РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
А В Т О М Е Т Р И Я

№ 2 2001

УДК 681.3.069

Н. А. Елыков, И. В. Белаго, С. А. Кузиковский, Ю. Ю. Некрасов

(Новосибирск)

ОБ ОДНОМ ПОДХОДЕ К ВИЗУАЛЬНОЙ ИМИТАЦИИ ДИНАМИЧЕСКОЙ МОРСКОЙ ПОВЕРХНОСТИ

Описана методика генерации и анимации изображения морской поверхности в реальном времени. Проведено сравнение нескольких моделей поведения морской поверхности. Детально описан алгоритм управления детализацией геометрии поверхности в зависимости от положения наблюдателя. Предложены методы генерации и анимации текстуры, накладываемой на морскую поверхность.

Введение. Бурный рост производительности и качества графических средств персональных компьютеров в последние годы и глобальная стандартизация графических программных интерфейсов существенно расширили применение PC в качестве аппаратной платформы для графических приложений, приложений виртуальной реальности и симуляторов.

Моделирование визуальной обстановки в настоящее время – одна из наиболее динамичных и бурно развивающихся областей в компьютерной индустрии. Программные модели виртуального мира приобретают все большую схожесть с миром реальным. Благодаря присущей ему сложности, реалистичное моделирование природных феноменов является одной из наиболее вычислительно-емких областей компьютерной графики. Один из таких феноменов – это морские сцены. Данная статья посвящена исследованию методов визуальной имитации морских поверхностей в реальном времени.

Авторами исследованы и проанализированы подходы к решению проблемы и выбран метод, позволяющий генерировать реалистичные изображения морской поверхности в реальном времени. Предлагаемый метод учитывает эволюцию морской поверхности во времени в зависимости от направления и скорости ветра, формы морского дна и береговой линии.

Описываемая методика генерации изображения морской поверхности состоит из трех частей. Первая часть представляет собой алгоритм, позволяющий получить форму поверхности моря в зависимости от времени и других факторов. Вторая часть позволяет построить оптимальную с точки зрения соотношения качество/производительность триангуляцию морской поверхности. Третья часть посвящена методам получения текстур для морской поверхности.

1. **Техника генерации и анимации формы морской поверхности.** Для возможности эффективного использования алгоритм должен удовлетворять

ряду требований, сформулированных, например, в [1]: 1) генерировать волны, похожие на реальные; 2) не требовать больших вычислительных ресурсов; 3) иметь простую и реалистичную анимацию; 4) генерировать широкий диапазон волн (от небольшой ряби до штормовых волн); 5) моделировать такое явление, как рефракция, т. е. учитывать форму дна (скорость и направление распространения волны зависят от глубины); 6) предоставлять дизайнерам возможность задавать такие физические параметры, как направление и скорость ветра, линия берега и форма поверхности дна; 7) предоставлять возможность получения точки поверхности в произвольном месте (для нерегулярной триангуляции).

Поверхность моря принято задавать с помощью набора высот в узлах равномерной прямоугольной решетки, т. е. для точки (x, y, z), лежащей на представляемой поверхности, достаточно хранить только координаты (x, y), а третью координату (z) – получать с помощью функции $H(x, y) = z$, заданной на двумерном наборе данных функционально или с помощью таблицы. Это общий тип данных, обычно используемый для представления ландшафтов во многих программных приложениях, включая самолетные тренажеры и тренажеры наземной техники. Используя карту высот, можно абстрагировать модуль создания геометрической модели от характера поверхности, т. е. способов ее создания и управления. Задание поверхности моря с помощью набора высот имеет и ряд недостатков, например, с его помощью нельзя моделировать нависания поверхности (т. е. $H(x, y)$ принимает несколько значений). Для динамических поверхностей, таких как, например, море, вводится третий параметр – время, тогда $z = H(x, y, t)$.

Авторами данной работы реализовано и испытано несколько предложенных в литературе алгоритмов генерации морской поверхности:

Алгоритм 1. Суперпозиция синусов [1].

Алгоритм 2. Метод частотной фильтрации [2].

Алгоритм 3. Метод, предложенный D. R. Peachey [3].

Ниже рассмотрена суть данных алгоритмов, а также их достоинства и недостатки, выявленные авторами в процессе реализации и испытаний.

1.1. *Алгоритм 1. Суперпозиция синусов.* В данном методе морская поверхность задается как суперпозиция нескольких синусоидальных волн с собственной амплитудой A_i, периодом T_i, начальной фазой и направлением распространения $k_i(k_{ix}, k_{iy})$:

$$z(x, y, t) = \sum_{i=1}^{n} A_i \sin\left(k_{ix}x + k_{iy}y - \frac{t - t_0}{T_i} \right).$$

В процессе реализации и тестирования выявлены следующие достоинства:

– простота генерации и анимации (самое существенное достоинство);

– небольшой объем данных, необходимый для работы алгоритма;

– возможность быстрой смены амплитуды, длины, направления и скорости распространения волны.

К недостаткам относится следующее:

– получаемое изображение лишь отдаленно напоминает реальность;

– отсутствует возможность моделирования рефракции;

– получаемая геометрическая модель является регулярной, т. е. неестественной.

Вследствие перечисленных недостатков этот алгоритм использовался лишь на стадии отладки модуля упрощения геометрической модели поверхности.

