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

«СофтЛаб-НСК», Новосибирск

Методы непрерывной детализации террэйна.

1 АННОТАЦИЯ

В работе представлен и проанализирован ряд алгоритмов упрощения геометрической модели земной поверхности в зависимости от положения и ориентации наблюдателя, а так же выдвинуты требования, которым должна удовлетворять подсистема визуализации рельефа земной поверхности. На основе проведенного анализа алгоритмов выбран алгоритм *ROAM*, как наиболее полно удовлетворяющий выдвинутым требованиям и предложена модернизация алгоритма, позволяющая существенно уменьшить время генерации геометрической модели в случае малокогерентных кадров.

Ключевые слова: *terrain, continuous level of details, surface simplification view-dependent mesh, frame-to-frame coherence.*

1. ВВЕДЕНИЕ.

Современные промышленные графические ускорители для персональных компьютеров позволяют решать задачи, требующие генерации изображения визуальной обстановки фотографического качества в реальном времени. Высококачественная визуализация рельефа земной поверхности является исходным требованием при создании большинства тренажеров наземного и летного транспорта а также очень многих приложений виртуальной реальности.

Авторы статьи поставили перед собой задачу разработать алгоритм, наиболее адекватный в настоящий момент данной проблеме на современной аппаратной базе. Было принято решение разработать ряд требований и на их базе провести сравнительный анализ существующих алгоритмов. Результатом анализа должен был стать либо выбор наиболее адекватного, либо разработка абсолютно нового алгоритма. Вторым результатом работы должна была стать реализация алгоритма в виде программной подсистемы. При разработке подсистемы авторы поставили перед собой задачу создания «масштабируемой» подсистемы, т.е. удовлетворительно работающей на станциях с различными возможностями, как центрального процессора, так и графического ускорителя.

В данной работе рассмотрены только методы управления детализацией геометрической модели. Хотя при создании подсистемы визуализации авторами и было реализовано текстурирование земной поверхности методом особенностей, способы текстурирования в данной работе не рассматриваются.

2 ОБЗОР АЛГОРИТМОВ ГЕНЕРАЦИИ ЗЕМНОЙ ПОВЕРХНОСТИ В РЕАЛЬНОМ ВРЕМЕНИ ОСНОВАННЫХ НА ПРИНЦИПАХ НЕПРЕРЫВНОЙ ДЕТАЛИЗАЦИИ.

На подсистему управления детализацией можно наложить несколько формальных требований [1, 2]:

- Триангуляция должна удовлетворять требованию минимальной избыточности, т.е. важные с точки зрения текущего кадра части рельефа должны быть представлены большим количеством треугольников.
- Алгоритм должен избегать или сводить к минимуму разрывы в геометрии и в освещении земной поверхности.
- Высокочастотные особенности поверхности, такие как локальные выпуклости или вогнутости, не должны вести к глобальному усложнению модели.
- Алгоритм должен поддерживать устойчивую скорость генерации одного кадра.
- Алгоритм должен предоставить возможность создавать длинные последовательности *triangle strips* и *triangle fans* (*triangle strip* – последовательность из трех вершин, в которой каждая вершина, исключая первые три, задает новый треугольник, *triangle fan* – подобны *triangle strip*, только все треугольники имеют одну общую вершину [3]), использование которых существенно сокращает нагрузку на шину, что позволяет ускорить работу системы.
- Для сохранения скорости визуализации одного кадра постоянной и во избежание визуальных артефактов, небольшие изменения положения наблюдателя не должны вести к значительному изменению геометрии модели.
- Алгоритм должен предоставить возможность заранее задавать сложность получаемой модели (т.е. иметь возможность управлять производительностью подсистемы).

Первые два свойства отвечают за визуальное качество, а остальные за производительность и масштабируемость подсистемы. Наиболее важным требованием к подсистеме генерации рельефа земной поверхности является наличие динамической детализации геометрической модели, т.е. выбора уровня детализации на основе положения наблюдателя и особенностей рельефа.

В следующих главах будет рассмотрен ряд методов управления детализацией геометрической модели рельефа земной поверхности, а так же их достоинства и недостатки.

3 АЛГОРИТМ, ОСНОВАННЫЙ НА МЕТОДЕ РЕГУЛЯРНОГО ПРОРЕЖИВАНИЯ.

В настоящее время наличие аппаратного ускорения компьютерной графики стало стандартом в индустрии. Поэтому необходимы алгоритмы, которые наиболее подходят для аппаратного ускорения. Современные 3D ускорители способны обрабатывать и отображать большое количество примитивов, таким образом, алгоритмы основанные на упрощении треугольников проигрывают алгоритмам основанных на упрощении сущностей более высокого уровня. Рассмотрим один из таких алгоритмов основанных на методе метод регулярного прореживания.

