Сверхбольшие текстуры для высоко реалистичной визуализации виртуальных ландшафтов[1]

П.Ю. Тимохин, М.В. Михайлюк

Аннотация. В статье предлагается новый метод быстрого определения видимых текстурных тайлов, позволяющий выполнять высоко реалистичную визуализацию виртуальных ландшафтов со сверхбольшими текстурами в режиме реального времени. Описываются разработанные параллельные алгоритмы, реализующие предлагаемый метод и использующие современные возможности по организации параллельных вычислений на графическом процессоре. Приведена технология динамической подкачки тайлов сверхбольшой текстуры в реальном времени и алгоритм текстурирования ландшафта.

Ключевые слова: высоко реалистичная визуализация, текстура, виртуальный ландшафт, реальное время, GPU.

Введение

В настоящее время одним из активно развивающихся направлений компьютерной графики является высокореалистичная визуализация виртуальных текстурированных ландшафтов в масштабе реального времени. Такая визуализация особенно востребована в видеотренажерных комплексах космического, авиационного и наземного назначения [1, 2]. В таких комплексах в качестве текстур часто используют данные дистанционной фотосъемки реального ландшафта.

Одним из эффективных методов визуализации текстурированных ландшафтов является мипмэппинг, при котором наряду с исходными текстурами используются текстуры с меньшими уровнями детализации [3]. В этом случае при синтезе изображения для каждого пиксела подбираются текстуры тех уровней, детализация которых наиболее близка к разрешению экрана. Реализация такой технологии требует использования мип-пирамид, размеры которых для сверхбольших текстур превышают аппаратно допустимые значения современных графических ускорителей, что не позволяет загружать их в видеопамять и аппаратно обрабатывать.

Распространенным подходом к решению данной проблемы является разбиение уровней мип-пирамиды с помощью регулярной сетки на небольшие участки одинакового размера (тайлы) и подкачка в видеопамять только тех тайлов, которые нужны для визуализации текущего кадра. В связи с этим возникает задача разработки эффективных алгоритмов отбора, подкачки и визуализации видимых тайлов. В данной работе предлагается новый метод решения этой задачи, позволяющий визуализировать в реальном времени текстуры сверхбольших размеров (порядка $10^6 \times 10^6$ текселов). Предлагаемые алгоритмы используют быстрые параллельные вычисления на современных многоядерных графических процессорах (GPU) и адаптированы для эффективного применения геометрических шейдеров и технологии возврата примитивов (transform feedback).

[1] Работа выполняется при поддержке РФФИ, грант № 12-07-00256.

1. Предшествующие работы

Можно выделить два ключевых подхода к решению поставленной задачи.

При первом подходе отбор, подкачка и визуализация тайлов выполняются исходя из проверки видимости участков геометрии ландшафта, которые они покрывают. Достаточно распространенными являются методы, использующие проверку по полигонам [4], по ограничивающим боксам [5], проверку иерархического дерева геометрии ландшафта [6]. При относительной простоте наложения тайлов (для ландшафтов с регулярной геометрией) в данном подходе в геометрическую модель ландшафта нужно вносить дополнительные вершины (для ландшафтов с динамической оптимизацией сетки вершин или нерегулярной структурой) и текстурные координаты, обеспечивающие привязку тайлов к геометрии ландшафта, что неэффективно расходует видеопамять и увеличивает время визуализации такой модели, ограничивая применение данного подхода в режиме реального времени [7].

При втором подходе работа с мип-пирамидой ведется так, как будто она полностью присутствует в видеопамяти, при этом физически в видеопамять подкачивается лишь небольшая ее часть (по аналогии с виртуальной памятью) [8]. В этом случае для каждой вершины модели ландшафта задаются только глобальные текстурные координаты в полной текстуре, которые при визуализации пересчитываются в локальные текстурные координаты тайлов, загруженных в видеопамять. В отличие от первого подхода, где тайлы являются самодостаточными текстурами, здесь они выступают в роли страниц, которыми мип-пирамида подкачивается в единый буфер в видеопамяти, что позволяет сделать текстурирование ландшафта независимым от моделирования его геометрии. Для обеспечения высокой реалистичности визуализации ландшафтов, имеющих произвольную геометрию, с помощью данного подхода необходимо точное определение видимых на экране тайлов-страниц. Такую точность позволяет обеспечить попиксельный анализ экрана [9, 10], однако при этом возникает задача быстрого и эффективного выполнения большого количества вычислений в режиме реального времени. Многие существующие методы используют подход [11], при котором такие вычисления распределяются между графическим (GPU) и центральным процессором (CPU), однако временные задержки, связанные с пересылкой промежуточных результатов от GPU к CPU и их CPU-обработкой, ограничивают использование данных методов для визуализации в реальном времени.