1.2. *Алгоритм 2. Метод частотной фильтрации белого шума.* В качестве фильтра использован фильтр Пирсона – Московитца (см. [2])

$$F_{PM} = \frac{\alpha g^2}{(2\pi)^4 f^5} \exp\left(-\frac{5}{4}\left(\frac{f_m}{f}\right)^4\right),$$

где $F_{PM}(f)$ – частотный спектр; f – частота; f_m – пиковая частота, $f_m = \frac{0{,}13g}{u_{10}}$ (u_{10} – скорость ветра на высоте 10 м над поверхностью); α – константа Филипса (0,0081); g – гравитационная постоянная.

Модернизированный Хасельманом (см. [2]) для двумерного случая фильтр принимает вид

$$F(f, \theta) = F_{PM} D(g, \theta),$$

где D – весовое распределение, зависящее от θ – угла к вектору направления ветра. На рис. 1 изображен фильтр в пространстве Фурье для скорости ветра 15 м/с, дующего под углом 60°. С использованием этого фильтра получено изображение морской поверхности, представленное на рис. 2.

Для анимации поверхности может быть использовано несколько техник управления фазой в пространстве Фурье.

В процессе реализации и тестирования авторами выявлены следующие достоинства:

– алгоритм генерирует достаточно реалистичную картину;

– имеются широкие возможности анимации;

– получаемая геометрическая модель выглядит «случайно».

К недостаткам относится следующее:

– алгоритм не учитывает формы дна и берега, т. е. предназначен только для генерации поверхности «открытого» моря;

– необходимо оперировать сравнительно большими объемами данных, что ведет к потере производительности;

– алгоритм требует больших вычислительных ресурсов;

– изменение скорости и направления ветра сопровождается значительными вычислениями.

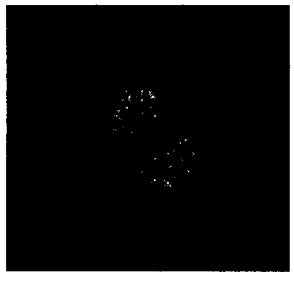

Рис. 1. Частотный фильтр Пирсона–Московитца в пространстве Фурье

Рис. 2. Изображение поверхности «открытого» моря, полученное методом частной фильтрации белого шума

1.3. *Алгоритм 3. Метод D. R. Peachey.* Как сообщалось выше, форма морской поверхности представляется как функция трех переменных: $z(x, y, t) = H(x, y, t)$, где x, y – обычные прямоугольные координаты в модельном пространстве; t – время, изменяемое для каждого кадра анимации. Функция $H(x, y, t)$ есть сумма нескольких «длинных» гребней волн W_i с амплитудами A_i, распространяющихся в различных направлениях:

$$H(x, y, t) = \sum_{i=1}^{n} A_i W_i(x, y, t),$$

где $W_i(x, y, t) = w_i(\text{fraction}[\theta_i(x, y, t)])$, т. е. W_i представляет собой суперпозицию двух функций: w_i – профиль волны (единичная периодическая функция, заданная на промежутке [0, 1]) и $\theta_i(x, y, t) = \theta_i(x, y, t_0) - \frac{t - t_0}{T_i}$ – функция фазы, состоящая из двух компонент (пространственной и временной), где T_i – период i-й волны; $\theta(x, y, t_0)$ – фаза в некоторый момент времени t_0 (пространственная компонента фазы). Пространственная компонента фазы позволяет ввести зависимость от поверхности дна, что делает возможным моделирование таких эффектов, как рефракция волн.

Хорошим приближением для случая, когда период почти не зависит от глубины, является трансцендентное выражение [1]: $k\text{th}(kh) = k_\infty$, где h – глубина в некоторой точке; k – волновое число; k_∞ – волновое число на бесконечной глубине. Это уравнение легко решается для предельных значений h: для малой глубины ($h < 0{,}05L$, где L – период волны) $k = \sqrt{k_\infty / h}$, для большой глубины ($h > 0{,}25L$) $k = \frac{k_\infty}{\sqrt{\text{th}(k_i h)}}$. Соответственно находим пространственную компоненту фазы:

$$\theta_i(x_i) = \int_0^{\bar{x}_i} k_i(u)du = \sum_0^{\bar{x}_i} \frac{k_\infty}{\sqrt{\text{th}(k_\infty h(x_i))}} \Delta x,$$

где осуществлен переход к численному интегрированию. (Следует отметить, что разделение фазы на пространственную и временную компоненты позволяет выполнять численное интегрирование для данной формы дна, длины волны и направления распространения только один раз, а не отдельно на каждом кадре.)

Профиль волны – $w_i(u)$ – некоторая функция, определенная на $u \in [0,1)$, с множеством значений в интервале $[-1, 1]$, которая задает форму волны. Для сохранения непрерывности необходимо, чтобы $w_i(1) = w_i(0)$. Алгоритм использует два вида профилей: $w_i = \cos(2\pi u)$ – синусоидальный для «глубокой воды», $w_i = 8(u - 0{,}5)^2 - 1$ – более острый для прибрежных волн.

Для достижения большего реализма осуществляется линейная интерполяция между двумя этими профилями в зависимости от глубины и длины волны. На рис. 3 приведено изображение пляжа с набегающими на него волнами, причем береговая линия представлена ломаной, а глубина меняется линейно с увеличением расстояния от берега.