3.1 Представление Геометрической Модели

В данном методе рельеф представляется в виде равномерной решетки, в узлах которой находятся вершины представляющие геометрическую модель. Такое представление модели просто в реализации и требует минимальных затрат памяти. Расстояние между узлами решетки постоянно, а размер решетки *N*x*N*, где *N* [1,∞], таким образом, получается (*N – 1*)*(*N - 1*) четырехугольников, каждый из которых в свою очередь состоит из двух треугольников, которые и задают геометрическую модель земной поверхности. Далее решетка разбивается на блоки размером *M*x*M* (например, 256 x 256). В каждом из блоков можно хранить охватывающую сферу для отбраковки блоков, не попадающих в пирамиду видимости.

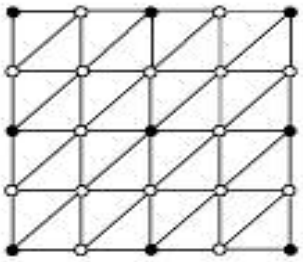

Рис 1. Пример построения первого уровня детализации для блока размером 5x5. Черным выделены вершины, попавшие в первый уровень детализации.

Блоки находящиеся вдали от наблюдателя необходимо отображать с более низким уровнем детализации, чем блоки находящиеся вблизи. Для получения блока с более низкой детализацией исключим из первоначального блока (уровень детализации 0) каждый второй столбец и строку, в результате будем иметь блок с более низкой детализацией (уровень детализации 1). Каждый следующий уровень детализации получается из предыдущего подобным образом. Так можно создать полную последовательность уровней детализации для каждого из блоков задающих геометрическую модель рельефа. На рисунке 1 представлен первый уровень детализации блока 5x5 вершин, заметим, что так же может быть получен и второй уровень детализации.

3.2 Использование метода регулярного прореживания для отображения земной поверхности.

При отображении земной поверхности на основе информации о видимости данного участка земной поверхности, принимается решение об отображении блока и на основе ошибки в экранном или объектном пространстве и положении наблюдателя выбирается тот или иной уровень детализации для данного блока. Когда два соседних блока имеют различный уровень детализации, то на границах блока могут появляться различного рода нестыковки геометрии. Для решения данной проблемы можно вставлять дополнительные вершины вдоль границы таких блоков, но это ведет к тому, что придется динамически изменять

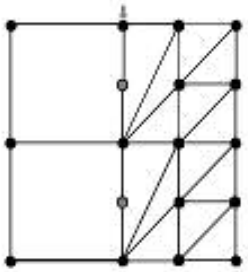

Рис 2. Пример стыковки блоков с разным уровнем детализации. Стрелочкой показана граница блока. Слева находится блок с более низкой детализацией. Серым выделены вершины, которые нарушают стыковку блоков.

вершины и их соединения для данного уровня детализации и делать это каждый раз, когда соседний блок будет менять свою детализацию, что приведет к падению производительности. В [4] предложен оригинальный алгоритм для решения данной проблемы. На рисунке 2 показан стык двух блоков с разной детализацией (граница блоков указана стрелочкой), вершины которые выделены серым цветом отсутствуют в левом блоке, и поэтому в этом месте возникает нестыковка. Что бы этого избежать, соединение вершин в блоке с более высоким разрешением изменяются, так что бы исключить “проблемные” вершины (вершины в которых возникает нестыковка) из геометрической модели, тем не менее, оставляя их в самом блоке. Наиболее быстрый способ сделать это рисовать пограничные вершины, объединяя их в *triangle fan*, а остальные, рисуя обычным путем.

3.3 Достоинства и недостатки алгоритма регулярного прореживания.

В процессе реализации и тестирования данного алгоритма авторами были выделены следующие достоинства и недостатки:

Достоинства:

- Алгоритм отображает различные районы земной поверхности с различным уровнем детализации.

- Алгоритм поддерживает устойчивую скорость генерации одного кадра.
- Алгоритм автоматически поддерживает искусственную ликвидацию разрывов между блоками с различной детализацией.
- На основе охватывающих параллелепипедов, алгоритм отбраковывает невидимые блоки.
- Алгоритм поддерживает генерацию больших последовательностей *triangle strip*.
- Алгоритм поддерживает динамические изменения качеством получаемой геометрической модели.