Подход, предлагаемый в данной работе, позволяет быстро выделить список видимых тайлов-страниц полностью на стороне GPU, используя его мощный потенциал параллельных вычислений и сводя к минимуму объем данных, передаваемых в оперативную память (CPU отводится лишь функция управления подкачкой полученного списка тайлов), что в результате позволяет обойти описанные выше ограничения.

2. Предлагаемый подход

Пусть задана исходная текстура T_0 ландшафта размером $w = 2^m$ на $h = 2^n$ текселов. Будем называть ее текстурой нулевого уровня. По этой текстуре создаются текстуры 1-го, 2-го, …, L-го уровней детализации. Текстура T_s s-го уровня имеет размеры $2^{m-s} \times 2^{n-s}$ текселов и получается из текстуры T_{s-1}, например, вычислением среднего значения цветов четырех соседних текселов. Разобьем каждую текстуру s-го уровня на $2^{m-s-k} \times 2^{n-s-k}$ тайлов размером $2^k \times 2^k$ текселов (Рис.1) и обозначим массив этих тайлов через M_s. Ввиду того, что видеопамять имеет ограниченные размеры, выделим в ней кусок, в котором постоянно будут находиться L' текстур $T_{L-L'+1},\ldots,T_L$ (группа G_1 текстур), и кусок памяти, в который будут динамически (по мере необходимости) подгружаться тайлы текстур остальных уровней (группа G_2 тайлов текстур). Нашей задачей является определение множества тайлов каждого уровня, которые необходимо подкачать для визуализации текущего кадра. Заметим, что оставшаяся часть видеопамяти будет использоваться для текущих работ графического процессора.

Решение поставленной задачи предлагается выполнить в 3 этапа. На первом этапе для каж-

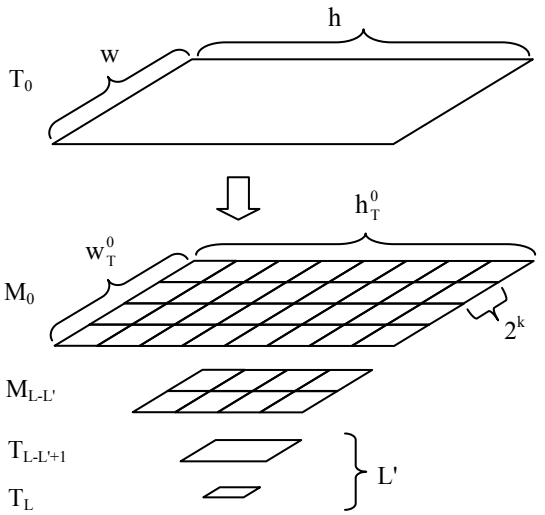

Рис. 1. Подготовка исходной текстуры

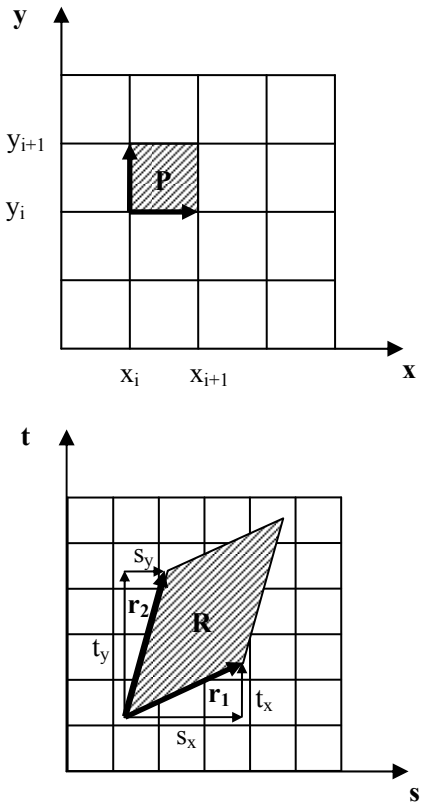

Рис. 2. Приближенное представление пиксела в экранной системе координат (квадрат P, вверху) и его проекции в пространстве текстуры T_0 (параллелограмм R, внизу)

дого пиксела текущего кадра определяется соответствующий ему тайл нужного уровня. Множество этих тайлов является сильно избыточным, так как при визуализации каждый тайл может участвовать в закраске сразу группы пикселов экрана. Для устранения этой избыточности выполняются второй и третий этапы. На втором этапе происходит удаление повторяющихся элементов из множества, а на третьем этапе оставшиеся элементы объединяются в отдельное множество, которое и является искомым. Рассмотрим все этапы более подробно.