В процессе реализации и тестирования выявлены следующие достоинства:

– генерируемая геометрическая модель выглядит реалистично;

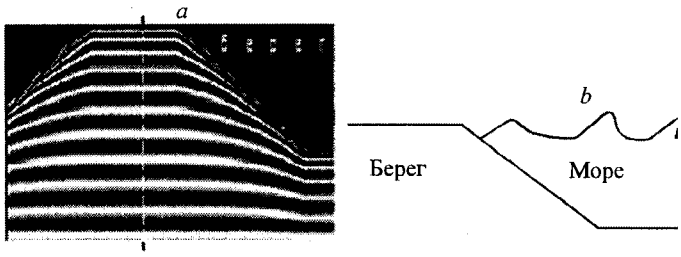

Рис. 3. Берег с линейно изменяющейся глубиной и набегающие на берег волны (*a* – вид сверху, *b* – срез). Хорошо видна рефракция волн

– простота анимации;

– алгоритм учитывает форму дна и берега.

Недостатки следующие:

– смена длины и направления распространения волны сопровождается сложными вычислениями;

– необходимо манипулировать сравнительно большими объемами данных, что в большинстве случаев ведет к потере производительности;

– получаемая поверхность излишне регулярна.

1.4. *Выводы.* При реализации и тестировании вышеперечисленных алгоритмов установлено, что все алгоритмы обеспечивают быстрый пространственный поиск и позволяют генерировать поверхность с различными уровнями детализации; алгоритмы 1 и 3 дают возможность привязать текстурные координаты к создаваемым моделям. Модели, создаваемые алгоритмами 1 и 3, выглядят регулярно (впрочем, алгоритму 3 это присуще в меньшей степени), тогда как поверхность, созданная алгоритмом 2, выглядела довольно «случайно». Алгоритмы 2 и 3 предъявляют высокие требования к скорости выполнения операций с плавающей запятой.

Проведенный анализ показал, что метод Peachey (алгоритм 3), несмотря на присущие ему недостатки, обладает наилучшими визуальными качествами, наряду с приемлемой производительностью, что и позволило выбрать его базовым вариантом для использования в качестве подсистемы генерации и анимации формы водной поверхности.

2. **Оптимальная триангуляция морской поверхности в зависимости от положения наблюдателя.** При разработке алгоритмов учитывалась необходимость в реальном времени генерировать и визуализировать большие площади морской поверхности, поэтому разработана подсистема управления детализацией в зависимости от положения камеры и ее ориентации. Морская поверхность в силу своей природы не разбивается на блоки, детализация которых может настраиваться независимо. Поверхность должна остаться неразрывной, при этом та часть поверхности, которая видна и находится вблизи от наблюдателя, должна представляться большим количеством треугольников.

Подсистема управления детализацией должна удовлетворять нескольким формальным требованиям [4].

1. Триангуляция минимально избыточна, т. е. важные с точки зрения текущего кадра части морской поверхности должны быть представлены большим количеством треугольников.

2. В любом состоянии геометрическая модель и ее компоненты, такие как вершины и треугольники, легко доступны для других частей приложения,

например для подсистемы проверки столкновений. Также необходим быстрый пространственный поиск вершин и треугольников.

3. Динамические изменения сетки, влекущие за собой изменение параметров поверхности или ее геометрии, незначительно влияют на производительность.

4. Высокочастотные особенности поверхности, такие как локальные выпуклости или вогнутости, не ведут к глобальному усложнению модели.

5. Небольшие изменения положения наблюдателя не ведут к значительному изменению геометрии модели во избежание значительных изменений получаемого изображения (кадра от кадра) и для сохранения скорости генерации одного кадра почти постоянной.

6. Алгоритм имеет возможность заранее задавать сложность получаемой модели (т. е. управлять изменениями качества).

Если по данным, предоставляемым подсистемой, задающей форму морской поверхности, строить геометрическую модель, используя равномерную триангуляцию, то получаемая геометрическая модель не отвечает требованиям минимальной избыточности. С помощью подсистемы, производящей упрощения геометрической модели, можно значительно уменьшить количество треугольников в сцене без значительной потери визуального качества, что в результате позволяет уменьшить общее время, уходящее на обработку одного кадра.

На рис. 4 представлена равномерная триангуляция сетки 9×9, содержащая 256 треугольников. На рис. 5 показана сетка после упрощения (более чем в 2 раза сокращено общее количество треугольников в сцене за счет уменьшения количества треугольников, не попадающих в пирамиду видимости, и количество «дальних» треугольников).

В литературе предложено несколько техник для управления уровнем детализации. В работе [5] предложен алгоритм, использующий изменяющиеся сетки. Это относительно новая и хорошая техника для добавления треугольников в нерегулярную сетку при изменении детализации сцены, но, к сожалению, она излишне сложна и требует значительного количества оперативной памяти. В [6] представлена структура данных (четвертичное дерево), используемая для представления частей поверхности. Четвертичное дерево рекурсивно разбивается на треугольники для создания подходящей аппроксимации карты высот. Это очень простая и эффективная техника. Однако в качестве основы для подсистемы управления уровнем детализации использовался предложенный в работе [7] алгоритм, имеющий много общего с чет-

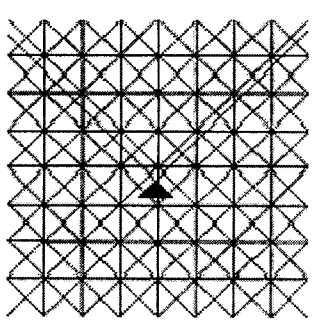

Рис. 4. Равномерная триангуляция (256 треугольников)