Недостатки:

- Т.к. алгоритм принимает решения о детализации целого блока, то триангуляция не полностью удовлетворяет требованиям минимальной избыточностности, т.е. в одном блоке могут присутствовать участки (например, равнина) для которых нужна низкая детализация, и участки с высокочастотными особенностями, для которых требуется высокая детализация.
- При переключении детализации блоков, происходит сильное изменение в геометрии по всему блоку, а так же изменяется освещение на стыках между блоками.
- Высокочастотные особенности блока, ведут к глобальному усложнению блока.
- Алгоритм не поддерживает динамическое изменение рельефа.
- Алгоритм генерирует регулярную триангуляцию.

Не смотря на простоту реализации, высокую скорость работы алгоритма и то, что алгоритм хорошо оптимизируется для 3D акселераторов последнего поколения, получаемая геометрическая модель очень избыточна, и при переключении детализации блоков возникают сильные визуальные артефакты, что делает этот алгоритм неприемлемым для визуализации произвольных ландшафтов земной поверхности.

4 ИСПОЛЬЗОВАНИЕ ПРОГРЕССИВНЫХ СЕТОК (*PROGRESSIVE MESHES*) ДЛЯ УПРАВЛЕНИЯ ДЕТАЛИЗАЦИЕЙ.

4.1 Представление прогрессивных сеток.

Сетка треугольников – M задается набором вершин $V \in R^3$ и набором $F \in V^3$, упорядоченных троек (v_i,v_j,v_k), которые задают вершины треугольника. Соседними вершине v треугольники называются все треугольники из M которые, содержат вершину v. Соседним по ребру $e = \{v_i,v_j\}$, называют объеденение треугольников соседних вершинам v_i и v_j. Ребро, принадлежащее только одному треугольнику, называют граничным ребром. Вершины, из которых состоят граничные ребра, называют граничными треугольниками, остальные вершины из M называют внутренними вершинами. Валентностью вершины назовем количество ребер, к которым принадлежит данная вершина.

Пусть M_0 – грубая триангуляция, полученная из M, тогда прогрессивная сетка (*PM*) это последовательность n операций разделения вершин - *vsplit*, превращающих M_0 в M. Операция *vsplit* – добавляет в сетку одну новую вершину и два новых треугольника. Т.е. *PM* представляется, как

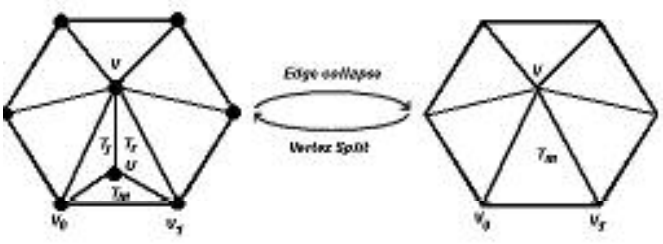

Рис 3. Пример выполнения операции *vsplit* и *ecol*.

$$M_0 \xrightarrow{vsplit_0} M_1 \xrightarrow{vsplit_1} \dots \xrightarrow{vsplit_{n-1}} M_n (M_n = M)$$

, операция vsplit_{i-1} генерирует i–ую сетку из *PM*. Таким образом, *PM* задается набором $(M_0,\{vsplit_0,vsplit_1,\dots,vsplit_n\})$. Можно ввести операцию обратную разделению вершин - схлопывание ребра - *ecol*, которая в противоположность *vspit*, убирает из M два треугольника и одну вершину. 0

На рисунке 3 представлен пример работы операций разделение вершин и схлопывания ребер, добавляется в случае *vsplit* или исключается в случае *ecol* вершина v и треугольники T_l и T_r.

4.2 Ориентационно-независимые прогрессивные сетки (*View Independent Progressive Mesh)*.

Пусть мы имеем *PM* заданную массивом вершин V, содержащий все вершины из полной сетки, массив триплетов F – $\{v_i,v_j,v_k\}$, которые на основе массива V описывают треугольники составляющие полную геометрическую модель и последовательность операций $\{\boldsymbol{vsplit}_0,\boldsymbol{vsplit}_1,\dots,\boldsymbol{vsplit}_n\}$.

