Динамическая оптимизация ландшафта на базе преобразования Хаара и квадродерева вершин

Егор Юсов, Вадим Турлапов
Нижегородский государственный университет им. Н.И.Лобачевского,
Нижний Новгород, Россия
yusov_egor@mail.ru, vadim.turlapov@cs.vmk.unn.ru

Аннотация

Предлагается метод построения адаптивной триангуляции ландшафта (АТЛ), основанный на использовании квадродерева вершин и вейвлет-преобразований. Метод АТЛ отличается тем, что: допускает различие между соседними узлами сетки на любое число уровней иерархии; процесс триангуляции не ограничен размерами сегмента разбиения, то есть одновременно решается проблема оптимальной сшивки сегментов без вставки дополнительных узлов. Алгоритм учитывает при построении триангуляции положение и направление камеры, естественным образом поддерживает геоморфинг и имеет хорошие возможности для конкурирования с другими методами упрощения поверхности.

Ключевые слова: *динамическая визуализация ландшафта, квадродерево, вейвлет-преобразование, уровень детализации, адаптивная триангуляция, временная когерентность, геоморфинг.*

1. ВВЕДЕНИЕ

Высококачественная визуализация земной поверхности в реальном времени является необходимым требованием при создании имитационно-тренажерных комплексов, геоинформацион систем [1], компьютерных игр, а также многих других приложений компьютерной графики. Пространственное моделирование рельефа является трудной задачей и постоянный рост производительности оборудования решает ее, как показывает практика, лишь отчасти. Для оптимизации вывода сцены необходимо использовать специальные алгоритмы, которые позволяют динамически контролировать уровень детализации различных участков поверхности в зависимости от локальных особенностей рельефа, а также от расстояния до камеры и ее ориентации. К настоящему времени разработано довольно много алгоритмов, предназначенных для построения адаптивной триангуляции рельефа путем использования иерархических структур данных (бинарных деревьев, квадродеревьев). Основное назначение таких алгоритмв – сократить число выводимых треугольников без существенной потери качества изображения. В [2] и [3] перечислены требования, которым должна удовлетворять система визуализации ландшафта. Наиболее значимые среди них: минимальная избыточность полученной триангуляции, исключение разрывов в геометрии и текстуре, отсутствие влияния локальных особенностей местности на общую сложность модели, использование межкадровой когерентности (схожести кадров, следующих непосредственно друг за другом), генерация длинных последовательностей полос треугольников (triangle strip) или вееров (triangle fan).

2. ОБЗОР РАБОТ

Обзоры общих методов управления уровнем детализации геометрических моделей представлены в работах [4, 5].

Наиболее просты в реализации алгоритмы, основанные на методе регулярного прореживания ([1, 2, 6]). Однако они порождают слишком избыточную модель, так как уровень детализации контролируется только на уровне целого блока. Другой подход к управлению уровнем детализации основан на использовании прогрессивных сеток [7, 8]. Подобные методы могут применяться к любому объекту, заданному произвольным набором вершин и треугольников. Однако они весьма сложны в реализации и требуют очень высоких вычислительных затрат. Несмотря на то, что они порождают минимально избыточную триангуляцию, общей производительности системы визуализации ландшафта оказывается недостаточно для их использования в реальном времени [2].

Наиболее хорошо для решения задачи упрощения ландшафта зарекомендовали себя алгоритмы, использующие иерархические структуры данных. Среди этих методов можно выделить два основных направления – алгоритмы, основанные на: 1) квадродеревьях вершин [3, 9, 10, 11] и 2) на бинарных деревьях треугольников [12, 13]. Достаточно подробный обзор различных алгоритмов упрощения проведен в [14].

В [15] и [16] представлен близкий к предлагаемому в данной работе подход, основанный на применении вейвлет-анализа и квадродеревьев. Согласно предложенному авторами методу узлам дерева соответствуют квадратные ячейки, вершинами которых являются узлы исходной сетки высот. Уровень детализации ячейки определяется на основе анализа значений вейвлет-коэффициентов, а триангуляция производится отдельно для каждой ячейки путем просмотра специального словаря возможных соединений. При этом авторы накладывают ограничение, что уровни детализации соседних ячеек должны отличаться не более чем на 2.

В отечественной литературе проблеме визуализации ландшафта уделяется существенно меньшее внимание. Можно отметить работы [2, 17, 18].

3. ОПИСАНИЕ МЕТОДА

Новый метод, предлагаемый в данной работе, является развитием подхода, представленного в [18]. Основная идея алгоритма схожа со всеми предыдущим: среди вершин исходной сетки сначала отыскиваются наиболее значимые, а затем на них строится итоговая триангуляция. Основное отличие метода АТЛ заключается в том, что триангуляция строится не только на узлах исходной сетки, но также и на дополнительных узлах, получаемых путем усреднения

координат исходных узлов. Введение усредненных узлов позволяет получать более точную полигональную аппроксимацию карты высот, чем триангуляция, использующая только узлы исходной сетки. Второе существенное отличие заключается в том, что на дерево не накладывается никаких ограничений, и триангуляция может быть построена по любому набору вершин.

Алгоритм, предложенный в [18], предназначался только для оптимизации сетки на этапе предподготовки путем удаления из нее незначащих вершин и не рассчитан на работу в реальном времени. Он не учитывает положение камеры в пространстве, и поэтому не может быть в чистом в виде использован для динамической визуализации ландшафта.

3.1 Построение дерева вершин

Дерево вершин строится на исходной сетке высот, имеющей размеры $2^n \times 2^n$ узлов и образующей *n*-й, самый нижний уровень дерева (считается, что дерево растет вниз). Вся сетка разбивается на квадратные ячейки по 2×2 узла, составленные из четырех соседних вершин, и в центре каждой ячейки размещается новый узел. Этот новый узел считается родителем 4-х узлов ячейки, а сами узлы – его потомками. Координаты каждого центрального узла ячейки определяются путем усреднения координат его потомков. Описанным способом формируется *n*-1 уровень квадродерева, являющийся сеткой из $2^{n-1} \times 2^{n-1}$. Аналогично получаются *n*-2, *n*-3 уровни и так далее до 0 уровня, содержащего единственную корневую вершину. Каждый следующий уровень иерархии является огрубленной версией нижележащего, т.о. формируется многомасштабное представление ландшафта. В терминологии вейвлет-преобразований сетка, образующая *i* уровень дерева называется сигналом разрешения *i*. На рис. 1 представлено трехуровневое дерево, построенное на сетке 4×4 узла.