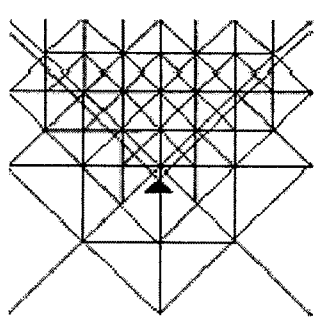

Рис. 5. Упрощенная триангуляция (всего 100 треугольников)

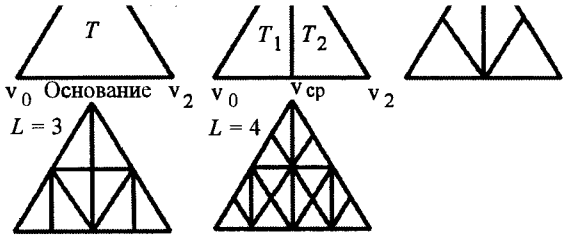

Рис. 6. Уровни $L = 0-4$ бинарного дерева треугольников

вертичным деревом, но более простой и гибкий. Рассмотрим его более детально.

2.1. *Представление геометрической модели.* Основной структурой, используемой модулем управления детализацией, является двоичное дерево треугольников. На рис. 6 показано несколько первых уровней этого дерева.

Корнем дерева является треугольник $T = (v_1, v_0, v_2)$, это самый низкий, нулевой уровень детализации. Для любого треугольника введем обозначения: (v_1, v_0) – левое ребро, (v_1, v_2) – правое ребро, (v_0, v_2) – ребро основания. Соответственно вводятся обозначения для соседних треугольников: правый, левый и сосед основания. Следует заметить, что у треугольника с уровнем детализации L могут быть соседи только уровней L, $L+1$, $L-1$. Следующий уровень дерева получается, как показано на рисунке, разбиением треугольника T на два новых треугольника-потомка: $T_1 = (v_0, v_{ср}, v_1)$ и $T_2 = (v_2, v_{ср}, v_1)$ – соответственно с уровнем детализации $L = 1$. Остальные уровни получаются рекурсивно. Геометрическую модель поверхности моря составляют только треугольники, соответствующие самым нижним узлам двоичного дерева треугольников, которые не имеют потомков (рис. 7).

Замечание. С каждым узлом дерева можно хранить «оболочки» соответствующего ему треугольника, что позволит существенно ускорить следующие операции: проверку пересечения других объектов виртуальной среды с морской поверхностью, пространственный поиск треугольников, вершин и ребер, отсечение треугольников, не попадающих в пирамиду видимости.

2.2. *Динамические преобразования геометрической модели.* Введем операции Split и Merge для модификации бинарного дерева:

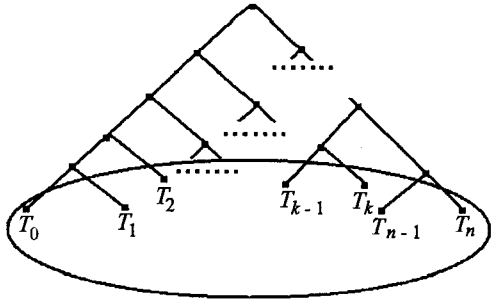

Рис. 7. Двоичное дерево треугольников. В рамку заключены треугольники, составляющие геометрическую модель

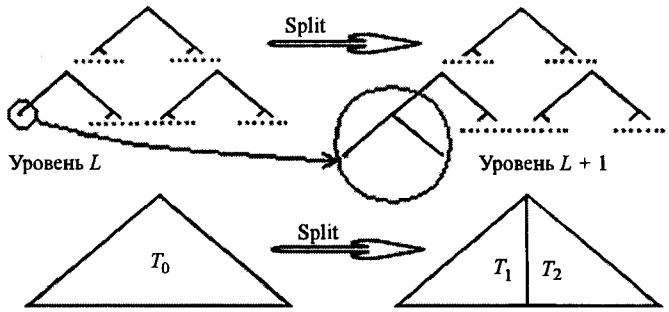

Рис. 8. Операция Split

Split: операция по разбиению некоторой висячей вершины двоичного дерева некоторого уровня детализации L и соответствующего ей треугольника T_0 на две новые вершины уровня $L+1$ и соответствующие им треугольники T_1 и T_2 (рис. 8).

В результате появляется новая вершина. Если разбиваемый треугольник T_0 имел соседа по основанию – T_E, то для избежания разрывов и нестыковок (T-vertices) мы вынуждены применить операцию Split к треугольнику T_E. Такой процесс называется Forced Split, и он может затрагивать несколько треугольников (рис. 9).

Замечание. Глубина Forced Split ограничена максимальной глубиной дерева треугольников (которое, в свою очередь, ограничено разрешением карты высот, на которой строится дерево). Допустим, что разбиваемый треугольник T_0 принадлежит уровню L бинарного дерева. Тогда его сосед T_E, к которому мы вынуждены для сохранения непрерывности применить операцию Split, может лежать на уровне L или $L-1$. В случае когда уровень равен $L-1$, придется повторять операцию для еще одного треугольника, а в случае уровня L процесс разбиения заканчивается.

Merge. Введем новую геометрическую структуру «ромб» (T_L, T_R) (рис. 10), состоящий из двух узлов двоичного дерева треугольников (соответствующие им треугольники – T_L, T_R), имеющих одинаковый уровень детализации и таких, что, во-первых, их дети не имеют потомков, т. е. соответствующие детям треугольники (T_0, T_1, T_3, T_4) содержатся в геометрической модели, и, во-вторых, треугольники T_0, T_1, T_3, T_4 образуют ромб со сторонами из ребер основания соответствующих треугольников. Операция Merge уничтожает узлы T_0, T_1, T_3, T_4 бинарного дерева, и соответствующие им треугольники удаляются из геометрической модели, а вместо них появляются треугольники T_R и T_L с более низким уровнем детализации.