И пусть мы имеем *PM* в некотором текущем состоянии - i, это состояние задается двумя массивами V и F, количеством активных вершин - *CurNumVerts* в V, и количеством активных треугольников – *CurNumTris* в F. Что бы увеличить количество активных вершин, мы применим $(i–1)$-ю операцию ***vsplit***, которая добавит одну активную вершину, соединенную новым ребром с некоторой вершиной $splitVert \in V$, так же вдоль нового ребра появится два новых треугольника (исключение составляют граничные вершины, в этом случае появляется один новый треугольник). Так же с появлением новых треугольников произойдет изменение в самой геометрической модели, См. например рисунок 3, треугольник T_m до применения операции **vsplit** задается триплетом $\{v_0,v,v_1\}$, а после применения задается с помощью триплета $\{v_0,u,v_1\}$. Т.к. последовательность выполнения операций ***vsplit*** постоянна, то новая вершина и триплеты, соответствующие новым треугольникам уже находятся в массивах V и F соответственно, и для добавления новой вершины, достаточно увеличить на единицу *CurNumVerts*, а для добавления новых треугольников, увеличить *CurNumTrees* на количество новых треугольников. Для выполнения операции разделения вершины остается только модифицировать триплеты, где до выполнения операции находилась вершина *splitVert*, а теперь должна быть новая вершина. Такие триплеты можно для каждой из операций ***vsplit*** выявить на этапе подготовки.

Параметризуем операцию разделения вершины, как ***vsplit***$(splitVert,numN,numF,f_0,..,f_{numF})$, где *splitVert* - вершина к

которой применяется операция ***vsplit***, *numN* – количество треугольников добавляемых в модель, $f_0,..,f_{numF}$ – старые треугольники, которые необходимо модифицировать для выполнения операции ***vsplit***, *numF* – количество этих треугольников. Таким образом, операция разделения вершины будет выглядеть следующим образом:

vsplit(*splitVert*,*numN*,*numF*,$f_0,..,f_{numF}$) {
    CurNewTris = *CurNewTris* + *numN*;
    Для каждого $f \in \{f_0,..,f_{numF}\}$ {
        Заменить *splitVert* в *f* на *CurNewTris*;
    }
    CurNewVert = *CurNewVert* + 1;
}

Для того, что бы уменьшить количество вершин в текущей *PM*, определим операцию ***ecol***, которую параметризуем так же как и ***vsplit***:

ecol(*splitVert*,*numN*,*numF*,$f_0,..,f_{numF}$) {
    CurNewTris = *CurNewTris* – *numN*;
    Для каждого $f \in \{f_0,..,f_{numF}\}$ {
        Заменить *CurNewVert – 1* в *f* на *splitVert*
    }
    CurNewVert = *CurNewVert* – 1
}

В каждой из операций ***vsplit*** можно хранить также текущее отклонение *PM* от полной модели, таким образом, на этапе подготовке (процесс построения *PM* рассмотрен в [5, 6]) мы подготавливаем последовательность операций $vsplit_0 \ldots vsplit_n$, соответствующее этим операциям отклонение $error_0 \ldots error_n$. А на этапе отображения, будем изменять *PM* до достижения ею некоторого выбранного нами отклонения, просто выполняя над *PM* операцию ***vsplit***, пока текущее отклонение больше выбранного, или ***ecol*** если меньше.

Использование ориентационно-независимых прогрессивных сеток для отображения рельефа земной поверхности.

В качестве данных для отображения берется карта высот размером *N*x*N*, затем она разбивается на куски размером *M*x*M*, таким образом, получается $L = (N/M)^2$ кусков. Затем для каждого из полученных кусков создается прогрессивная сетка и при отображении земной поверхности детализация каждого из кусков происходит независимо. Основная проблема при использовании ориентационно-независимых прогрессивных сеток *(VIMP)* для отображения земной поверхности, это стыковка блоков. Т.к. детализация блоков происходит независимо, то между соседними блоками могут возникать нестыковки. Существует несколько методов решения данной проблемы.

Во-первых, можно просто оставлять граничные ребра в максимальной детализации, т.е. так построить *PM*, что бы операции ***vspilt*** и ***ecol*** не применялись к граничным вершинам блока. Но тогда каждый блок не сможет быть меньше, чем *M* * 4 треугольника, тогда как для очень далеких блоков вполне достаточно 2-х треугольников.

Во-вторых, можно сделать, так что бы соседние блоки пересекались на границах, и граница каждого блока, лежала бы под соседним блоком. Таким образом, при упрощении поверхности, стыковка блоков хотя и будет неправильной, но зато не появляется разрывы на границах блоков.

4.3 Достоинства и недостатки использования *VIPM* для отображения рельефа земной поверхности.

В процессе реализации и тестирования данного алгоритма авторами были выделены следующие достоинства и недостатки:

Достоинства:

- Алгоритм отображает различные регионы земной поверхности с разными уровнями детализации.
- Алгоритм прост в реализации, основная нагрузка ложится на этап подготовки, сама же система детализации требует минимальных ресурсов.
- Алгоритм хорошо оптимизируется для акселераторов последнего поколения.
- Существует возможность создавать длинные последовательности *triangle strips*.
- Алгоритм поддерживает возможность изменения качества получаемой модели.
- Алгоритм работает с произвольными моделями земной поверхности.