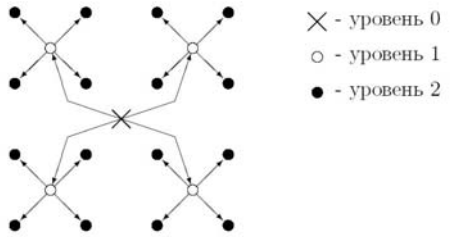


Рис. 1. Трехуровневое квадродерево

3.2 Выделение множества наиболее значимых вершин при помощи преобразования Хаара

3.2.1 Структура вейвлет-разложения сигнала

Вейвлет-преобразование – это разложение сигнала по системе функций, являющихся сдвинутыми и масштабированными копиями одной функции – *порождающего вейвлета*. Распространенный случай – *диадное* вейвлет-преобразование, с помощью которого сигнал $f(x) \in L_2(R)$ может быть представлен в виде следующего разложения:

$$f(x) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} w_j^i \sqrt{2^i} \psi(2^i x - i)$$

Порождающим вейвлетом в данном случае является функция $\psi(x)$. Минимальными требованиями к порождающему вейвлету являются пространственная локализация и наличие хотя бы одного нулевого момента.

Частичную сумму ряда

$$f^{(i_0)}(x) = \sum_{i=-\infty}^{i_0-1} \sum_{j=-\infty}^{+\infty} w_j^i \sqrt{2^i} \psi(2^i x - i), \quad i_0 \in \mathrm{Z}$$

называют приближением сигнала $f(x)$ с разрешением i_0. Таким образом, сигнал $f(x)$ может быть представлен в виде суммы $f^{(i_0)}(x)$ (грубого приближения) и оставшихся членов ряда (детализирующей информации).

3.2.2 Вейвлет-преобразования дискретных сигналов

Одномерный дискретный сигнал задается последовательностью $s = \{s_j\}_{j \in Z}$. Вейвлет-преобразование ставит в соответствие сигналу *s* два новых сигнала $v = \{v_j\}_{j \in Z}$ и $w = \{w_j\}_{j \in Z}$:

$$v = \downarrow_2 [s * h],\ w = \downarrow_2 [s * g]$$

где *h* и *g* – фильтры, выполняющие свертку сигнала, $\downarrow_2 [\bullet]$ – оператор удаления каждого второго элемента последовательности. Например, преобразование Хаара задается следующей парой фильтров: $h = [1/2, 1/2], g = [1/2, -1/2]$. Поэлементно сигналы *v* и *w* задаются следующими формулами:

$$v_j = (s_{2j} + s_{2j+1})/2,\ w_j = (s_{2j} - s_{2j+1})/2 \cdot$$

Сигнал *v* является огрубленной версией исходного сигнала, а сигнал *w* несет детализирующую информацию. К сигналу *v* снова может быть применено вейвлет-преобразование, которое породит два новых сигнала. Таким образом, мы получим последовательность сигналов $v^{(i)}$ (приближений исходного сигнала с различными уровнями разрешения) и $w^{(i)}$ (детализирующей информации).

В случае, когда исходный сигнал является конечным, вместо последовательности мы имеем конечный набор значений: $s = \{s_j\}_{j=0}^{2^l-1}$. Тогда на каждом шаге преобразования сигналы *v* и *w* будут получаться в два раза короче, и после выполнения *i* шагов мы получим единственное значение.

3.2.3 Вейвлет-преобразования двумерных конечных дискретных сигналов

В двумерном случае сигнал $s = \{s_{j,k}\}_{j,k=0}^{2^l-1}$ представляется матрицей. Вейвлет преобразование двумерного сигнала состоит из двух шагов. Вначале к каждой строке матрицы применяется одномерное вейвлет-преобразование, в результате чего получаются две новые матрицы. Затем к каждому столбцу полученных матриц снова применяется одномерное преобразование. В итоге получаются четыре матрицы, одна из которых является огрубленной версией исходной ($v_{j,k}^{(i)}$), а три других ($vw_{j,k}^{(i)}$, $wv_{j,k}^{(i)}$ и $ww_{j,k}^{(i)}$) несут детализирующую информацию.

Двумерное преобразование Хаара описывается следующими формулами:

$$v_{j,k}^{(i)} = \tfrac{1}{4}\left(v_{2j,2k}^{(2i+1)} + v_{2j+1,2k}^{(2i+1)} + v_{2j,2k+1}^{(2i+1)} + v_{2j+1,2k+1}^{(2i+1)}\right)$$
$$vw_{j,k}^{(i)} = \tfrac{1}{4}\left(v_{2j,2k}^{(2i+1)} + v_{2j+1,2k}^{(2i+1)} - v_{2j,2k+1}^{(2i+1)} - v_{2j+1,2k+1}^{(2i+1)}\right)$$
$$wv_{j,k}^{(i)} = \tfrac{1}{4}\left(v_{2j,2k}^{(2i+1)} - v_{2j+1,2k}^{(2i+1)} + v_{2j,2k+1}^{(2i+1)} - v_{2j+1,2k+1}^{(2i+1)}\right)$$
$$ww_{j,k}^{(i)} = \tfrac{1}{4}\left(v_{2j,2k}^{(2i+1)} - v_{2j+1,2k}^{(2i+1)} - v_{2j,2k+1}^{(2i+1)} + v_{2j+1,2k+1}^{(2i+1)}\right)$$

В терминологии вейвлет-преобразований $v_{j,k}^{(i)}$ называется низкочастотной составляющей сигнала *i*-го уровня разрешения, а $vw_{j,k}^{(i)}$, $wv_{j,k}^{(i)}$ и $ww_{j,k}^{(i)}$ – высокочастотными составляющими. Последние три числа также называются вейвлет-коэффициентами.