2.1. Определение множества тайлов для пикселов текущего кадра

При создании модели ландшафта для каждой его геометрической вершины указываются текстурные координаты этой вершины в текстуре T_0. Используя эти координаты, при визуализации текущего кадра для каждого пиксела видеокарта автоматически вычисляет текстурные координаты (s, t) на T_0, соответствующие этому пикселу. Поставим в соответствие каждому пикселу тройку чисел (q, i, j), где i, j – индексы тайла в массиве M_0, соответствующие текстурным координатам (s, t). Число q равно наименьшему номеру уровня детализации текстуры T_0, которую необходимо использовать для закраски пиксела. Определение номера q выполняется на основе вычисления количества текселов, охватываемого сторонами проекции пиксела в пространстве текстуры T_0. На практике такая проекция эффективно приближается параллелограммом (Рис. 2), длины сторон которого вычисляются с помощью частных производных текстурных координат (s, t) [3]. Отметим, что при этом для корректного определения q на малых углах наблюдения текстуры T_0 необходимо также использование коэффициента n_A, компенсирующего анизотропное изменение сторон параллелограмма [12].

Определим новую текстуру U_1 размера, совпадающего с размером кадра, в которой на каждый цветовой канал r, g, b отводится 32 бита. Первая шейдерная программа будет рендерить изображение не в буфер кадра, а в текстуру U_1, записывая величины (q, i, j) для каждого пиксела в ее цветовые каналы r, g, b. Для каждого пиксела получаем следующий алгоритм.

Алгоритм вычисления тройки (q, i, j)

1. Вычисляем индексы i, j тайла из массива M_0:
$$i = \lfloor h_T^0 \cdot t \rfloor,\ j = \lfloor w_T^0 \cdot s \rfloor,$$
где h_T^0, w_T^0 – число строк и столбцов тайлов в массиве M_0.
2. Вычисляем наименьший необходимый уровень q детализации текстуры T_0:
   2.1. находим квадраты максимальной длины d_{max} (в текселах) и минимальной длины d_{min} сторон параллелограмма R (Рис. 2):
$$d_{\max}^2 = \max\left(|\mathbf{r}_1|^2, |\mathbf{r}_2|^2\right),$$
$$d_{\min}^2 = \min\left(|\mathbf{r}_1|^2, |\mathbf{r}_2|^2\right),$$
где $|\mathbf{r}_1|^2 = s_x^2 + t_x^2$, $|\mathbf{r}_2|^2 = s_y^2 + t_y^2$, и
$$s_x = w\frac{\partial s}{\partial x},\ s_y = w\frac{\partial s}{\partial y},\ t_x = h\frac{\partial t}{\partial x},\ t_y = h\frac{\partial t}{\partial y};$$
   2.2. вычисляем номер ℓ наименьшего необходимого уровня детализации текстуры T_0:
$$\ell = \lfloor \log_2(d_{\max}/n_A) \rfloor =$$
$$= \lfloor 0.5 \cdot \log_2(d_{\max}^2) - \log_2(n_A) \rfloor,$$
где $n_A = \min\left(\lceil d_{\max}/d_{\min} \rceil, N_A\right) =$
$= \min\left(\left\lceil \sqrt{d_{\max}^2/d_{\min}^2} \right\rceil, N_A\right)$, а N_A - максимальное аппаратно поддерживаемое количество выборок при анизотропной фильтрации текстуры;
   2.3. отсекая номер ℓ по границам диапазона [0, L-L'+1], получаем q.
3. Записываем вычисленные величины q, i, j в каналы r, g, b соответствующего тексела текстуры U_1.

Для корректной обработки случаев, когда s = 1 или t = 1, в п. 1. алгоритма вычисленные индексы i, j необходимо привести к виду $i = \min\left(i, h_T^0 - 1\right)$ и $j = \min\left(j, w_T^0 - 1\right)$.