Недостатки:

- Т.к. система упрощения работает с блоками, то получаемая триангуляция не полностью удовлетворяет требованиям минимальной избыточности.
- Алгоритм не поддерживает автоматическое устранение разрывов в геометрии, а существующие способы избегать этого, либо искажают, либо неоправданно усложняют геометрическую модель.
- Существует только косвенная возможность влияния на сложность получаемой геометрической модели, а, следовательно, скорость генерации одного кадра может изменяется.
- Алгоритм не поддерживает динамическое изменение рельефа.

Алгоритм *VIPM* хорошо оптимизируется для последнего поколения 3D ускорителей, не требует большого количества оперативной памяти и имеет отличную производительность. Но т.к. последовательность выполнения операций ***ecol*** и ***vsplit*** постоянна и зависит только от расстояния до наблюдателя, то получаемая геометрическая модель избыточна, т.е. для получения приемлемого визуального качества требуется большое количество полигонов. Ввиду своей вычислительной простоты алгоритм освобождает ресурсы для других частей приложения, что ведет к ускорению работы всей системы в целом, но объем получаемой геометрической модели позволяет использовать *VIPM* только вместе с 3D акселераторами последнего поколения, а значит, использования данного алгоритма ведет к потере масштабируемости системы и делает неприемлемым использование его в подсистеме визуализации рельефа земной поверхности.

4.4 Ориентационно-зависимые прогрессивные сетки *(View Dependent Progressive Mesh).*

Ориентационно-независимые прогрессивные сетки (*VIPM)* используют тот факт, что последовательность выполнения операций ***vsplit*** и ***ecol*** остается неизменной, и соответственно детализация блока зависит только от расстояния до наблюдателя, а положение и ориентация наблюдателя в расчет не принимается.

Действительно, изменение последовательности выполнения операций ***vsplit*** и ***ecol*** может привести к

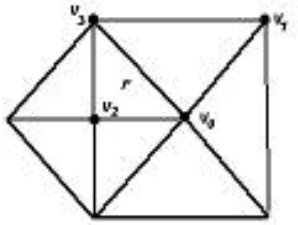

Рис 4. Изменение последовательности выполнения операций *ecol* для ребер $\{v_0,v_1\}$ и $\{v_2,v_3\}$ для представленной *PM*, приведет ее в некорректное состояние.

противоречивому состоянию *PM*. На рисунке 4 показан пример последовательного выполнения двух операций ecol для ребер $\{v_0,v_1\}$ и $\{v_2,v_3\}$, для выполнения первой операции необходимо модифицировать треугольник *F* заданный триплетом $\{v_0,v_2,v_3\}$, так что бы он задавался триплетом $\{v_1,v_2,v_3\}$, а при выполнении второй операции ***ecol*** треугольник *F* исключается из *PM*.

Изменение последовательности выполнения этих операций приведет к противоречивой ситуации. В [5] предложен способ решения данной проблемы, для этого операция разделения вершины параметризуется как: ***vsplit***$(v_0,v_1,f_L,f_R,f_0,f_1,f_2,f_3)$, где v_0 – новая вершина, v_1 – вершина, к которой применяется операция ***vsplit***, f_l и f_R – новые треугольники, f_0,f_1,f_2,f_3 – треугольники соседние с вершиной v_1, на которые повлияет операция ***vsplit*** (см. рисунок 5). Аналогично параметризуется операция ***ecol***$(v_0,v_1,f_L,f_R,f_0,f_1,f_2,f_3)$.

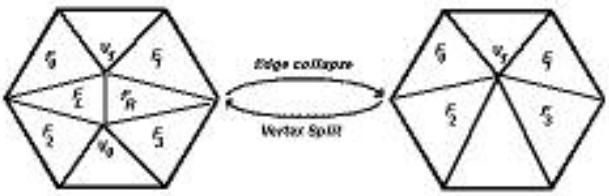

Рис 5. Пример выполнения операций разбиения вершины и схлопывания ребра.

Теперь для применения операции ***vsplit*** необходимо выполнение следующих условий: вершина v_1 и грани f_0,f_1,f_2,f_3 уже находятся в текущей триангуляции. Если эти условия выполняются, то операция ***vsplit*** называется легальной. Соответственно для выполнения ***ecol***, необходимо, что бы вершины v_0,v_1 и грани f_L,f_R,f_0,f_1,f_2,f_3 находились в текущей триангуляции. Если эти операции выполняются, то операция ***ecol*** называется легальной.

