УДК 004.921

Методы процедурной генерации и визуализации облаков в тренажерно-обучающих системах

Н. А. Агафонов, А. М. Гиацинтов, К. А. Мамросенко

Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", Москва, Россия.

Аннотация: В статье рассмотрен метод генерации облаков при помощи шумовой функции Перлина и билбордов. Приведен процесс создания шумовых текстур, использующихся для генерации карты облачности. Используя трехмерный шум Перлина возможно получить значительное количество генерируемых процедурно разнообразных карт облачности, а при использовании четырехмерного шума появляется возможность добавить временную составляющую для анимации облаков. Облака состоят из большого числа билбордов - полигонов, все время направленных на наблюдателя, и для их корректной визуализации необходима сортировка полупрозрачных поверхностей. Из-за большого количества билбордов производить сортировку на *CPU* ресурсозатратно, поэтому целесообразней перенести сортировку на *GPU*. Тем не менее, на *GPU* традиционные методы сортировки работают с низкой эффективностью. Отчего появляется потребность в методах сортировки данных, приспособленных для архитектур современных GPU. Одним из видов сортировки, позволяющих получить максимальную производительность на *GPU*, является битоническая сортировка. Битоническая сортировка конвертирует случайную последовательность чисел в битоническую последовательность, которая сначала монотонно возрастает, а затем убывает. Алгоритм состоит из двух частей: сначала неотсортированная последовательность чисел преобразуется в битоническую последовательность; затем последовательность разделяется на ряд меньших последовательностей и обрабатывается до тех пор, пока числа не будут отсортированы. Для реализации битонической сортировки подходят вычислительные шейдеры, позволяющие решать ряд вычислительных задач, которые прежде можно было реализовать только на *CPU*. В статье также приведено описание дополнительных подходов для повышения производительности системы визуализации при отображении облачности.

Ключевые слова: тренажерно-обучающие системы, рендеринг, визуализация облаков, шум перлина, битоническая сортировка, билборды, *GPGPU*, *OpenGL*.

ВВЕДЕНИЕ

В настоящее время, в связи со значительной сложностью технических систем, при эксплуатации возникает риск возникновения нештатных и опасных для жизни ситуаций, появляется потребность в тренажерно-обучающих системах (ТОС). Под *тренажерно-обучающей системой* (ТОС) оператора сложной технической системы (СТС) будем понимать техническое средство для подготовки операторов в едином информационном окружении, отвечающее требованиям методик подготовки, создающее условия для получения знаний, навыков и умений, реализующее модель таких систем и обеспечивающее контроль над действиями обучаемого, а также для исследований. При создании ТОС главной задачей является создание условий, максимально приближенных к реальным [1]. Визуализация погодных явлений - неотъемлемая часть при создании авиационных ТОС. При моделировании внекабинного пространства зачастую необходимо воспроизводить эффекты, возникающие при полетах в облаках. В руководстве ИКАО 9625, описаны требования, предъявляемые к визуализации внекабинного пространства. Должны воспроизводиться эффекты: переменная плотность облаков, многослойная облачность, включая отдельные, рассеянные и разорванные облака, а также сплошная облачность, которая частично или полностью препятствует обзору поверхности земли, постепенный выход из облаков и переход к видимости окружающей обстановки [2]. Так же немаловажным условием является дальность прорисовки облаков. В реальных условиях, видимость может достигать десятков километров. При этом необходимо обеспечить работу подсистемы визуализации в режиме реального времени с частотой смены кадров не менее 25 в секунду.

Подходы к визуализации облаков были описаны, в частности, в работах [3], [4] и [5]. Однако описанные методы не позволяют достичь требований, предъявляемых к визуализации облаков в тренажерно-обучающих системах. В частности, при использовании метода [5] невозможен пролет через облака, методы [3] и [4] не обеспечивают достаточной производительности визуализации окружающей среды при визуализации облаков. Соответственно, исследование новых подходов, позволяющих достичь большей производительности при визуализации облаков, являются актуальными.

1. ШУМ ПЕРЛИНА

Существует несколько подходов к визуализации облаков, каждый из которых имеет свои преимущества и недостатки - использование skybox, процедурное моделирование, моделирование на основе клеточного автомата, использование шума Перлина совместно с билбордами и вокселями. Выбранный для визуализации облачного покрова подход основан на использование шума Перлина и билбордов [6].