2.2. Удаление повторяющихся элементов

Как видно из приведенного выше алгоритма, текстура U_1 обладает большой избыточностью, так как во многие ее текселы записываются повторяющиеся тройки (q, i, j). Для удаления избыточных троек предлагается следующий подход. Поставим в соответствие каждой тройке (q, i, j) вершину V, имеющую координаты (j, i) и цвет q. Определим новую одноканальную текстуру U_2 глубины 32 бита и размера $w_T^0 \times h_T^0$ (совпадающего с размером массива M_0 тайлов). Обнулим значения текселов этой текстуры перед началом работы. Вторая шейдерная программа будет рендерить вершину V в текстуру U_2, добавляя в цветовой канал с помощью битовой операции «ИЛИ» (OR) значение $1 << q$. В результате такого рендеринга вершины с одинаковыми координатами и цветом попадут в один и тот же тексел, т.е. в текстуре U_2 будут записаны в одном экземпляре. При этом в цветовом канале позиции единичных битов будут соответствовать номерам уровней детализации исходной текстуры.

Прямая реализация данной идеи, при которой рендерятся все вершины с помощью вершинного (VS) и фрагментного шейдера (FS), активно расходует вычислительный ресурс GPU и ресурс канала видеопамять-GPU, что ограничивает общую скорость визуализации ландшафта. Существенно уменьшить количество вершин, подлежащих рендерингу, можно за счет использования геометрического шейдера (GS). В графическом конвейере геометрический шейдер позволяет передать на обработку лишь одну вершину, но обработать целый блок, связанный с этой вершиной. Поэтому разобьем текстуру U_1 на блоки размера $2^p \times 2^p$ текселов (Рис. 3) и выберем в каждом блоке крайний левый нижний тексел. Номера этих текселов будут иметь вид $\left(i_B 2^p, j_B 2^p\right)$, где i_B, j_B – номера строки и столбца блока текселов в текстуре U_1. Для этих текселов создадим массив вершин M_{V_B}, вершины V_B которого и будут отосланы в геометрический шейдер. Получив такую вершину, шейдер сгенерирует все такие вершины соответствующего ей блока, для которых $q \le L - L'$. Таким образом, в графическом конвейере выполняется следующий параллельный алгоритм.

Алгоритм удаления повторяющихся троек (q, i, j)

1. Передаем из вершинного шейдера в геометрический шейдер вершину V_B с координатами $\left(i_B 2^p, j_B 2^p\right)$.

2. В геометрическом шейдере выполняем
   Цикл по всем (i_t, j_t) текселам блока (i_B, j_B)
   Если $q(i_t, j_t) \le L - L'$, то
   Генерируем вершину $V(i_t, j_t)$ с координатами $x = -1 + 2j(i_t, j_t)/w_T^0$ и $y = -1 + 2i(i_t, j_t)/h_T^0$ (в системе координат нормализованного объема видимости) и цветом $C = 1 << q(i_t, j_t)$.
   Конец цикла.
3. С помощью фрагментного шейдера для каждой сгенерированной вершины $V(i_t, j_t)$ добавляем цвет C в цветовой канал texсела $(i(i_t, j_t), j(i_t, j_t))$ текстуры U_2.

Можно повысить эффективность работы этого алгоритма, если запоминать texсел, для которого была сгенерирована вершина. В этом случае каждый следующий texсел сравнивается с запомненным, и новая вершина генерируется только в случае их несовпадения.

2.3. Объединение неповторяющихся элементов в отдельное множество

Полученная на предыдущем этапе текстура U_2 содержит два типа данных: 1) данные о неповторяющихся тройках (q, i, j) (соответствующие единичным битам в texселах), и 2) данные об отсутствующих тройках (q, i, j) (в texселах соответствующие биты равны нулю), которые занимают большую часть текстуры U_2. Для дальнейшей работы требуются только неповторяющиеся тройки (q, i, j), поэтому их необходимо объединить в отдельное множество. Для этого поставим в соответствие каждому единичному биту, находящемуся в q-ой позиции в (i, j)-ом texселе текстуры U_2 вершину V', имеющую координаты x = q, y = i, z = j. Определим новый одномерный массив U_3 длины, равной количеству пикселов в кадре, в каждом элементе которого будут храниться координаты x, y, z вершины (3×4 байта). С помощью геометрического шейдера сгенерируем все N вершин V' и запишем их координаты в массив U_3. Для этого, как и на предыдущем этапе, разобьем текстуру U_2 на квадратные блоки размера $2^p \times 2^p$ texселов (рис.4), создадим соответствующий им массив вершин M_{V_B},