Прогрессивная сетка может быть представлена как набор бинарных деревьев, в котором корни деревьев представляют собой вершины начальной триангуляции M_0 полученной из ориентальной триангуляции *M*, а потомки представляют собой вершины получаемые при выполнении операции ***vsplit***.

Ориентационно-зависимая прогрессивная сетка (*VDPM)* поддерживает фронт активных вершин *PM* (на рис. 8 они выделены серым цветом). Применение операций ***vsplit*** и ***ecol*** к активным вершинам *PM*, сдвигает фронт активных вершин по дереву вверх и вниз.

Алгоритм поддерживает списки активных вершин - *V* и активных треугольников - *F*, при генерации каждого кадра для каждой активной вершины, в зависимости от положения наблюдателя и экранной ошибки этой вершины, принимается решение применить операцию ***vsplit***, ***ecol*** к данной вершине или не трогать ее. Если принято решения о применении операции ***ecol*** к данной вершине, то операция ***ecol*** выполняется, только в случае ее легальности. Если принято решение о применении операции ***vsplit***, то производится серия операции ***vsplit*** для выполнения условий легальности, такой процесс называется ***ForcedVSplit.***

Операция ***ForcedSplit***, рекурсивно обходит дерево и производит ряд операций ***vsplit***, необходимых для выполнения ***vsplit***(v), т.е. делающая эту операцию легальной. Условия легальности операции ***vsplit***, это наличие потомка вершины *v* в списке активных вершин - *V* и присутствие граней f_0,f_1,f_2,f_3 в текущей триангуляции - *F*. Если хоть одно из этих условий не выполняется, то производится ряд операций ***vsplit***, что бы сделать вершину v.parent активной, и привести грани f_0,f_1,f_2,f_3 в текущую триангуляцию.

Поддержка древовидной структуры *PM*, списка вершин, и треугольников, а так же всех связей между ними, необходимых для выполнения операции ***ForcedSplit***, делает алгоритм *VDPM* очень требовательным к количеству оперативной памяти установленной на компьютере. Нужно так же заметить, что время работы алгоритма пропорционально размеру списка активных вершин и размеру текущей триангуляции.

4.5 Достоинства и недостатки использования *VDPM* для отображения рельефа земной поверхности.

В процессе реализации и тестирования данного алгоритма авторами были выделены следующие достоинства и недостатки:

Достоинства:

- Алгоритм отображает различные регионы земной поверхности с разными уровнями детализации.
- Триангуляция удовлетворяет требованиям минимальной избыточности.
- Алгоритм автоматически избегает разрывов в геометрии и освещенности земной поверхности.
- Существует возможность поддержки алгоритмом морфинга, что уменьшает визуальные артефакты.
- Локальные особенности рельефа не ведут к глобальному усложнению геометрической модели.
- Алгоритм поддерживает создание последовательностей *triangle strip*.
- Существует возможность динамически изменять сложность

Недостатки:

- Несмотря на то, что алгоритм использует межкадровую когерентность, время генерации одного кадра зависит от размера получаемой геометрической модели.

- Алгоритм не поддерживает динамическое изменение рельефа.
- Алгоритм требует использования большого количества оперативной памяти для поддержания своих внутренних структур.
- Существует только косвенная возможность влияния на сложность получаемой геометрической модели, а, следовательно, скорость генерации одного кадра может изменяться.
- Сложно организовать динамическую подгрузку с диска различных блоков рельефа.

Алгоритм *VDPM* строит хорошо адаптированную к положению наблюдателя и желаемой экранной ошибке триангуляцию. Он может работать с произвольными геометрическими моделями земной поверхности, а не только с картой высот как многие алгоритмы визуализации земной поверхности. Но, к сожалению, время генерации одного кадра *VDPM*, несмотря на предложенные в [5] оптимизации, остается пропорционально сложности получаемой геометрической модели. Алгоритм так же использует большое количество оперативной памяти, что в итоге ведет к снижению производительности системы в целом.

5 ОПТИМАЛЬНО АДАПТИРУЮЩИЕСЯ СЕТКИ (*REAL-TIME OPTIMALLY ADAPTING MESHES*).

Оптимально адаптирующиеся сетки (*ROAM*) были предложены Duchaineau [2], как продолжение работ Lindstrom [1]. Duchaineau предложил две новые для управления детализацией треугольников. К сожалению предложенные им операции не позволяют в полной мере использовать преимущества межкадровой когерентности. Авторами данной работы предложена новая операция управления детализацией. А так же разработан алгоритм, использующий принцип непрерывной детализации для отображения больших участков земной поверхности, использующий эту новую операцию. В следующих главах будет описан и проанализирован подход, предложенный Duchaineau, так же будут описаны предложенные авторами операция и алгоритм использующий ее.