3.2.4 Выделение набора значимых вершин

Выделение набора наиболее значимых вершин осуществляется на основе статических погрешностей, приписываемых узлам всех уровней от *n*-1 до 0-го. Это значение вычисляется при помощи преобразования Хаара, применяемого к исходной карте высот (выступающей в роли двумерного входного сигнала). Благодаря свойству пространственной локализации вейвлет-преобразования, абсолютные значения коэффициентов будут велики в тех областях, где поверхность имеет особенности, и малы там, где поверхность изменяется незначительно. Поддерево, все коэффициенты которого пренебрежимо малы, называется *нуль-поддеревом*. Не внося значительной ошибки в полигональное представление рельефа, все листовые вершины нуль-поддерева можно заменить одной вершиной, являющейся его корнем.

Каждому узлу *i* уровня дерева соответствуют три вейвлет коэффициента, полученные применением преобразования Хаара к сигналу *i*-1 уровня. Для решения задачи выделения наиболее значимых узлов для каждого узла дерева достаточно хранить только одно значение, являющееся к.-л. нормой вектора коэффициентов. Для экономии вычислений и по геометрическому содержанию задачи предпочтительнее чебышева норма $\| \cdot \|_\infty$:

$$\varepsilon_{j,k}^{(i)} = \max\left(\left|vw_{j,k}^{(i)}\right|, \left|wv_{j,k}^{(i)}\right|, \left|ww_{j,k}^{(i)}\right|\right)$$

После удаления всех незначимых вершин, расположение узлов будет адаптировано к особенностям конкретной поверхности. Их плотность будет высока там, где рельеф имеет мелкие высокочастотные особенности, и низка в областях, где поверхность гладкая и не имеет мелких деталей.

3.3 Вычисление экранной погрешности вершины на основе статической

Статическую погрешность вершины следует понимать как характеристику скорости изменения рельефа в данной области. Но итоговая сетка должна быть адаптирована не только к локальным особенностям поверхности, но и к положению и ориентации камеры в пространстве. Поэтому для решения вопроса о включении узла в итоговую триангуляцию для него необходимо вычислять экранную погрешность. Последняя должна отражать максимальное видимое на экране отклонение рельефа от точного изображения, которое может возникнуть, если данный узел станет корнем нуль-поддерева. Рис. 2 иллюстрирует способ определения экранной погрешности узла. На нем через *C* обозначена позиция камеры, *P* – положение узла, ε – его статическая погрешность, $\vec{e}_h$ – единичный вектор оси *Z* пространства, *d* – расстояние от камеры до ближайшей точки ограничивающего бокса, $\vec{e}_P = (P-C)/\|P-C\|$ – единичный вектор направления от камеры к узлу.

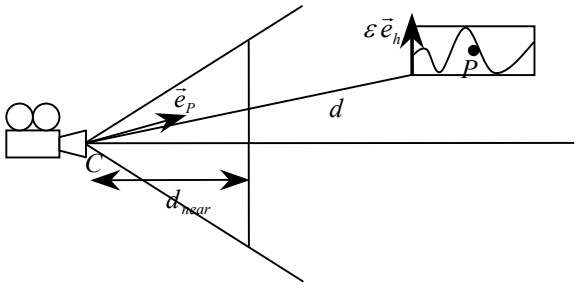


Рис. 2. Вычисление экранной погрешности

Поскольку заранее нельзя предугадать, на каком узле поддерева будет достигнуто максимальное отклонение, необходимо ориентироваться на наихудший случай, то есть рассматривать точку ограничивающего бокса, наиболее близкую к положению камеры. Экранную погрешность узла *P* предлагается вычислять по следующей формуле:

$$\varepsilon_{Scr} = \frac{d_{near} \cdot s}{d}\sqrt{1-(\vec{e}_P \bullet \vec{e}_h)^2}$$

где d_{near} – расстояние до ближней отсекающей плоскости пирамиды видимости, а $s = \frac{1}{2}\sqrt{V_x^2 + V_y^2}$ – коэффициент, учитывающий размеры экрана, где V_x и V_y – горизонтальный и вертикальный размеры области вывода в пикселях.

В [11] предложен похожий способ, однако он принимает в расчет только расстояние от камеры до узла. Кроме того, предложенный в [11] критерий использует ограничивающую сферу, которая дает слишком большой запас точности для участков рельефа с незначительными колебаниями высот, тогда как ограничивающий бокс позволяет сделать более точную оценку. В отличие от критерия предложенного в [11] и других критериев, представленная формула учитывает и расстояние, и расположение узла относительно камеры (за счет $\sqrt{1-(\vec{e}_P \bullet \vec{e}_h)^2}$). Экранная погрешность убывает не только с ростом расстояния до узла, она также мала в тех областях, где перепады высоты не видны с позиции камеры.

Для каждого узла также можно определять ограничивающую сферу, и с ее помощью осуществлять отбраковку невидимых поддеревьев, как это предложено в [11]. Если охватывающая сфера узла лежит вне конуса видимости, то его экранная погрешность принимается равной 0 и этот узел заменяет все вершины, которые содержатся в растущем из него поддереве.

3.4 Алгоритм определения нуль-поддеревьев

Каждому узлу дерева приписывается один из трех типов: корень нуль-поддерева (NullSubTreeRoot), узел, принадлежащий некоторому нуль-поддереву (NullSubTreeSubNode) и узел, не являющийся ни корнем, ни узлом нуль поддерева (NotNullNode). На любом пути, ведущем из корневой вершины дерева к его листовой вершине, обязательно встречается, и притом только один узел, являющийся корнем нуль-поддерева (NullSubTreeRoot). Все узлы пути, лежащие до него выше в дереве (ближе к корню), если такие есть, являются NotNullNode, а все узлы, лежащие ниже в дереве (если такие есть), имеют тип NullSubTreeSubNode. Итоговая триангуляция строится на узлах, имеющих тип NullSubTreeRoot. Приписывание узлам типов позволяет существенно упростить и, как следствие, ускорить процесс построения триангуляции.