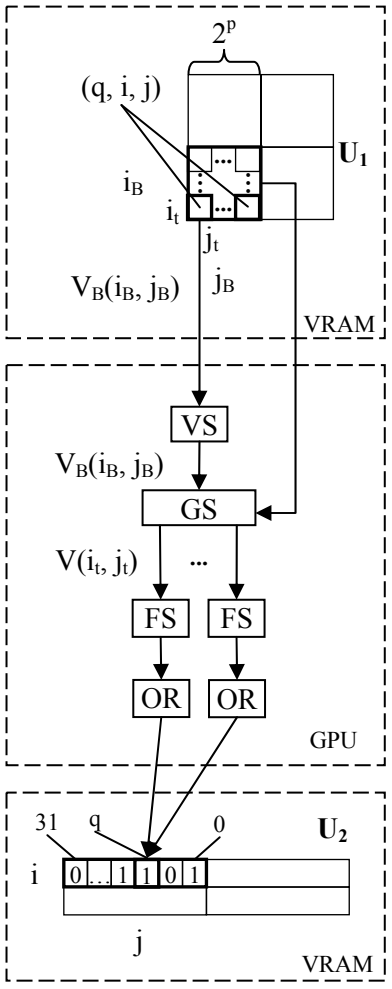

Рис. 3. Удаление повторяющихся троек (q, i, j)

вершины V_B которого будут отосланы в геометрический шейдер. Для каждой такой вершины геометрический шейдер будет генерировать все вершины блока, для которых существуют единичные биты в texселах. Единичные биты выделяются с помощью битовых масок $B_q = 1 << q$, $q = 0, \ldots, L - L'$. Таким образом, на графическом процессоре выполняется следующий параллельный алгоритм.

Алгоритм объединения неповторяющихся троек (q, i , j)

1. Передаем из вершинного шейдера в геометрический шейдер вершину V_B с координатами $(i_B 2^p, j_B 2^p)$.
2. В геометрическом шейдере выполняем
   Цикл по всем (i, j) texселам блока (i_B, j_B)

Считываем содержимое $C_{i,j}$ тексела.

Цикл по позициям q битов в тексе ле от 0 до $L - L'$

Если $C_{i,j}\ \&\ B_q \neq 0$, то

Генерируем вершину $V'(q,i,j)$

с координатами x = q, y = i, z = j.

Конец цикла.

3. Записываем координаты x, y, z каждой сгенерированной вершины $V'(q,i,j)$ в следующий свободный элемент массива U_3.

Запись координат сгенерированных вершин в последнем пункте алгоритма осуществляется с помощью технологии возврата примитивов (transform feedback). С помощью данной технологии вершины, сгенерированные геометрическим шейдером, перехватываются из графического конвейера и в произвольной последовательности добавляются в специальный буфер в видеопамяти (Рис. 4). Выгрузив x,y,z-координаты всех N записанных вершин из этого буфера в оперативную память, мы получаем набор неповторяющихся троек (q, i, j), по которым можем выполнить подкачку соответствующих тайлов.

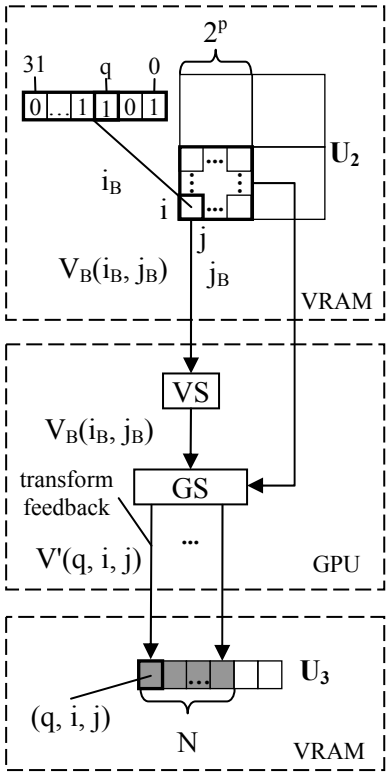

Рис. 4. Объединение неповторяющихся троек (q, i, j)