5.1 Представление геометрической модели

Основной структурой используемой модулем управления детализацией является двоичное дерево треугольников. На рисунке 6 показан пример подобного дерева. И соответствующая этому дереву геометрическая модель.

Корнем дерева является прямоугольный равнобедренный треугольник T это самый низкий – нулевой уровень детализации. Следующий уровень дерева получается, разбиением треугольника T на два новых прямоугольных равнобедренных треугольника с уровнем детализации $L = 1$, остальные уровни получаются рекурсивно. Из способа построения. Следует заметить, что у треугольника с уровнем детализации L могут быть соседи только уровней $L,L+1,L-1$. Геометрическую модель поверхности земли составляют только треугольники, соответствующие самым нижним узлам двоичного дерева треугольников, которые не имеют потомков, на рисунке 6 они выделены серым цветом.

5.2 Динамические преобразования геометрической модели.

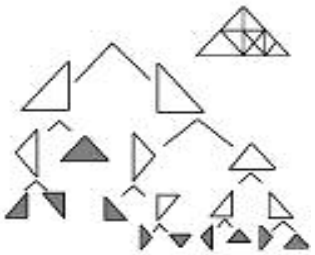

Рис 6. Пример двоичного дерева треугольников, серым цветом выделены треугольники составляющие геометрическую модель. В верхнем левом углу показана сама геометрическая модель

В [2] введены операции ***Split*** и ***Merge*** для модификации бинарного дерева.

Операция ***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 соответствующих треугольников T_1 и T_2. См. рисунок 7. В результате появляется новая вершина. Если разбиваемый треугольник T_0 имел соседа по основанию – T_1, то, чтобы избежать разрывов и нестыковок, мы вынуждены применить операцию ***Split*** к треугольнику T_1. Такой процесс называется ***Forced Split,*** и он может затрагивать несколько треугольников. См. рисунок 8.

Теперь начав с некоторой стартовой триангуляции ***T***, сопоставим каждому треугольнику в триангуляции некий приоритет $P(T) \in [0,1]$ и будем применять операцию ***ForcedSplit*** к наиболее приоритетному треугольнику до тех пор, пока получаемая триангуляция ***T'*** будет не достаточного размера или не достаточной точности. Если приоритет разбиения треугольника выбирать основе ошибки этого треугольника, то в итоге мы получим оптимальную триангуляцию, т.е. для триангуляции данного размера

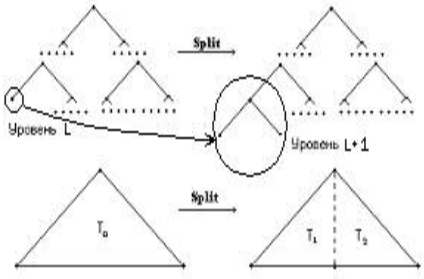

Рис 7. Операция *Split*

максимальная ошибка будет минимальна. Так же можно доказать что такой алгоритм дает оптимальную триангуляцию на каждом кадре [2]. К сожалению, данный алгоритм не использует межкадровую когерентность, и каждый раз начинает с некой начальной триангуляции. Что бы этого избежать используется операция ***Merge.***

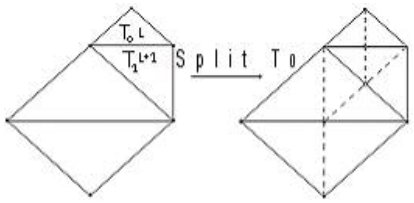

Рис 8. Разбиение одного треугольника ведет к появлению десяти новых.

Операция ***Merge***: Для использования межкадровой когерентности в [2] введена операция Merge над геометрическую структурой - “ромб” (*diamond*), состоящий из двух узлов двоичного дерева треугольников (соответствующие им треугольники T_L, T_R), удовлетворяющих следующим требованиям:

- Треугольники T_L, T_R должны иметь одинаковый уровень детализации.
- Их дети не имеют потомков, т.е. соответствующие детям треугольники - T_0,T_1,T_3,T_4, содержатся в геометрической модели
- 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. Т.к. скорость алгоритма предложенного в [2] использующего межкадровую когерентность зависит от количества изменений в геометрической модели, то если наблюдатель двигается быстро или совершает резкие повороты, то время генерации одного кадра в таком случае будет велико. Для ускорения этого процесса авторами данной работы была введена новая операция ***enhMerge***, она не ограничивает свое действие только на “ромбы”, любой