Для обеспечения экономии вычислений предлагается использовать следующие два алгоритма определения нуль-поддеревьев. Первый – восходящий нерекурсивный, работает один раз вначале и обрабатывает все узлы. Он начинает работу с узлов *n*-го уровня и далее последовательно обрабатывает все узлы всех уровней до 0-го. Коротко он может быть описан следующим псевдокодом:

```
PROCEDURE NonRecursiveDetermineNullSubTrees
BEGIN
   FOREACH Node in Level n DO
      Node.Type := NullSubTreeRoot
   END
   FOR i = n-1 DOWNTO 0 DO
      FOREACH Node in Level i DO
         IF Type of all children is NullSubTreeRoot THEN
            IF ComputeScreenError(Node) < Threshold THEN
               Node.Type := NullSubTreeRoot
               FOREACH Child of Node DO
                  Child.Type := NullSubTreeSubNode
               END
            ELSE //if screen error is greater than threshold
               Node.Type := NotNullNode
            ENDIF
         ELSE //if type of some child is not NullSubTreeRoot
            Node.Type := NotNullNode
         ENDIF
      END //FOREACH Node
   END //FOR i
END //NonRecursiveDetermineNullSubTrees
```

Алгоритм сначала помечает все узлы самого нижнего уровня как корни нуль-поддеревьев, после чего начинает обрабатывать верхние уровни. Если хотя бы один из четырех потомков очередного узла не является корнем нуль-поддерева, то текущий узел не может быть корнем нуль-поддерева, и помечается как NotNullNode (его экранная погрешность в этом случае не вычисляется). Если же все четыре потомка текущего узла являются корнями нуль-поддеревьев, и при этом его экранная погрешность меньше порога, то узел помечается как NullSubTreeRoot, а его потомки – как NullSubTreeSubNode. Если экранная погрешность больше порога, то узел помечается как NotNullNode.

Нерекурсивный алгоритм применяется только при первом отображении дерева для начального определения типов узлов. Второй алгоритм (нисходящий) является рекурсивным, он начинает работу с корня дерева и спускается вниз до требуемой глубины. Алгоритм учитывает свойство межкадровой когерентности, заключающееся в том, что структура дерева от одного кадра к другому меняется незначительно. Поэтому алгоритм предполагает лишь возможные изменения на один уровень. В итоге, этот алгоритм обрабатывает лишь малую часть от общего числа узлов. Алгоритм описывается следующим псевдокодом:

```
PROCEDURE RecursiveDetermineNullSubTrees(VAR Node:
                QuadTreeNode)
BEGIN
   IF Node.Type = NotNullNode
      IF Type of all children is NullSubTreeRoot THEN
         IF ComputeScreenError(Node) < Threshold THEN
            Node.Type = NullSubTreeRoot
            FOREACH Child of Node DO
               Child.Type := NullSubTreeSubNode
            END
         ELSE //if screen error > Threshold
            FOREACH Child of Node DO
               RecursiveDetermineNullSubTrees(Child)
            END
         ENDIF
      ELSE // if type of some child is not NullSubTreeRoot
         FOREACH Child of Node DO
            RecursiveDetermineNullSubTrees(Child)
         END
      ENDIF
   ELSE IF Node.Type = NullSubTreeRoot
      IF ComputeScreenError(Node) > Threshold THEN
         Node.Type = NotNullNode
         FOREACH Child of Node DO
            Child.Type := NullSubTreeRoot
         END
   ENDIF
END RecursiveDetermineNullSubTrees
```

Алгоритм начинает работу с корня дерева и функционирует следующим образом: если текущий узел имеет тип NotNullNode, то для того, чтобы он стал корнем нуль подерева должны выполниться два условия: все его потомки уже являются корнями нуль-поддеревьев и его экранная погрешность меньше порогового значения. Если оба эти условия выполнены, то узел помечается как NullSubTreeRoot, а все его потомки становятся узлами нуль-поддерева. Если экранная погрешность больше порога, то необходимо запустить алгоритм для каждого потомка. Если хотя бы один из потомков узла не является корнем нуль-поддерева, то экранная погрешность не вычисляется, и алгоритм запускается рекурсивно для каждого из 4 потомков.

Если узел имеет тип NullSubTreeRoot, то необходимо вычислить его экранную погрешность и сравнить ее с порогом. Если проверка дает положительный результат, то узел остается корнем нуль-поддерева, в противном случае он помечается как NotNullNode, а все его потомки становятся корнями нуль-поддеревьев.

Наиболее сложную операцию – вычисление экранной погрешности – алгоритм выполняет только для той части поддерева, в которой возможны изменения, экономя таким образом вычислительные ресурсы. Время работы алгоритма пропорционально текущей сложности локальной модели и не зависит от объема исходной информации о поверхности. Предложенная схема также позволяет явно контролировать качество создаваемой модели и прекращать добавление новых узлов, когда достигнуто некоторое ограничение.

3.5 Построение триангуляции

3.5.1 Основное правило

Триангуляция строится на множестве вершин, помеченных как NullSubTreeRoot. Если листовая вершина (лежащая в уровне с максимальным разрешением) не входит ни в одно нуль-поддерево, она помечается как корень нуль-поддерева (которое в этом случае не содержит ни одного узла).

Основное правило построения весьма простое: триангуляция строится так, как если бы в полной триангуляции максимального разрешения все вершины, находящиеся в нуль-поддеревьях, «притянулись» к их корням, и все вырожденные треугольники после этого были отброшены.