Итак, мы имеем операции, позволяющие гибко изменять уровень детализации сцены. Рассмотрим алгоритм, который управляет процессом разбие-

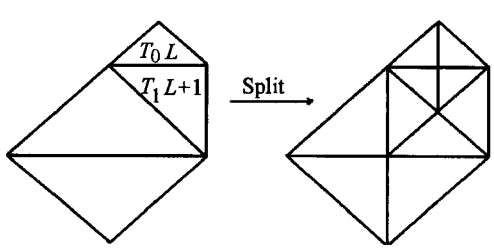

Рис. 9. Разбиение одного треугольника, приводящее к появлению десяти новых

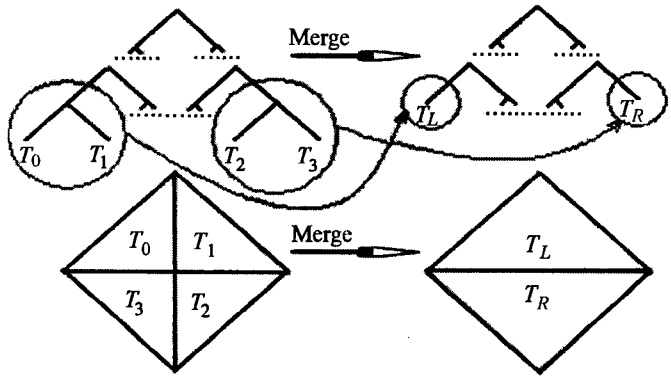

Рис. 10. Операция Merge

ния и слияния треугольников для получения наиболее подходящей степени детализации. Присвоим каждому треугольнику T в сцене некоторый приоритет $P(T) \in [0,1]$и, начиная с начальной триангуляции, будем производить операцию Forced Split для наиболее приоритетного треугольника. Как будет показано далее, такой «жадный» алгоритм строит последовательность триангуляций, которая при условии, что приоритеты назначаются монотонно (т. е. приоритет потомка при разбиении меньше приоритета родителя), минимизирует максимальный приоритет. Если, например, взять в качестве приоритета величину, обратно пропорциональную расстоянию до геометрического центра данного треугольника, а треугольникам, не попадающим в пирамиду видимости, присвоить нулевой, т. е. минимальный приоритет, то в первую очередь операция Forced Split будет применяться к треугольникам, находящимся в пирамиде видимости и наиболее близким к камере, т. е. для заданного количества треугольников в итоге получим в некотором смысле оптимальную триангуляцию (чем ближе к камере, тем детализация больше). Задав приоритеты другим способом, получим сцену, оптимальную по некоторым другим параметрам, т. е. критерием оптимальности триангуляции служит минимальность максимального приоритета (более подробно выбор приоритетов описан в п. 2.5). Операция Merge позволяет «жадному» алгоритму продолжать работу при изменении приоритетов в сцене с предыдущей оптимальной триангуляцией и, таким образом, пользоваться преимуществами покадровой когерентности.

2.3. *Алгоритм, производящий оптимальную триангуляцию с наперед заданным количеством треугольников.* Предположим, что с каждым треугольником в триангуляции **T** сопоставлен монотонный приоритет $P(T) \in [0,1]$.

Алгоритм I.

Пусть T – начальная триангуляция;
Сопоставим с каждым $T \subseteq$ **T** приоритет $P(T) \in [0, 1]$;
Пока (количество треугольников меньше заданного или **T** недостаточной точности) {
Определить треугольник T с максимальным приоритетом в триангуляции **T**;
Forced Split T;
}

Так как глубина Forced Split ограничена максимальной глубиной двоичного дерева треугольников, то количество операций Split, производимых ал-

горитмом, пропорционально количеству треугольников N в конечной триангуляции. Для ускорения процесса поиска имеет смысл поддерживать очередь Q_s из треугольников, составляющих текущую геометрическую модель и отсортированных в порядке убывания приоритетов. Докажем, что данный алгоритм генерирует оптимальную триангуляцию с минимальным, максимальным приоритетом на каждом шаге. Рассмотрим любую другую триангуляцию T', которая имеет более низкий максимальный приоритет, чем T. Очевидно, что T' должна содержать только потомков всех треугольников, к которым была применена операция Forced Split при формировании T. Поскольку операция Forced Split делает только минимальные необходимые изменения, чтобы сохранить непрерывность, то T' не может содержать никаких предков к треугольникам из T. Наконец, поскольку T' имеет более низкий приоритет, она должна содержать только потомков, по крайней мере, одного из треугольников T. Поэтому триангуляция T' должна содержать больше треугольников, чем T, значит, T – оптимальная триангуляция.

2.4. *Модификация алгоритма I для случая изменяющихся приоритетов.* Пусть теперь для каждого кадра $f \in N$ с каждым треугольником в триангуляции **T** сопоставлен изменяющийся во времени приоритет $P_f(T) \in [0,1]$. Задача – получить последовательность триангуляций $(T_0, T_1, T_2, \ldots)$, для каждой из которых максимальный приоритет треугольников сцены минимален. Если приоритеты кадр от кадра меняются медленно, то любые две последовательные триангуляции T_f и T_{f-1} оказываются подобными друг другу. Таким образом, для значительного ускорения работы алгоритма I для последовательности кадров триангуляцию T_f лучше получать из триангуляции T_{f-1}. Это достигается путем создания и управления второй очередью приоритетов Q_m, которая содержит в себе все «ромбы» $D = (T_L, T_R)$ в текущей триангуляции, упорядоченные по приоритетам, которые для некоторого ромба (T_L, T_R) вычисляются по правилу $P_f(D) = \max(P_f(T_L), P_f(T_R))$.