Шум Перлина – градиентный шум, состоящий из набора псевдослучайных единичный векторов (направлений градиента), расположенных в определенных точках пространства и интерполированных функцией сглаживания между этими точками [7].

Для генерации шума Перлина в одномерном пространстве, необходимо для каждой точки этого пространства вычислить значение шумовой функции, используя направление градиента (или наклон) в указанной точке. При этом для вычисляемой точки, находящейся между двумя заранее определенными точками пространства (или опорные точки и их значения, которые фактически являются данными, оперируемые алгоритмом), значение вычисляется интерполяцией в соответствии со значениями соседних опорных точек. Данная интерполяция не является линейной, т.к. в противном случае она не будет удовлетворять требованию: производная шумовой функции должна быть непрерывной на опорных точках. Интерполяция достигается при помощи функции смешивания, которая имеет нулевую производную на конечных точках. Первоначально Кен Перлин использовал функцию смешивания «*Hermite*»

$$f(t) = pow(3t,2) - pow(2t,3), \qquad (1)$$

т.к. для шума крайне желательно иметь непрерывной вторую производную шумовой функции, позднее он изменил ее:

$$f(t) = pow(6t,5) - pow(15t,4) + pow(10t,3) \quad (2).$$

Эти две функции очень похожи – уравнение пятой степени также имеет нулевую вторую производную на концах отрезка, т.е. шумовая функция – непрерывная везде, и это делает шум более подходящим для применения в задачах компьютерной графики (*Рис*. 1).

В двумерном пространстве, опорные точки формируют собой регулярную квадратную сетку. На каждой точки сетки формируется псевдослучайный 2*D* вектор (градиент) (*Рис*. 2). Для произвольной точки на поверхности, значение шума вычисляется по четырем ближайшим точкам сетки.

Значение выбранной произвольной точки вычисляется при помощи скалярного произведения векторов ближайших опорных точек, образующих одну ячейку сетки (*Рис*. 3).

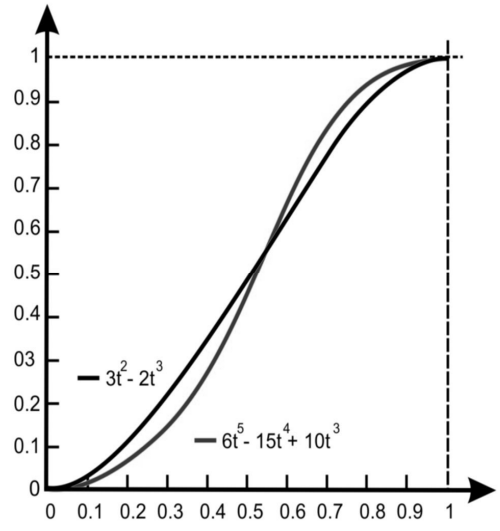


Рис. 1. Графики функций смешивания

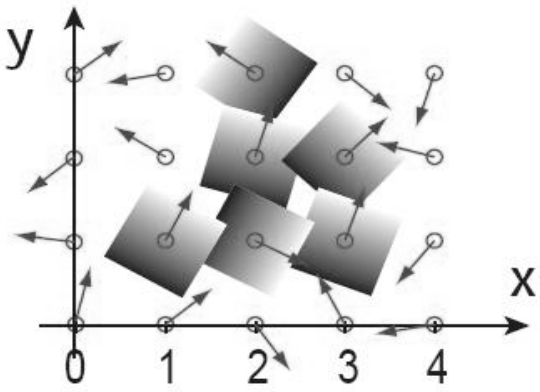


Рис. 2. Формирование псевдослучайных векторов

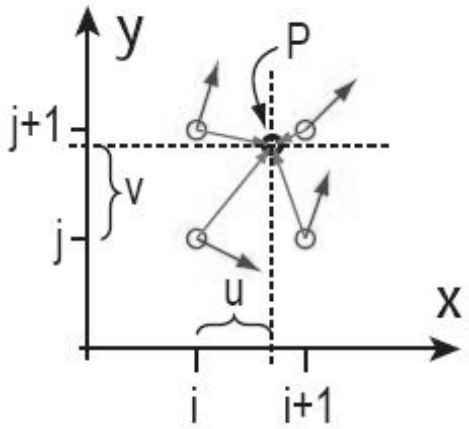


Рис. 3. Вычисление значения выбранной точки

При использовании полинома пятой степени вычисление градиента произвольной точки происходит аналогично. В результате данных манипуляций вычисляется конечное значение шума в выбранной точке пространства. В 3*D* пространстве, градиенты являются трехмерными, и интерполяция осуществляется по трем осям.