3. Подкачка тайлов

При подкачке тайлы исходно загружаются с жесткого диска. Поскольку эта операция является достаточно медленной (что препятствует режиму реального времени), то для уменьшения числа обращений к жесткому диску предлагается сохранять (кэшировать) часто запрашиваемые тайлы в видео- и оперативной памяти. Для этого в видеопамяти создается массив структур J_1 для хранения N_1 тайлов размера $2^k \times 2^k$ текселов и их номеров q уровней детализации, а в оперативной памяти – аналогичный массив структур J_2 для хранения N_2 тайлов того же размера с их уровнями детализации, а также наборы H_1 и H_2 таблиц, хранящих информацию о расположении тайлов в массивах J_1 и J_2. Отметим, что в данной работе размер N_1 подбирается экспериментально таким, чтобы он был не меньше максимально возможного количества видимых тайлов. Кроме того, вводится очередь Q_1 троек (q, i', j'), где $i' = \lfloor i/2^q \rfloor$, $j' = \lfloor j/2^q \rfloor$, для подкачки тайлов из массива J_2 в массив J_1 и очередь Q_2 для подкачки тайлов с жесткого диска в массив J_2. Если тайл с номером (i', j') уровня q записан в k-ом элементе массива J_1, то в ячейку с номером (i', j') таблицы $H_{1,q}$ записывается этот индекс k. Если этот тайл отсутствует в массиве J_1, но поставлен на закачку в очередь Q_1, то в рассматриваемую ячейку записывается -1, если же он отсутствует и не поставлен на закачку, то записывается -2. Аналогично устроен и набор таблиц $H_{2,q}$, относящийся к массиву J_2.

Опишем теперь алгоритмы подкачки тайлов, использующие описанные структуры данных. Алгоритм A_1 подкачки в видеопамять получает очередную тройку (q, i, j), полученную на этапе определения видимых тайлов, преобразует ее в тройку (q, i', j') и проверяет значения k_1 и k_2, записанные в ячейках (i', j') таблиц $H_{1,q}$ и $H_{2,q}$. В зависимости от этих значений выполняется следующее. Если $k_1 = -2$ и $k_2 = -2$, то это означает, что тайл отсутствует в J_1, J_2 и в очередях Q_1 и Q_2, поэтому данная тройка добавляется

в очередь Q_2, и в соответствующую ячейку таблицы $H_{2,q}$ записывается $k_2 = -1$. Если $k_1 = -2$ и $k_2 \geq 0$, тайл уже присутствует в оперативной памяти и необходимо поставить его в очередь Q_1 для закачки в видеопамять, после чего изменить значение k_1 на -1. Если $k_1 = -2$ и $k_2 = -1$, это значит, что тайл уже стоит в очереди Q_2 и его подкачивать не надо. При $k_1 = -1$ соответствующий тайл с номером (q, i', j') уже поставлен на закачку в очередь Q_1, а при $k_1 \geq 0$ он уже содержится в массиве J_1. Поэтому в этих случаях (независимо от k_2) его подкачка также не требуется.

После обработки всех троек начинает работу алгоритм A_2 подкачки тайлов из массива J_2 в массив J_1 в соответствии с очередью Q_1. После подкачки каждого тайла очереди производится корректировка значения k_1 в таблице $H_{1,q}$ и запись номера q в массив J_1. При этом выделяется некоторый промежуток времени, по истечении которого подкачка прекращается, а обработка оставшихся в очереди тайлов переносится в следующий кадр визуализации. В этом случае для каждого оставшегося тайла значение k_1 в таблице $H_{1,q}$ заменяется значением $k_1 \geq 0$ из соответствующей ячейки таблицы $H_{1,q'}$ с ближайшим номером $q' > q$. Если такого значения не оказывается во всех таблицах $H_{1,q'}$, то значение k_1 остается неизменным (равным -1 или -2), и для визуализации используются данные из группы G_1 постоянно хранимых текстур. Завершая рассмотрение алгоритма A_2 отметим, что после заполнения массива J_1 подкачка следующего необходимого тайла выполняется на место того, который дольше всех не использовался (стратегия LRU, Least Recently Used). При этом для каждого подкачиваемого в видеопамять тайла фиксируется время t его записи в k-ый элемент массива J_1, и пара (k, t) добавляется в очередь с приоритетом. При необходимости замещения тайла в J_1 из данной очереди извлекается пара (k, t) с наименьшим значением t, и тайл записывается в k-ый элемент массива J_1. Отметим, что для тайлов, видимых в текущем кадре и уже присутствующих в массиве J_1, выполняется обновление времени t в соответствующей паре (k, t), что предотвращает их преждевременную перезапись. Аналогично организовано замещение тайлов и в массиве J_2.

Параллельно с описанными алгоритмами в другом потоке выполняется алгоритм A_3 подкачки тайлов в оперативную память в соответствии с очередью Q_2. После подкачки очередного тайла он корректирует значение k_2 в соответствующей ячейке таблицы $H_{2,q}$, ставит этот тайл в очередь Q_1 для подкачки в видеопамять и записывает $k_1 = -1$ в соответствующую ячейку таблицы $H_{1,q}$. Здесь также для закачки выделяется некоторый временной промежуток, и подкачка оставшихся тайлов продолжается в следующем кадре визуализации.