Алгоритм II.

Если ($f = 0$) {
    Очистить Q_s и Q_m;
    Положить **T** – начальная триангуляция;
    Добавить в Q_s все треугольники из **T** и в Q_m все «ромбы» в **T**;
    Вычислить все приоритеты для Q_s и Q_m;
} **иначе** {
    T = $\mathbf{T}_f$;
    Обновить приоритеты в Q_s и Q_m;
}
Пока (количество треугольников меньше заданного, или **T** недостаточной точности, или максимальный приоритет треугольников в **T** больше, чем минимальный приоритет «ромбов») {
    если (**T** содержит слишком много треугольников) {
        Определить минимальный по приоритету «ромб» D в **T**;
        Merge D;
    } **иначе** {
        Определить максимальный по приоритету треугольник T в **T**;
        Forced Split T;
    }
}
Присвоить $T_f = T$;

Общее количество операций Split и Merge, выполняемое алгоритмом II при генерации одного кадра, пропорционально ΔN – количеству отличающихся треугольников в T_f и T_{f-1}. Алгоритм II производит триангуляцию T_f, имеющую тот же приоритет, что и триангуляция, получаемая алгоритмом I, примененным к начальной триангуляции, но вследствие межкадровой когерентности достигает этого за меньшее количество операций Split и Merge.

2.5. *Приоритеты*. В настоящее время существует много способов выбора для некоторого блока геометрической модели соответствующего уровня детализации в зависимости от расстояния до него, ориентации и геометрических особенностей упрощаемой поверхности и т. д. [4, 6, 7].

Так как поверхность, к которой применяется данный алгоритм, достаточно однородная (морская поверхность), то в качестве приоритета для некоторого треугольника берется величина, обратно пропорциональная расстоянию до геометрического центра данного треугольника, а треугольникам, не попадающим в пирамиду видимости, присваивается нулевой, т. е. минимальный приоритет. Таким образом, в первую очередь разбиваются наиболее близкие к камере попадающие в пирамиду видимости треугольники. Существуют и другие разновидности измерения приоритетов, которые можно использовать с данным алгоритмом:

– «*Уменьшение детализации задних треугольников*». Сохраняя вместе с двоичным деревом треугольников разброс нормалей к граням, можно принудительно уменьшать приоритет треугольников того поддерева, которое состоит целиком из задних треугольников.

– «*Искажения нормалей*». Для поверхностей, освещение которых задается нормалями, интерполированными по вершинам, для правильного освещения необходимо увеличить приоритет треугольникам, имеющим больший разброс нормалей в вершинах.

– «*Силуэты*». Для правильного получения силуэтов необходимо увеличивать приоритеты треугольникам, находящимся на краю поверхности.

– «*Атмосферная видимость*». Приоритеты могут быть снижены в условиях плохой видимости (во время тумана, дождя или ночью).

– «*Позиционирование объектов*». Чтобы корректно поставить некоторый объект на поверхность, приоритеты треугольников под каждым из устанавливаемых объектов должны быть искусственно увеличены.

2.6. *Выводы*. Перечисленные алгоритмы программно реализованы в виде встраиваемого модуля для генерации морской поверхности в реальном времени. На ЭВМ с процессором Pentium II, 300 MHz 128 Mb RAM при движении камеры на небольшой высоте над поверхностью со средней скоростью на процесс генерации одного кадра, содержащего 1500 треугольников, в который входят генерация и упрощение геометрической модели, вычисление и сглаживание нормалей, задание текстурных координат и подготовка к выводу изображения средствами популярного графического API фирмы "Microsoft-DirectX" [8], в среднем уходит около 9 мс, из них только около 3 мс занимает работа подсистемы управления уровнем детализации.

Для ускорения работы алгоритма использовались следующие оптимизации.

– Создание и управление очередями треугольников и ромбов, отсортированных соответственно в порядке возрастания и убывания приоритетов, позволяющих осуществлять быстрый поиск треугольников с максимальным и ромбов с минимальным приоритетами, используемых в работе описанных алгоритмов.

– Техника отложенных вычислений:

1. Вычисление приоритетов ко всем треугольникам в сцене на каждом кадре требует значительных вычислительных затрат; пользуясь межкадровой когерентностью сцены (в случае плавного передвижения камеры приоритеты треугольников кадр от кадра меняются мало), это время можно уменьшить. Для этого поддерживается очередь, элементами которой являются группы треугольников, и на каждом кадре вычисляются приоритеты треугольников только из текущей группы, а затем статус текущей передается следующей группе треугольников.

2. Алгоритм II не гарантирует постоянство времени вычислений, но так как он может начинать свою работу с произвольной триангуляции, по истечении некоторого определенного времени работы алгоритм может быть прерван. Конечно, полученная триангуляция не будет оптимальной для данного кадра (положения камеры), однако благодаря тому, что шаги разбиения/слияния выполняются в уменьшающемся порядке важности, частично сделанная работа оптимальна в том смысле, что триангуляция стала настолько близка к оптимальной, насколько позволило время. Техника отложенных вычислений позволяет поддерживать скорость генерации одного кадра почти постоянной.