Аналогичное обобщение может быть сделано и для произвольного числа измерений.

Определяя искомые градиенты на опорных точках, необходимо иметь в виду, что имея один и тот же набор градиентов, на выходе всегда будет один и тот же шум.

Двумерное представление шума Перлина выглядит так, как показано на *Рис*. 4.

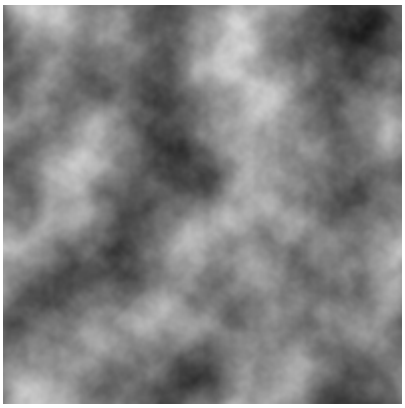


Рис. 4. Двумерный шум Перлина

Каждая генерация шума называется октавой. Октавы отличаются друг от друга масштабом, частотой и амплитудой. С помощью сложения октав шума можно получить шумовые текстуры, на основе которых происходит построение облачных карт (*Рис*. 6). Первая октава используется для задания общей формы облаков. Вторая октава и последующие, генерируются с меньшим масштабом и применяются для придания шуму большей детальности.

Используя трехмерный шум Перлина, путем складывания октав, можно получить значительное количество разнообразных карт облачности генерируемых процедурно, а при использовании четырехмерного шума можно анимировать облака.

2. БИЛБОРДЫ

Билборд - это полигон, все время направленный на наблюдателя. В данном подходе облачная поверхность представляет собой трехмерную решетку, в узлах которой находятся билборды. С помощью шума Перлина, генерируется трехмерная карта облачности. На основе это карты выставляются позиции билбордов. После чего к каждому билборду применяется небольшая текстура, соответствующая части облака.

Так как части облака являются полупрозрачными объектами, для корректного отображения, необходимо производить сортировку билбордов, от дальнего к ближнему [8]. Это диктуют правила смешивания цветов по альфа-каналу (*alpha blending*).

3. СОРТИРОВКА ВЕРШИН ОБЛАКА

Так как облака состоят из большого количества билбордов, производить сортировку на *CPU* крайне ресурсозатратно, поэтому целесообразней перенести сортировку на *GPU*. Тем не менее, к *GPU* неприменимы традиционные методы сортировки. Отчего появляется потребность в методах сортировки данных, приспособленных для архитектур современных *GPU*. Разработанный авторами подход предусматривает перенос процесса сортировки билбордов облака на *GPU*, тем самым разгрузив центральный процессор для других целей, при этом обеспечивается визуализация облачности в реальном масштабе времени (не менее 25 кадров в секунду) и воспроизведение эффектов, возникающих при полетах в облаках.

Главным отличием алгоритмов, приспособленных для архитектур *GPU*, является их параллельный характер и многопроходность. Проанализируем случай, когда необходимо отсортировать одномерный массив с данными. Самый простой вариант заключается в сравнении элементов x_{2i} и x_{2i+1} между собой на каждом проходе, и их перестановки, если в зависимости от критерия упорядочения это необходимо. Такую операцию можно легко распараллелить и, в итоге, на каждом проходе элементы с большими ключами будут сдвигаться к концу массива, а элементы с меньшими значениями ключей - к началу. Однако, для такого подхода (*odd-even transition sort*) потребуется ровно n проходов для сортировки массива из n элементов (например, при сортировке по возрастанию, когда наибольший элемент находится в самом начале массива - за один проход он будет опускаться ровно на одну позицию). Для практических целей данный метод, как правило, не подходит.

Используя сеть компараторов (*comparator networks*) и сортирующие сети, которые приспособлены для архитектуры *GPU*, можно решить данную задачу.

Компаратор представляет собой устройство с n выходами и n входами, которое в зависимости от результатов сравнения выполняет некоторую перестановку элементов (*Рис*. 5).