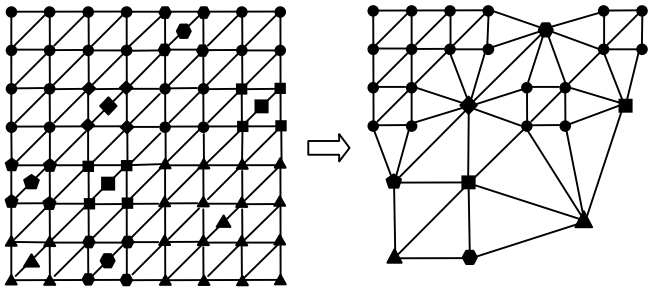


Рис. 3. Пример триангуляции дерева

Рис. 3 иллюстрирует это правило. На нем разными фигурами обозначены вершины, принадлежащие различным нуль-поддеревьям. Крупными фигурами обозначены корни этих поддеревьев, к которым «притягиваются» все вершины той же формы

3.5.2 Алгоритм построения триангуляции

Полный алгоритм построения достаточно объемен, и полностью рассмотреть его здесь не представляется возможным. Поэтому придется ограничиться только кратким описанием основных моментов. Из рис. 3 можно заметить, что триангуляция содержит довольно много фэнов (вееров), то есть последовательностей треугольников, имеющих одну общую вершину. Алгоритм использует это свойство триангуляции. Он начинает работу с корня дерева и спускается вниз по дереву до всех вершин, являющихся корнями нуль-поддеревьев. Для каждой такой вершины алгоритм составляет фэн с учетом того, кем являются ее соседи. Алгоритм последовательно обрабатывает соединения с верхним, левым, нижним и правым соседом, рассматривая все возможные варианты и добавляя необходимые вершины в веер.

К примеру, обработка соединения с верхним соседом производится согласно схемам, представленным на рис. 4. Текущая обрабатываемая вершина обозначена на схемах жирным крестиком и расположена в правом нижнем углу.

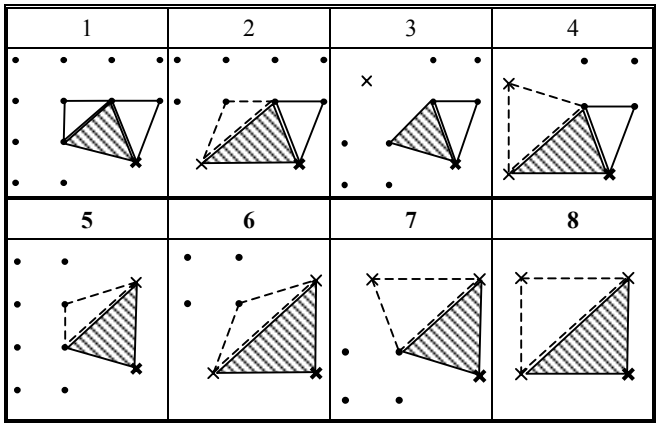


Рис. 4. Различные варианты соединения узла с верхним соседом

Если верхний сосед имеет тип NotNullNode (варианты 1-4), то сначала в веер добавляются все вершины, имеющие тип NullSubTreeRoot и образующие нижнюю границу поддерева, корнем которого является верхний сосед. Для этого используется простой рекурсивный алгоритм, начинающий работу с верхнего соседа:

```
PROCEDURE AddVertsFromBottomBorderToFan (VAR Node:
                QuadTreeNode)
BEGIN
  IF BottomLeftChild.Type   = NullSubTreeRoot AND
     BottomRightChild.Type = NullSubTreeRoot THEN
     AddVertexToFan(BottomRightChild)
     AddVertexToFan(BottomLeftChild)
     END
  ELSE IF BottomLeftChild.Type   = NullSubTreeRoot AND
            BottomRightChild.Type = NotNullNode THEN
     AddVertsToFanFromBottomBorder(BottomRightChild)
     AddVertexToFan(BottomLeftChild)
     END
  ELSE IF BottomLeftChild.Type   = NotNullNode AND
            BottomRightChild.Type = NullSubTreeRoot THEN
     AddVertexToFan(BottomRightChild)
     AddVertsToFanFromBottomBorder(BottomLeftChild)
     END
  ELSE
     AddVertsToFanFromBottomBorder(BottomRightChild)
     AddVertsToFanFromBottomBorder(BottomLeftChild)
  ENDIF
END AddVertsFromBottomBorderToFan
```

Если верхний сосед является корнем нуль-поддерева (случаи 5-8), то он сразу добавляется в веер. Треугольники, обозначенные на схемах пунктирными линиями, будут отображены при обработке других вершин, поэтому при обработке текущего узла они пропускаются.

Если верхний сосед имеет тип NullSubTreeSubNode, то есть он лежит в нуль-поддереве, то верхний сосед пропускается, а соединение с текущей вершиной будет обработано, когда алгоритм дойдет до корневой вершины поддерева, в котором находится верхний сосед.

После этого начинается обработка соединений с левым соседом. Треугольники, которые при этом добавляются, на схемах заштрихованы. Затем похожим образом обрабатываются соединения с нижним и правым соседями. Для каждого из них также используются похожие таблыицы, согласно которым и осуществляется триангуляция. При гарантируется, чтобы каждый треугольник был добавлен только один раз.

Описанный способ чем-то похож на метод, изложенный в [15] и [16], который основан на использовании словаря различных триангуляций. В нашем алгоритме таблицы являются лишь схемами, по которым должны добавляться вершины в фэн для получения итогового результата, но не конечными вариантами триангуляций. Для наглядности в таблице представлены случаи, когда уровни узлов отличаются на 1, однако метод точно так же работает и в случае перехода больше чем на 1 уровень. В отличие от алгоритма, представленного в [15,16], где изменение уровня детализации ограничено двумя, в предлагаемом методе подобных ограничений не накладывается.

3.5.3 Исключение разрывов

Важной проблемой при построении триангуляции является исключение разрывов (cracks). В большинстве алгоритмов необходимым условием этого является наложение ограничения, что узлы дерева должны отличаться не более чем на 1, иногда на 2, но не более, уровней в иерархии. В иных случаях это достигается за счет сложных схем введения дополнительных вершин или треугольников, требующих

излишних вычислительных затрат. Например, в [3] и [9] ограничение реализуется в виде отношений зависимости на узлах квадродерева. Включение одной вершины в триангуляцию вызывает включение всех вершин, которые напрямую или косвенно от нее зависимы, что приводит к ненужному увеличению сложности результирующей модели. Аналогичное требование налагается на двоичное дерево, используемое в алгоритме ROAM [12].

В отличие от всех других методов, триангуляция, построенная по предлагаемому правилу, лишена этого недостатка. Она гарантирует отсутствие разрывов без введения дополнительных вершин и дополнительных треугольников в сетку, даже если уровни соседних вершин отличаются больше, чем на 1 (что и подтверждает рис. 3).

