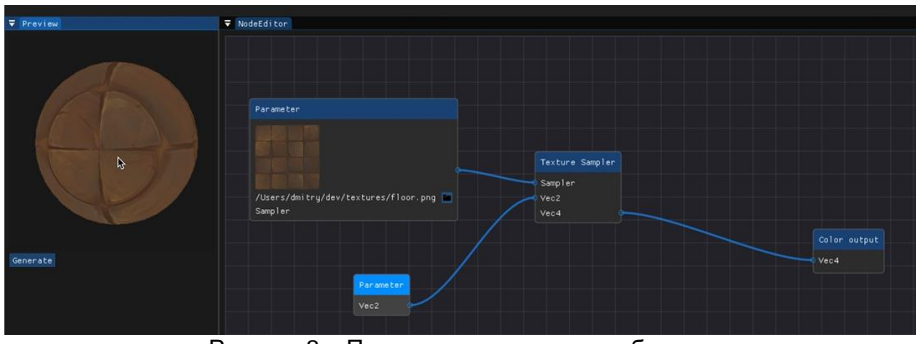

Рисунок 8 – Применение текстуры к объекту

Заключение. Одной из важнейших функций проекта является экспорт разработанного шейдера в GLSL-код. Именно эта возможность позволяет совмещать создаваемые в программе шейдеры с различными настриваемыми движками. Отдельно стоит упомянуть, что использование данного ПС не требует наличия интернета и установки какого-либо дополнительного программного обеспечения.

В результате было разработано программное средство для визуального программирования шейдеров, которое практически не требует от пользователя знания программирования и обладает удобным минималистичным интерфейсом и сдержанным, деловым стилем оформления.

Список использованных источников:

1 DevTribe. Основы программирования шейдеров [Электронный ресурс]. – Режим доступа: https://devtribe.ru/p/unity/quick-theory-of-shaders-1

2 Habr.com. Программирование шейдеров на Unity [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/474812

3 GameDev.ru Программирование шейдеров на HLSL [Электронный ресурс]. – Режим доступа: https://gamedev.ru/code/articles/HLSL

UDC 004.42+004.925

SOFTWARE TOOL FOR VISUAL SHADER PROGRAMMING

Metselitsa D.S., Savenko A.G.

Institute of Information Technologies of the Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Savenko A.G. – Senior Lecturer, Master of Engineering Sciences

Annotation. The paper presents a developed software tool for visual programming of shaders, designed to automate the process of visual description of shaders by technical artists and programmers to speed up the prototyping process. Using of this tool by technical artists will reduce the interaction between the artist and the programmer in the process of working on a project, since the software tool does not require programming knowledge or other specific skills from the user. The result of shader programming will be export to GLSL code, compatible with many game engines.

Keywords. Shader programming, visual description, GLSL code