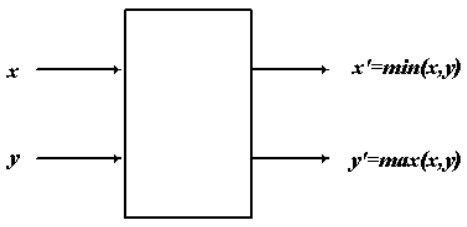


Рис. 5. Компаратор

С помощью компараторов возможно создать сеть, на вход которой приходит последовательность $a_0 \ldots a_{n-1}$, а на выходе

формируется последовательность $b_0 \ldots b_{n-1}$. Число компараторов является размером такой сети, а наибольшее число компараторов, проходимое одним входным элементом - ее глубиной.

Полуфильтр (*half-cleaner*) B_n является одной из простейших сетей компараторов (*Рис*. 7), он определяется таким образом: для всех i от 0 до $n/2-1$ сравнивает между собой элементы x_i и $x_{i+n/2}$.

Таким образом, сеть B_n сравнивает между собой элементы x_i и $x_{i+n/2}$ и упорядочивает их заданным образом (например, по возрастанию). При этом глубина сети равна единице, что значит каждый элемент обрабатывается ровно один раз.

Реализовать компаратор можно как для упорядочивания элементов внутри пары по убыванию, так и по возрастанию. Можно заметить, что сеть B_n требует ровно одного прохода и пригодна для архитектур *GPU*. Произвольный массив заданного размера можно отсортировать путем применения определенной последовательности полуфильтров, т.е. создать сортирующую сеть из отдельных полуфильтров.

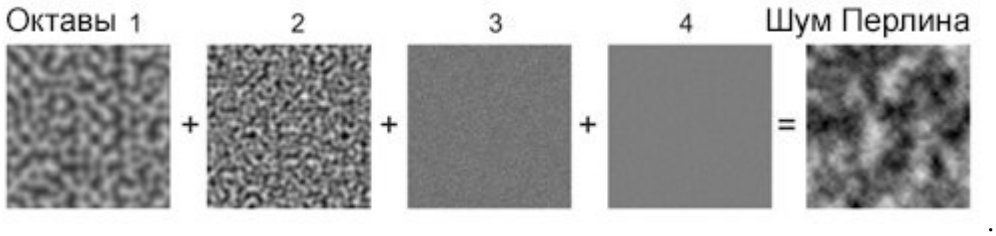


Рис. 6. Сложение октав шума Перлина

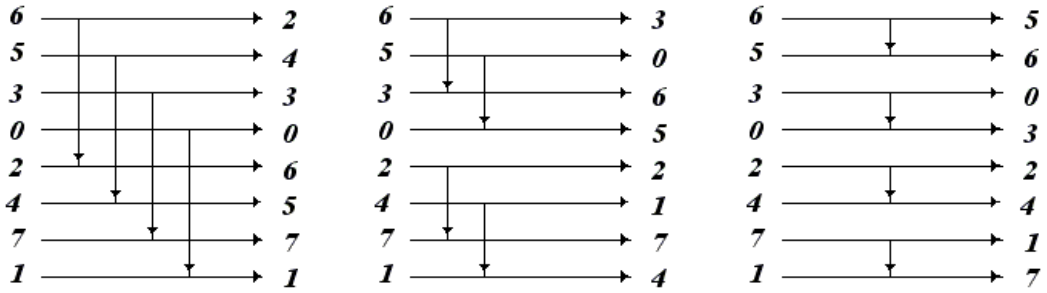


Рис. 7. Сеть компараторов B_n.

Справедливо следующее утверждение: если сеть компараторов сортирует произвольную последовательность из нулей и единиц, то она является сортирующей, т.е. она упорядочивает произвольную последовательность чисел. Следуя этому утверждению, сведя все к наборам из единиц и нулей, можно ощутимо упростить анализ сортирующих сетей. Введем следующее определение: если последовательность $a_0 \ldots a_{n-1}$ сперва убывает, а потом возрастает, она называется битонической. Если все элементы битонической последовательности записать по кругу, то максимальный и минимальный элемент разобьют битоническую последовательность на два монотонных участка. Битоническая последовательность имеет следующее свойство: если к битонической последовательности $a_0 \ldots a_{n-1}$ (длины n) из единиц и нулей применить сеть B_n, то в результате получившаяся последовательность $b_0 \ldots b_{n-1}$ будет иметь следующие свойства:

1. Любой элемент второй (нижней) половины всегда будет больше или равен любого элемента первой (верхней) половины.
2. Хотя бы одна из половин – монотонная.
3. Ее нижняя и верхняя половины также будут битоническими.