– Для еще большего ускорения работы алгоритма предусмотрена возможность отказаться от анимации треугольников заднего плана.

– Оптимизация программ для процессоров Pentium II и Pentium III [9].

В результате проведенных оптимизаций скорость работы модуля выбора уровня детализации уменьшилась, а алгоритм получил возможность при заданном количестве треугольников в сцене менять время генерации данных для визуализации одного кадра, тем самым освобождая время для визуализации остальных частей системы визуализации виртуальной реальности.

3. **Техника генерации и анимации текстур.** Если рассматривать примеры различных природных материалов и явлений, таких как горы, галька и песок, камни, облака, трава в поле, волны на воде, огонь, движение муравьев, рисунок на мраморе, ветви деревьев, то все они обладают двумя общими («фрактальными») свойствами:

1) свойство «самоподобия», т. е. при различном увеличении объекты выглядят почти одинаково (подобно);

2) объекты задаются регулярной структурой с наложением на нее некоторой «случайности».

Для придания подобной «случайности» работе программы обычно используется датчик случайных чисел, но, к сожалению, обычные датчики случайных чисел являются слишком грубыми (резкими) и не годятся для таких целей, кроме того, они не обладают свойством «самоподобия».

Для анимации текстурных координат некоторой искусственной текстуры, представляющей поведение поверхности воды, и для придания естественности морскому дну может применяться шумовая функция Перлина (см. [10, 11]).

Шумовая функция представляет собой в общем случае датчик псевдослучайных чисел, который в качестве входного параметра получает некоторое число, а на выходе выдает псевдослучайное число, причем, что важно, если использовать одинаковый параметр дважды, то мы получим дважды одно и то же число. На рис. 11 представлены результаты работы некоторого дискретного датчика псевдослучайных чисел. Чтобы перейти от дискретного датчика к непрерывному, можно использовать любую устраивающую нас

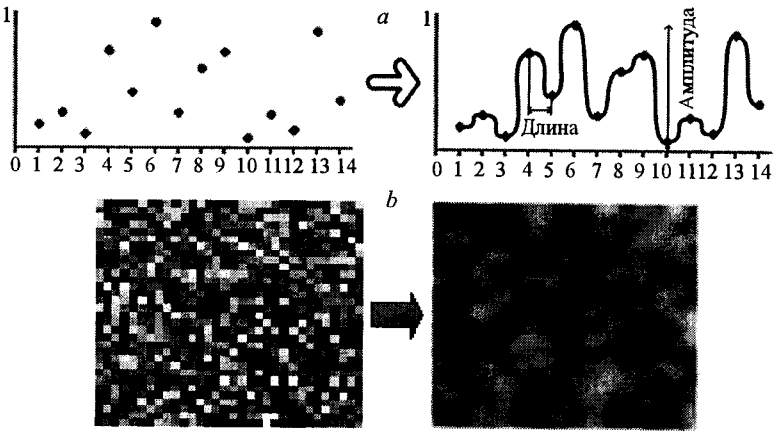

Рис. 11. Переход от дискретного датчика случайных чисел к непрерывному с использованием интерполяции косинусами для одномерного случая (*a*), для двумерного случая (*b*)

по производительности и качеству интерполяцию (линейную, кубическую, интерполяцию косинусами и пр.).

Введем некоторые определения: амплитуда шума – расстояние между максимальным и минимальным значениями шумовой функции; частота – величина, обратная «длине волны», т. е. расстояние между двумя точками в множестве, на котором задан датчик случайных чисел.

Шумовая функция Перлина определяется следующим способом: $\mathrm{PerlinNoise}(x) = \Sigma(\mathrm{persistence})^i \mathrm{Noise}_i(2^i x)$, где $\mathrm{Noise}_i(x)$ – шумовая функция, частота и амплитуда которой соответственно равны $\mathrm{Frequency} = 2^i$ и $\mathrm{Amplitude} = \mathrm{persistence}^i$, где persistence – некоторая константа из отрезка [0, 1].

Каждая из следующих одна за другой шумовых функций называется октавой, так как каждая следующая имеет в 2 раза большую частоту. На рис. 12 показаны последовательность из шести октав и результат их сложения (обычно количество октав берется равным $\log_2(\mathrm{Resolution}) - 2$, где Resolution, например, разрешение текстуры).

Теперь, имея гладкую шумовую функцию, обладающую свойствами самоподобия, можно использовать ее следующим образом:

– начинаем с некоторой регулярной структуры $V(x)$;

– применяем шумовую функцию Перлина к данной структуре как $V(x) = V(x) + \mathrm{PerlinNoise}(x)$ или $V(x) = V(x + \mathrm{PerlinNoise}(x))$;

– выбираем область определения шумовой функции Перлина для того, чтобы удовлетворять взятой структуре;

– подбираем подходящее количество октав (добавление одной новой ведет к замедлению работы алгоритма), размерность (например, $\mathrm{PerlinNoise3}D(u, v, t)$, где u и v – текстурные координаты; t – время, используемое для анимации) и амплитуду.