Как правило, большие участки рельефа представляются в виде совокупности блоков (патчей), каждый из которых триангулируется отдельно. На границах блоков также возможно возникновение разрывов. Чаще всего для их исключения в результирующую модель также добавляются новые вершины [9]. Однако вставка вершины на границе патча может повлечь за собой включение в триангуляцию дополнительных вершин, согласно графу зависимости. Причем они могут появиться в том числе и на границе блока, что может явиться причиной появления нового разрыва.

Особенностью описанного метода является то, что для него не играет роли, принадлежит вершина текущему патчу, или соседнему. В обоих случаях триангуляция строится совершенно одинаково, при этом гарантируется отсутствие разрывов без добавления новых узлов и треугольников.

3.6 Геоморфинг

Для улучшения качества визуального восприятия сцены переключения между уровнями детализации должны происходить как можно незаметнее. Поэтому изменения геометрии, сопровождающие переход от одного уровня детализации к другому, должны осуществляться не скачкообразно, а путем плавного «перетекания» одного уровня в другой. Этот процесс называется *геоморфингом*. Предлагаемая структура квадродерева очень хорошо подходит для его реализации. При увеличении уровня детализации происходит разбиение одной вершины на 4, а при уменьшении – наоборот, 4 вершины сливаются в одну. Поскольку камера в пространстве перемещается плавно, значения экранных погрешностей узлов также изменяются плавно. Поэтому для реализации морфинга достаточно начать перемещение четырех вершин к их родителю, когда значение экранной погрешности становится близкой к порогу слияния. Близость положения потомков к родителю определяется на основе текущей экранной погрешности (а не по времени, как это предлагается, например, в [11]).

Морфинг вершины выполняется линейной интерполяцией между ее исходным положением и положением ее родителя, когда экранная погрешность родителя находится в интервале $[\tau, \lambda\tau]$, где τ - порог слияния, а $\lambda > 1$ - коэффициент начала морфинга. Это можно выразить следующей формулой:

$C_i = \alpha C_i^0 + (1-\alpha)P$ где $\alpha = (\varepsilon_{\text{Scr}} - \tau)/(\lambda\tau - \tau)$.

Здесь $P \in R^3$ – положение вершины-родителя, ε_{Scr} – ее экранная погрешность, $C_i^0 \in R^3, i = \overline{1,4}$ – исходные положения в пространстве четырех ее потомков, а $C_i \in R^3, i = \overline{1,4}$ – их текущие положения в процессе морфинга. Смещение всех четырех потомков вершины осуществляется одновременно с одинаковым коэффициентом α. В начальный момент, когда экранная погрешность родителя равна $\lambda\tau$, все 4 вершины занимают свои исходные положения. По мере уменьшения ε_{Scr} вершины будут плавно перемещаться все ближе и ближе к их родителю, и в момент, когда экранная погрешность станет равна порогу слияния, все они будут находится точно там же, где их родитель. Этот процесс проиллюстрирован на рис. 5. Обратный процесс происходит абсолютно аналогично.

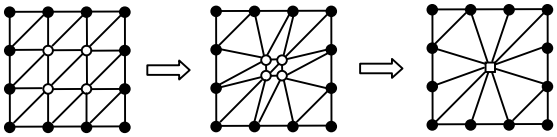


Рис. 5. Геоморфинг

Способ построения триангуляции гарантирует, что после слияния 4-х вершин в одну, дерево будет триангулировано точно также, как если бы эти 4 вершины присутствовали, но все находились в одной точке. Поэтому переключение не будет заметно. Следует отметить, что предложенная схема может быть эффективно реализована в вершинном шейдере.

4. РЕЗУЛЬТАТЫ

В приложении приведены результаты работы алгоритма на примере визуализации ландшафта Африки, восстановленного по ее географической карте, в сравнении с алгоритмом QUADTin, описанным в [9]. Размер сетки 512x512, сетка разбита на 16 патчей по 128x128 узлов. Целью данных иллюстраций является демонстрация принципиальных отличий триангуляции, порожденной АТЛ, а также демонстрация степени упрощения модели и доли вершин (в примерах – от 1.3% до 7.3%), обрабатываемой нисходящим алгоритмом.

Рисунки хорошо иллюстрируют тот факт, что предложенная схема построения триангуляции позволяет лучше адаптировать триангуляцию к рельефу. В тех местах, где рельеф изменяется быстро, алгоритм размещает большее число треугольников. За счет возможности быстрой смены уровня детализации алгоритм может большее число треугольников использовать в тех областях, где происходит быстрое изменение рельефа, в то время как алгоритму QUADTin необходимы дополнительные треугольники, соединяющие области с разными уровнями детализации.

5. ЗАКЛЮЧЕНИЕ

В работе представлен новый метод построения адаптивной полигональной аппроксимации рельефа при помощи вейвлет-анализа и квадродеревьев. Алгоритм строит триангуляцию не только на узлах исходной сетки, но также использует дополнительные, усредненные узлы. В отличие от всех имеющихся методов, для построения триангуляции, не имеющей разрывов, алгоритм не накладывает на дерево никаких ограничений и не требует, чтобы соседние узлы триангуляции лежали в соседних уровнях иерархии. Метод построения триангуляции гарантирует отсутствие разрывов как внутри одного блока, так и на границах патчей. Реализован улучшенный способ вычисления экранной погрешности узлов, учитывающий не только расстояние до

камеры, но и ее ориентацию, рассмотрена реализация геоморфинга, расширяющая возможности метода.

БИБЛИОГРАФИЯ

[1] Jan Vaněk, Bruno Ježek. *Real Time Terrain visualization on PC.* Proc. of the 12th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2004– W S C G ' 2004, February 2 - 6, 2004.

[2] Елыков Н.А., Белаго И.В., С.А. Кузиковский, Некрасов Ю.Ю. *Методы непрерывной детализации террэйна.* Материалы конф. по комп. граф. и визуализации – GraphiCon'2002, Н.Новгород, 16-21 сент. 2002.