Допустим, существует битоническая последовательность $a_0 \ldots a_{n-1}$. Применив к ней сеть B_n, в результате получим 2 части последовательности, каждая из которых будет битонической и все элементы первой половины меньше или равны элементов второй, значит если отсортировать каждую из частей по-отдельности, то в результате получится отсортированная последовательность. Далее если применить сеть $B_{n/2}$ к каждой из половин в результате получатся 4 части, все они будут корректно упорядочены между собой и каждая из них будет битонической. Так же применима сеть $B_{n/4}$ к каждой из 4 частей, в итоге мы получим 8 монотонных частей, каждая из которых по отношению друг к другу будет правильно упорядочена.

Данный процесс можно проделывать до такого момента, пока размеры получившихся частей не станут равны 2 – тогда каждую из этих частей отсортирует сеть B_2. А поскольку между собой эти части правильно упорядочены, то отсортированной будет вся получившееся последовательность. В итоге, получается построить сортирующую сеть для битонических последовательностей.

Подобная сеть также называется битоническим слиянием (*bitonic merge*). На следующем рисунке представлена ее структура (*Рис*. 8):

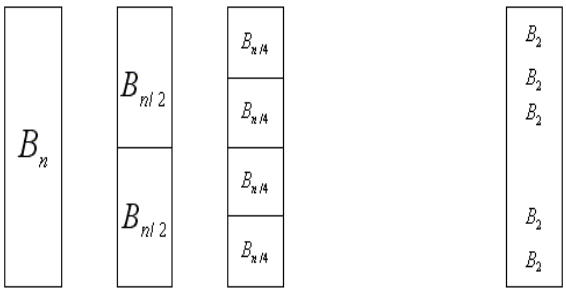


Рис. 8. Битоническое слияние

Так же можно отсортировать произвольную последовательность длины n построив сеть используя сети B_n и M_n.

Предположим существует последовательность a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7. Разобьем ее на пары и применим к каждой сеть B_2, но с чередующимся порядком сортировки. В результате получится последовательность (*Рис*. 9) со следующим порядком (стрелки показывают убывание/возрастание).

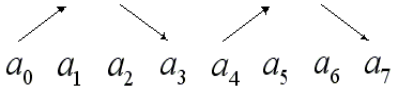


Рис. 9. Результат применения B_2 с чередующимся порядком упорядочивания

Можно отметить, что первые 4 элемента (a_0, a_1, a_2, a_3) и вторые 4 элемента (a_4, a_5, a_6, a_7) последовательности являются битоническими. Следовательно, если применить операцию битонического слияния M_4, сортирующей по возрастанию к первой, а операцию слияния M_4, сортирующей по убыванию ко второй, то в результате получится последовательность, которая состоит из двух монотонных участков и является битонической (*Рис*. 10).

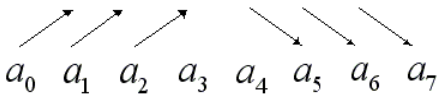


Рис. 10 Результат применения битонического слияния M_4

Тогда мы получим полностью отсортированную последовательность применив ко всей последовательности операцию M_8.Такая последовательность операций и будет являться битонической сортировкой. Битоническая сортировка требует $\log_2 n*(\log_2 n+1)/2$ проходов.

Для реализации битонической сортировки подходят вычислительные шейдеры. Впервые вычислительные шейдеры (compute shaders) появились в графическом *API DirectX* 11, а затем в *OpenGL* версии 4.3. Вычислительный шейдер представляет собой программу, в ходе которой можно выполнять произвольные вычисления. Он дает возможность задействовать *GPU* и присущий ему параллелизм для решения любых вычислительных задач, которые прежде можно было реализовать только на *CPU*. Вычислительные шейдеры можно использовать для задач, не связанных непосредственно с отображением графики, таких как симуляция физических процессов. В настоящее время уже существуют библиотеки, обеспечивающие возможность выполнения универсальных вычислений на *GPU*, такие как *OpenCL* и *CUDA*, однако они никак не связаны с *OpenGL*. Вычислительные шейдеры, напротив, интегрированы непосредственно в *OpenGL* и, как следствие, лучше подходят для организации вычислений, имеющих отношение к отображению графики. Вычислительный шейдер не является традиционным этапом графического конвейера прорисовки, как фрагментный или вершинный шейдер. Вычислительные шейдеры не имеют входных и выходных переменных, определяемых пользователем. Все исходные данные они могут получать только прямым обращением к памяти, используя функции доступа к изображениям или посредством буферных объектов. Результаты вычислений должны сохраняться в те же самые или в другие объекты.