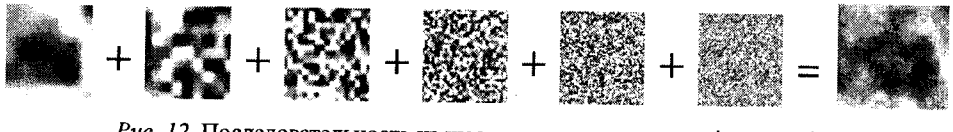

Рис. 12. Последовательность из шести октав, взятых с persistence = 0,5

Приведем пример использования шумовой функции Перлина для анимации координат текстуры, накладываемой на морскую поверхность: в качестве регулярной структуры выбрана равномерная сетка: $u(i,j,t)=i$, $\mathrm{v}(i,j,t)=j$, затем к равномерной сетке шумовая функция Перлина применяется следующим образом:

$$u(i,j,t)=i+\cos(\mathrm{PerlinNoise3}D(i,j,t)),$$

$$\mathrm{v}(i,j,t)=j+\sin(\mathrm{PerlinNoise3}D(i,j,t)).$$

Для генерации псевдослучайных чисел ($\mathrm{PerlinNoise3}D(i,j,k)$) использовался следующий алгоритм [10]. Пусть есть массив $g[n]$ предварительно сгенерированных псевдослучайных чисел $[-1,1]$ и случайная перестановка $p[]$ массива $g[]$, тогда $\mathrm{PerlinNoise3}D(i,j,k)=g[(p[(p[i]+j)\,\%n]+k)\,\%n]$, вычисления можно ускорить путем расширения массивов $g[]$ и $p[]$ так, чтобы $g[i]=g[i+n]$ и $p[i]=p[i+n]$, тогда $\mathrm{PerlinNoise3}D(i,j,k)=g[p[p[i]+j]+k]$. После этого полученный шум интерполируется.

Функция шума Перлина – мощный инструмент для придания виртуальному миру большей естественности, причем речь идет не только о процедурных текстурах. Область ее применения значительно шире. Например, функцию Перлина можно применять для «оживления» движения цифровых персонажей по виртуальному миру, генерации искусственных ландшафтов, создания и анимации $3D$ облаков и т. д. При реализации программы функция Перлина использовалась для генерации поверхности морского дна и анимации координат текстуры, накладываемой на морскую поверхность. Использование функции Перлина позволило генерировать более реалистичную морскую поверхность.

Заключение. Авторами данной работы программно реализованы все перечисленные модели поведения и алгоритмы генерации геометрической модели морской поверхности, а также алгоритмы анимации и генерации текстуры. Использованные алгоритмы оптимизированы для получения наибольшего быстродействия, и в случае возможности выбора реализации взяты те способы, которые обеспечивают наилучшее визуальное качество, наряду с хорошей производительностью. На основе проведенных исследований реализован встраиваемый модуль для визуализации морской поверхности, который позволяет пользователю варьировать различные параметры, влияющие на визуальное качество получаемых изображений и скорость генерации одного кадра (количество треугольников в сцене, степень детализации, время работы подсистемы упрощения геометрической модели и пр.), а также регулировать такие «физические» параметры, как скорость и направление ветра, рельеф морского дна и линия берега, ширина полосы прибоя и пр. Для тестирования встраиваемого модуля написана программа, визуализирующая кусок океанской поверхности 3000×2000 м (рис. 13). При тестировании на ЭВМ Pentium II (400 MHz; RAM – 128 Mb; Intel 1740 video card, 4 Mb video RAM), работающей под управлением операционной системы Windows 98, программа тратила на генерацию одного кадра изображения анимированной морской поверхности около 17 мс (около 55 кадров в секунду), причем около 9 мс уходило на создание физической и соответствующей ей геометрической модели поверхности, состоящей из 1500 треугольников, анимацию текстурных координат и подготовку вывода изображения средствами DirectX, т. е. на работу встраиваемого модуля. Такой скорости в большинстве случаев оказы-

Рис. 13. Пример визуализации морской поверхности

вается достаточно для эффективного применения библиотеки в системах синтеза визуальной обстановки.

СПИСОК ЛИТЕРАТУРЫ

1. **Fournier A., Reeves W. T.** A simple model of ocean waves // Computer Graphics: Proc. SIGGRAPH'86. 1986. P. 75.
2. **Mastyn G. A., Watterberg P., Mareda J.** Fourier synthesis of ocean scenes // IEEE Computer Graphics and Applications Mar. 1987. P. 16.
3. **Peachey D. R.** Modeling waves and surfaces // Computer Graphics: Proc. SIGGRAPH'86. 1986. P. 65.
4. **Lindstrom P., Koller D., Ribarsly W. et al.** Real-time, continuous level of detail rendering of height fields // SIGGRAPH'96.
5. **Hoppe H.** Smooth view-dependent level-of-detail control and its application to terrain rendering // Microsoft Research.
6. **Lindstrom P., Koller D., Hodges L. F. et al.** Level-of-Detail Management for Real-time Rendering of Photextured Terrain // Graphics, Visualization and Usability Center, Georgia Tech TR-95-06 1995.
7. **Duchaineau M., Wolinsky M., Sigeti D. E. et al.** ROAMing terrain: Real-time optimally adapting meshes // IEEE Visualization. 1997. P. 81.
8. **Microsoft** DirectX 6.0 Programmer's Reference // Microsoft Corporations © 1998.
9. **Auerbach M., Hagege A., Lederer E. et al.** Оптимизация программ для процессоров Pentium II и Pentium Pro // Intel Corporation ©. 1996–1998.
10. **Using** MMX™ Instructions for Procedural Texture Mapping (Based on Perlin's Noise Function) // Intel Developer Relations Group © 1996.
11. **Goehring D., Gerlitz O.** Advanced Procedural Texturing Using MMX™ Technology // Intel Corporation © 1997.

Институт автоматики и электрометрии СО РАН,
E-mail: Nicolas@sl.iae.nsk.su

Поступила в редакцию
29 августа 2000 г.