4. Текстурирование ландшафта

Имея в видеопамяти массив J_1 с подкачанными тайлами, набор H_1 заполненных таблиц, а также группу G_1 постоянно хранимых текстур, вычислим в текущем кадре для каждого пиксела изображения ландшафта цвет C_T его текстурной закраски. Все такие цвета будут вычисляться параллельно с помощью фрагментного шейдера и записываться в буфер кадра. В процессе вычисления такая шейдерная программа для каждого пиксела определяет два ближайших уровня детализации исходной текстуры, необходимых для его закраски, осуществляет выборку цвета из каждой текстуры такого уровня детализации и путем линейной интерполяции этих цветов вычисляет искомый цвет C_T. При этом, если определенный в шейдере ближайший уровень детализации относится к группе G_1 постоянно хранимых текстур, то выборка цвета выполняется из соответствующей текстуры по координатам (s, t) пиксела (раздел 2.1), в противном случае выборка цвета выполняется из тайла, записанного в массив J_1, для чего координаты (s, t) пересчитываются в текстурные координаты (s', t') этого тайла. Таким образом, получаем следующий параллельный алгоритм.

Алгоритм вычисления цвета C_T пиксела

1. Определяем наименьший необходимый уровень q детализации текстуры T_0, а также коэффициент f интерполяции между уровнями q и q + 1 (п. 2 алгоритма из раздела 2.1):

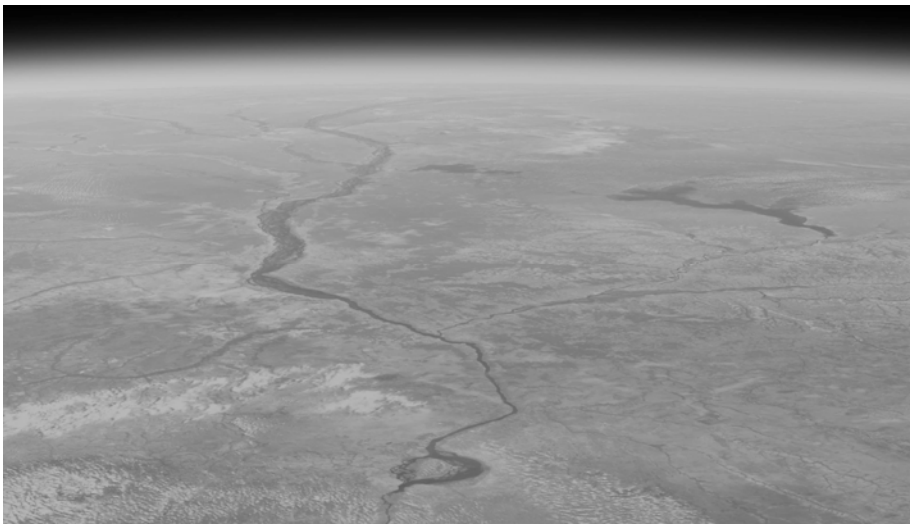

Рис. 5. Пример кадра визуализации модели земной поверхности с высоты 300 км с 4-х кратным увеличением

$q = \lfloor e \rfloor,\ q \in [0, \text{L-L}'+1],\ f = \{e\},$

где $e = 0.5 \cdot \log_2\left(d_{\max}^2\right) - \log_2\left(n_A\right)$.

2. Если $q \geq L - L' + 1$, то

   $C_T = G_1(q, s, t)$; // вычисляем цвет из текстур уровня q и q + 1 группы G_1

   в противном случае

   2.1. Вычисляем индексы i', j' необходимого тайла в массиве M_q:

   $$i' = \left\lfloor h_T^q \cdot t \right\rfloor,\ \ j' = \left\lfloor w_T^q \cdot s \right\rfloor,$$

   где h_T^q, w_T^q – число строк и столбцов массива M_q

   2.2. Вычисляем индекс k_1 (i', j')-го тайла в массиве J_1 и его номер q' уровня детализации:

   $k_1 = H_{1,q}(i', j'),\ \ q' = J_1(k_1)$.

   2.3. Если $k_1 \geq 0$, то

   - вычисляем текстурные координаты (s', t') пиксела в тайле $J_1(k_1)$:

   $$s' = \left\{h_T^{q'} \cdot t\right\},\ t' = \left\{w_T^{q'} \cdot t\right\};$$

   где $q' \geq q$ – номер уровня детализации тайла $J_1(k_1)$;

   - выполняем выборку цвета C_1 из тайла $J_1(k_1)$ уровня детализации q':

   $C_1 = J_1(k_1, q', s', t')$;

   - выполняем выборку цвета C_2 из тайла $J_1(k_1)$ уровня детализации q'+1:

   $C_2 = J_1(k_1, q'+1, s', t')$;

   - вычисляем цвет C_T интерполяцией цветов C_1 и C_2:

   $$C_T = C_1 \cdot f + C_2 \cdot (1 - f);$$

   в противном случае

   - вычисляем цвет из текстуры уровня L-L'+1 группы G_1:

   $C_T = G_1(\text{L-L}'+1, s, t)$;

   Конец Если.