Перенос сортировки на вычислительные мощности GPU повышают производительность системы, но для достижения лучших результатов необходимо применение дополнительных оптимизаций. Например, одной из оптимизаций является отсечение геометрии, невидимой на экране. Находясь внутри облака, нам не обязательно сортировать все билборды вокруг себя, достаточно сортировать лишь попадающие в усеченную пирамиду видимости (*frustum*). Также применяются различные подходы разделения объектного пространства на части [9]. Восьмеричное дерево (*octree*) является одним из таких алгоритмов. Данный алгоритм выполняет деление объектного пространства на восемь подпространств.

ЗАКЛЮЧЕНИЕ

В работе описан разработанный подход к визуализации облаков в авиационных тренажерах. Предлагаемый подход основан на использовании шума Перлина и билбордов, позволяет процедурно генерировать облачные карты, дающие значительное количество вариаций облаков разных типов. Поскольку необходимо постоянно сортировать билборды для корректного отображения полупрозрачных объектов, перенос данного действия на вычислительные мощности видеокарты и добавление такой оптимизации как *culling*

увеличивает производительность подсистемы визуализации. Полученные результаты можно применить при проектировании авиационных тренажеров для визуализации облачного неба, отвечающего критериям, описанным в руководстве ИКАО 9625.

Работа выполнена при поддержке РФФИ, грант № 17-07-00169 А.

ЛИТЕРАТУРА

[1] Гиацинтов А.М., Мамросенко К.А. Управление тренажерно-обучающими системами при помощи программируемых сценариев. Вестник Компьютерных и Информационных Технологий. 2016. № 5. С. 52–56.

[2] Андреев А. О., Дукальская М. В., Головина Е. Г. Облака: происхождение, классификация, распознавание. СПб: РГГМУ, 2007. 228 С.

[3] Prashant Goswami, Fabrice Neyret. Real-time landscape-size convective clouds simulation and rendering. Inria. 2016. P. 17.

[4] Roland Hufnagel, Martin Held, Florian Schroder. Large-Scale, Realistic Cloud Visualization Based on Weather Forecast Data. 2007. P. 54–59.

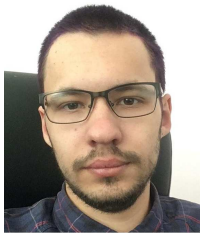
[5] Ksenia Mukhina, Alexey Bezgodov. The Method for Real-time Cloud Rendering. 2015. P. 697–704.

[6] Гиацинтов А.М., Родителев А.В., Агафонов Н.А. Методы визуализации погодных явлений в имитационных системах. Вестник Кибернетики. 2018. № 1. С. 61–65.

[7] Ken Perlin. An Image Synthesizer // In Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH, 1985. P. 287–296.

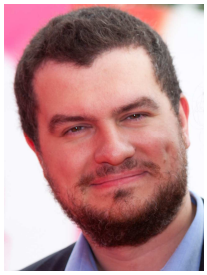
[8] Dobashi Yoshinori et al. A Simple, Efficient Method for Realistic Animation of Clouds. SIGGRAPH, 2000. P. 19–28.

[9] David S. Ebert et al. Texturing & Modeling: A Procedural Approach. Morgan Kaufmann, 2003. Issue. 3. 687 P.



Николай Александрович Агафонов, программист Центра визуализации и спутниковых информационных технологий ФГУ ФНЦ НИИСИ РАН, Wowerq@gmail.com
117218, Москва, Нахимовский просп., 36, к.1.

Александр Михайлович Гиацинтов, кандидат технических наук, старший научный сотрудник Центра визуализации и спутниковых информационных технологий ФГУ ФНЦ НИИСИ РАН, algts@inbox.ru
117218, Москва, Нахимовский просп., 36, к.1.