[3] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. *Real-time, continuous level of detail rendering of height fields.* In Proceedings SIGGRAPH 96, pages 109–118. ACM SIGGRAPH, 1996.

[4] P. Heckbert, and M. Garland. *Survey of polygonal surface simplification algorithms.* SIGGRAPH 97 Course Notes 25, 1997.

[5] David Luebke. *A developer's survey of polygonal simplification algorithms.* IEEE Computer Graphics & Applications, 21(3):24–35, May/June 2001.

[6] Bent Dalgaard Larsen, Niels Jørgen Christences. *Real-time Terrain Rendering using Smooth Hardware Optimized Level of Detail.* Journal of WSCG, Vol.11, No.1, ISNN 1213-6972, WSCG'2003, February 3 -7, 2003, Plzen, Czech Republic.

[7] Michael Garland and Paul S. Heckbert. *Fast polygonal approximation of terrains and height fields.* Technical Report cmu-cs-95-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1995.

[8] H. Hoppe. *Progressive meshes.* In Proceedings SIGGRAPH 96, pages 99–108. ACM SIGGRAPH, 1996.

[9] Renato Pajarola. *Large scale terrain visualization using the restricted quadtree triangulation.* In Proceeding Visualization 98. IEEE Computer Society Press, 1998.

[10] Laurent Balmelli and Jelena Kovaˇcevic, Martin Vetterli. *Quadtrees for Embedded Surface Visualization, Constraints and Efficient Data Structures.* In Proc. of IEEE International Conf. on Image Processing (ICIP),Vol. 2, p.487-491, October 1999.

[11] Renato Pajarola. *QuadTIN: Quadtree based Triangulated Irregular Networks.* In roceedings IEEE Visualization 2002 pages 395–402. IEEE Computer Society Press, 2002.

[12] M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. *Roaming terrain: Real-time optimally adapting meshes.* In Proceedings of Visualization 97, pages 81–88. IEEE, Computer Society Press, Los Alamitos, California, 1997.

[13] Williams Evans, David Kirkpatrick, and Greg Townsend. *Right-triangulated irregular networks*. Technical Report 97-09, Department of Computer Science, University of Arizona, 1997. to appear in Algorithmica.

[14] Renato Pajarola. *Overview of quadtree-based terrain triangulation and visualization.* Technical Report UCI-ICS-02-01, I&C Science, University of California Irvine, 2002.

[15] Markus H. Gross, Roger Gatti, and Oliver G. Staadt. *Fast multiresolution surface meshing.* In Proc. of 14th International Conf. on Data Engineering, ICDE'98, pp. 550–557. IEEE, 1998.

[16] Gross M.H., Staadt O.G., Gatti R. *Efficient Triangular Surface Approximations Using Wavelets and Quadtree Data Structures.* IEEE Trans. on Visualization and Computer Graphics. Vol. 2, No. 2, June 1996, pp. 130-143.

[17] Vladislav I. Suglobov *Appearance-Preserving Terrain Simplification.* Proc. of the Conf. on Comp. Graph. and Visual. - GraphiCon'2000, Moscow, August 28 - September 2, 2000.

[18] Переберин А.В. *Многомасштабные методы синтеза и анализа изображений.* Диссертация на соискание ученой степени канд. физ.-мат. наук, Институт прикладной математики им. М.В. Келдыша, Москва-2002.

Об авторах

Юсов Егор – аспирант факультета Вычислительной Математики и Кибернетики Нижегородского государственного университета, email: yusov_egor@mail.ru

Турлапов Вадим – профессор кафедры Математического обеспечения ЭВМ Нижегородского государственного университета, e-mail: vadim.turlapov@cs.vmk.unn.ru

Dynamic terrain simplification based on Haar transform and vertices quadtree

Abstract

In this paper a new real time terrain simplification and visualization algorithm is proposed. The algorithm is based on wavelet analysis and quadtrees. New method overcomes constraint that two adjacent nodes of the quadtree must differ by at most one level in the LOD hierarchy. A new efficient triangulation method is discussed. The method produces cracks free triangulation for patch interior as well as for patch junctions without introducing additional vertices and triangles into model. A new screen error which takes into account both distance to camera and its orientation and implementation of geomorphing are also proposed.

Keywords: *dynamic adaptive terrain simplification and visualization, wavelet transform, quadtree, LOD, geomorphing.*

About the authors

Egor Yusov is a Ph.D. student at Nizhny Novgorod State University, Department of Computational Mathematics and Cybernetics, email: yusov_egor@mail.ru

Vadim Turlapov is a professor at Nizhny Novgorod State University, Department of Computational Mathematics and Cybernetics, email: vadim.turlapov@cs.vmk.unn.ru

Приложение

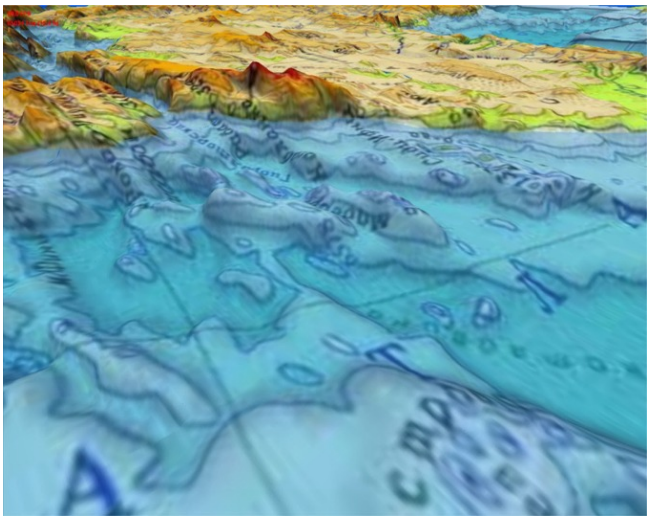
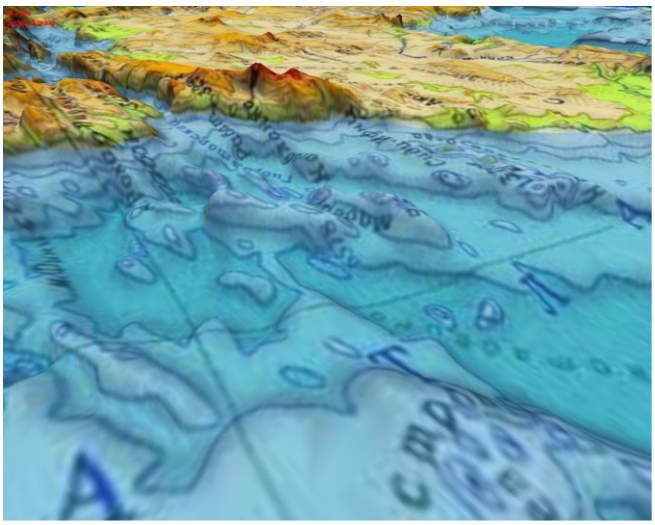