   Конец Если.

3. Записываем вычисленный цвет C_T в соответствующий пиксел буфера кадра.

Как и в разделе 2.1, для корректной обработки случаев, когда s = 1 или t = 1, в п. 2.1. данного алгоритма вычисленные индексы i', j' необходимо привести к виду $i' = \min\left(i', h_T^q - 1\right)$ и $j' = \min\left(j', w_T^q - 1\right)$.

Предлагаемая в данной статье технология работы со сверхбольшими текстурами реализована в виде комплекса программных модулей для системы визуализации виртуальной поверхности Земли. Пример кадра визуализации приведен на Рис. 5.

Апробация предлагаемой технологии проводилась с помощью трехмерной полигональной модели Земли, содержащей около 150 тыс. полигонов, и текстуры земной поверхности размером $2^{18} \times 2^{17}$ текселов, разбитой на тайлы размером 512×512 текселов. Для визуализации использовались видеокарта GeForce GTX 285 и

экран с разрешением 1920×1080 пикселов. Скорость визуализации на данной установке составила около 150 кадров/с.

Литература

1. Михайлюк М.В., Торгашев М.А. Система «GLVIEW» визуализации для моделирующих комплексов и систем виртуальной реальности // Вестник РАЕН, 2011. №2. С. 20-28.
2. Мальцев А.В., Михайлюк М.В. Регулярные сетки для высоко реалистичной визуализации 3D сцен в реальном времени // Информационные технологии и вычислительные системы, 2012. №4. С. 49-58.
3. Ewins J. et al. MIP-Map Level Selection for Texture Mapping // IEEE Transactions on Visualization and Computer Graphics. – 1998. – Vol. 4, no. 4. – Pp. 317-329.
4. Cline D., Egbert P. Interactive display of very large textures // Proceedings of the conference on Visualization '98. – 1998. – Pp. 343-350.
5. Huttner T. High resolution textures // Proceedings of IEEE Visualization '98. – 1998. – Pp. 13-17.
6. Dollner J. et al. Texturing techniques for terrain visualization // Proceedings of the Conference on Visualization '00. – 2000. – Pp. 227–234.
7. Тимохин П.Ю. Текстурирование больших регулярных сеток виртуальных сцен в имитационно-тренажерных комплексах. // Сб. докл. Межд. Науч. конф., посвященной 80-летию со дня рождения ак. В. А. Мельникова. – М.: 2009. – С. 84-87.
8. Tanner C. et al. The Clipmap: A Virtual Mipmap // SIGGRAPH '98 Proceedings. – 1998. – Pp. 151-158.
9. J.M.P. van Waveren. idTech 5 Challenges: From Texture Virtualization to Massive Parallelization // SIGGRAPH '09. – 2009.
10. Obert J. et al. Virtual Texturing in Software and Hardware // ACM SIGGRAPH '12 Courses – 2012. – Article No. 5.
11. J.M.P. van Waveren, Evan Hart. Using Virtual Texturing to Handle Massive Texture Data // GPU Technology Conference '10. – 2010.
12. OpenGL Extension. Texture Filter Anisotropic. NVIDIA. – 1999. http://www.opengl.org/registry/specs/EXT/texture_filter_anisotropic.txt (дата обращения 26.04. 2013).

Тимохин Петр Юрьевич. Научный сотрудник Центра визуализации и спутниковых информационных технологий НИИСИ РАН. Окончил Российский государственный технологический университет им. К.Э. Циолковского в 2008 году. Автор 15 печатных работ. Область научных интересов: компьютерная графика, информационные технологии.
E-mail: webpismo@yahoo.de

Михайлюк Михаил Васильевич. Заведующий отделом программных средств визуализации Центра визуализации и спутниковых информационных технологий НИИСИ РАН. Окончил Московский государственный университет им. М.В. Ломоносова в 1975 году. Доктор физико-математических наук, профессор. Автор более 120 печатных работ. Область научных интересов: компьютерная графика, системы виртуальной реальности, информационные технологии.
E-mail: mix@niisi.ras.ru