Кирилл Анатольевич Мамросенко, кандидат технических наук, руководитель Центра визуализации и спутниковых информационных технологий ФГУ ФНЦ НИИСИ РАН, mamrosenko_k@niisi.ras.ru
117218, Москва, Нахимовский просп., 36, к.1.

Статья поступила 12.11.2018.

Methods of procedural generation and visualization of clouds in training simulation systems

N.A. Agafonov, A.M. Giatsintov, K.A. Mamrosenko

Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow

Abstract: This article describes a method of procedural cloud generation with the use of Perlin noise function and billboards. Perlin noise can be used to generate various effects with natural qualities, such as clouds, landscapes, and patterned textures like marble. Process of creating noise textures that are used during cloudscape generation is depicted. Clouds are made of a large number of billboards and sorting is required in order to correctly draw a number of semi-transparent surfaces. Sorting large number of billboards on CPU is a resource-heavy operation, therefore it is desirable to move sorting process to GPU. But traditional methods of sorting are not performing well on GPU architecture, therefore, there is a need for special algorithms that can use the processing powers of GPU effectively. One type of sorting that is suitable for GPU architecture is called bitonic sorting. Bitonic sort is a comparison-based sorting algorithm that can be run in parallel. It focuses on converting a random sequence of numbers into a bitonic sequence, one that monotonically increases, then decreases. The algorithm consists of two parts. First, the unsorted sequence is built into a bitonic sequence; then, the series is split multiple times into smaller sequences until the input is in sorted order. In order to implement bitonic sorting compute shaders can be used that allow solving a number of computing tasks, which could only be done on CPU before, on GPU. A number of additional approaches to increase the performance of visualization system during cloud rendering are presented.

Key words: training simulation systems, rendering, cloud visualization, Perlin noise, bitonic sort, billboards, GPGPU, OpenGL.

REFERENCES

[1] Giacintov A.M., Mamrosenko K.A. Upravlenie trenazherno-obuchajushhimi sistemami pri pomoshhi programmiruemyh scenariev. Vestnik Komp'juternyh i Informacionnyh Tehnologij. 2016. № 5. S. 52–56.

[2] Andreev A. O., Dukal'skaja M. V., Golovina E. G. Oblaka: proishozhdenie, klassifikacija, raspoznavanie. SPb: RGGMU, 2007. 228 S.

[3] Prashant Goswami, Fabrice Neyret. Real-time landscape-size convective clouds simulation and rendering. Inria. 2016. P. 17.

[4] Roland Hufnagel, Martin Held, Florian Schroder. Large-Scale, Realistic Cloud Visualization Based on Weather Forecast Data. 2007. P. 54–59.

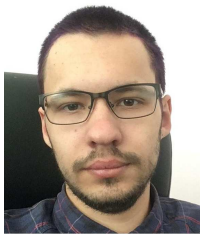
[5] Ksenia Mukhina, Alexey Bezgodov. The Method for Real-time Cloud Rendering. 2015. P. 697–704.

[6] Giacintov A.M., Roditelev A.V., Agafonov N.A. Metody vizualizacii pogodnyh javlenij v imitacionnyh sistemah. Vestnik Kibernetiki. 2018. № 1. S. 61–65.

[7] Ken Perlin. An Image Synthesizer // In Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH, 1985. P. 287–296.

[8] Dobashi Yoshinori et al. A Simple, Efficient Method for Realistic Animation of Clouds. SIGGRAPH, 2000. P. 19–28.

[9] David S. Ebert et al. Texturing & Modeling: A Procedural Approach. Morgan Kaufmann, 2003. Issue. 3. 687 P.

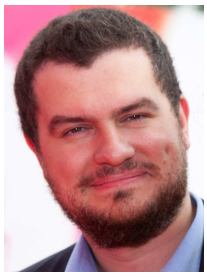


Nikolay Agafonov, Programmer
CVSIT SRISA RAS
Wowerq@gmail.com

36/1, Nahimovskij prosp.,
Moscow, 117218, Russian
Federation

Alexander Giatsintov, Ph.D. (Engineering), Senior researcher CVSIT SRISA RAS
algts@inbox.ru

36/1, Nahimovskij prosp.,
Moscow, 117218, Russian
Federation



Kirill Mamrosenko, Ph.D. (Engineering), Head of the CVSIT SRISA RAS
mamrosenko_k@niisi.ras.ru

36/1, Nahimovskij prosp.,
Moscow, 117218, Russian
Federation

The paper has been received on 12.11.2018.