Рис. 6. а) Полное разрешение, 524288 треугольников, 100%



б) АТЛ, 38530 треугольников, 7.3%

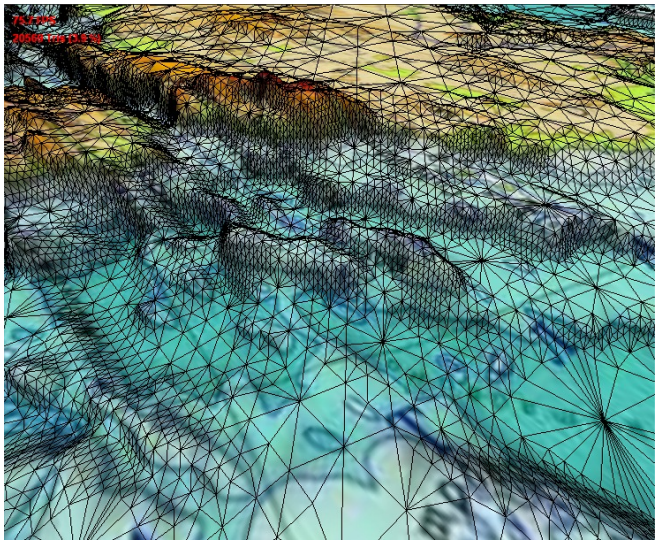
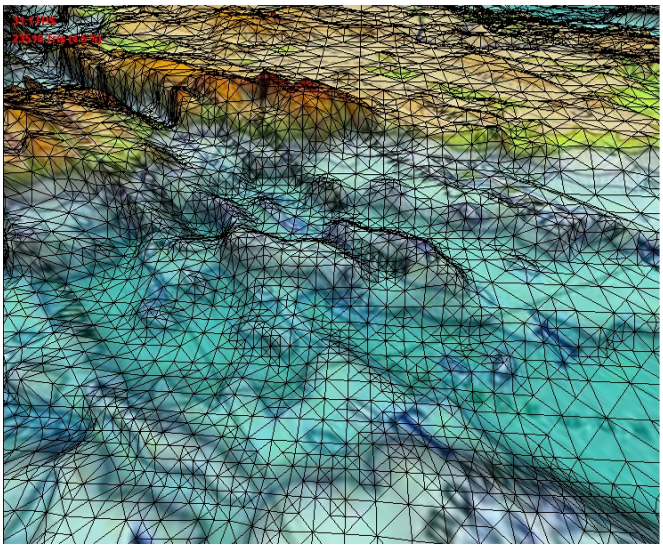


Рис. 7. а) АТЛ, 20569 треугольников, 3.9%



б) Алгоритм QUADTin, 23510 треугольников, 4.6%

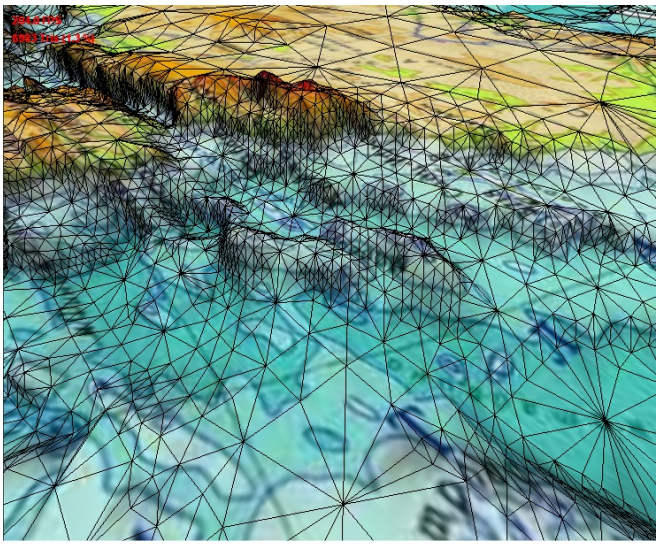
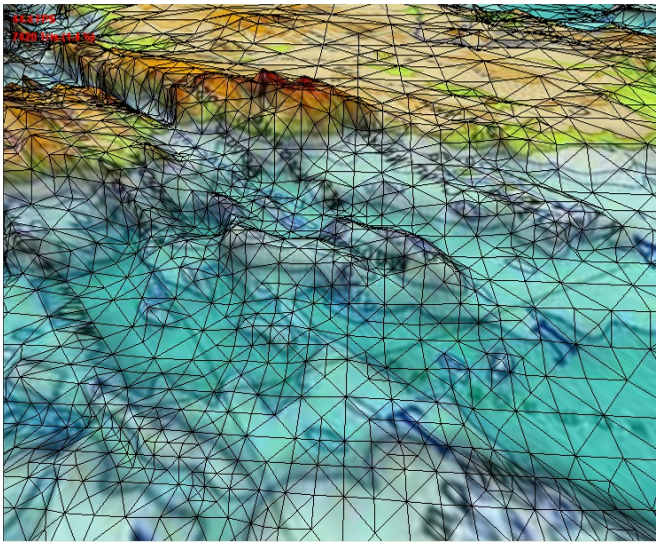


Рис. 8. а) АТЛ, 6983 треугольника, 1.3%



б) Алгоритм QUADTin, 7420 треугольников, 1.4%