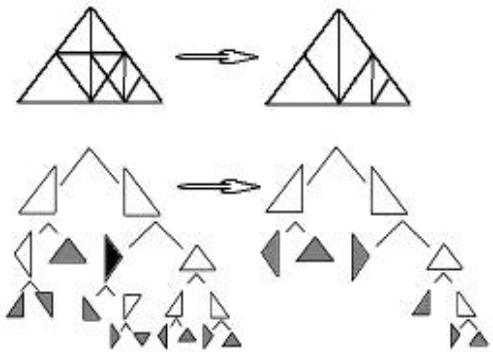

Рис 9. Пример выполнения операции *enhMerge.*

треугольник, который был до этого разделен, может быть соединен обратно, при этом все поддерево соответствующее данному треугольнику будет уничтожено. При этом для сохранения непрерывности геометрической модели и избежания нестыковок, необходимо применить операцию ***enhMerge*** ко всем соседним треугольникам. На рисунке 9 показан пример применения операции ***enhMerge*** к треугольнику выделенного черным цветом. Таким образом, изменение дерева, которое выполняется с помощью нескольких последовательных операций ***Merge***, можно проделать одним применением операции ***enhMerge***.

Основная проблема алгоритмов использующих межкадровое подобие, состоит в том, что в случае двух сильно различающихся последовательных кадров, время генерации одного кадра будет большим, за счет сильного изменения геометрической модели. Duchaineau предложил в этом случае отказываться от использования межкадрового подобия, и начинать триангуляцию заново, использование же операции ***enhMerge***, позволяет алгоритму работать даже в этом особом случае.

Итак, мы имеем операции, позволяющие гибко изменять уровень детализации сцены. Рассмотрим алгоритм, который управляет процессом разбиения и слияния треугольников для получения наиболее подходящей степени детализации.

Алгоритм рекурсивно обходит двоичное дерево треугольников, полученное на прошлом кадре, и для каждого треугольника на основе его экранной ошибки принимает решения, нужно к нему применить операцию ***forcedSplit***, ***enhMerge*** или оставить его с текущей детализацией.

5.3 Достоинства и недостатки алгоритма *ROAM*.

В процессе реализации и тестирования данного алгоритма авторами были выделены следующие достоинства и недостатки:

Достоинства:

- Алгоритм отображает различные регионы земной поверхности с разными уровнями детализации.
- Генерируемая триангуляция удовлетворяет требованию минимальной избыточности, т.е. важные с точки зрения текущего кадра части рельефа представлены с большей детализацией.
- Алгоритм в силу своей природы, автоматически устраняет разрывы в геометрии и в освещении земной поверхности.
- Алгоритм использует межкадровую когерентность и сложность алгоритма пропорциональна изменению в геометрической модели.
- Высокочастотные особенности не ведут к глобальному усложнению модели.
- Алгоритм поддерживает генерацию *triangle strip* и отбраковку граней, не попадающих в область видимости.

Недостатки:

- Алгоритм генерирует регулярную триангуляцию.
- Алгоритм использует в качестве геометрической модели только карту высот.
- Сложно организовать динамическую подгрузку с диска различных блоков рельефа.
- Алгоритм не способен полностью загрузить работой 3D акселератор последнего поколения.

Алгоритм *ROAM* хоть и строит более избыточную геометрическую модель, чем *VDPM* [1], но по сравнению с *VDPM* он менее ресурсоемкий. Кроме того, он использует межкадровую когерентность и количество операций требующихся для генерации следующего кадра пропорционально изменению геометрической модели. Геометрическая модель после выполнения операций ***ForcedSplit*** и ***enhMerge*** остается корректной, что позволяет прервать работу алгоритма в любой момент и продолжить

уже на следующем кадре. Эта особенность позволяет сохранять время генерации одного кадра постоянной. Алгоритм так же имеет возможность явным образом задавать сложность геометрической модели, что позволяет динамически управлять производительностью системы. В следующей главе будет дано более подробное сравнение описанных алгоритмов.

6 СРАВНЕНИЕ АЛГОРИТМОВ.

Авторами данной работы были программно реализованы все перечисленные алгоритмы визуализации рельефа земной поверхности в реальном времени. Так же было произведено сравнение описанных алгоритмов, с целью выявить наиболее подходящий для применения в подсистеме визуализации земной поверхности. Т.к. нас в меньшей степени волнует скорость и сложность системы подготовки, то сравнение касалось только скорости компонентов времени исполнения, визуального качества получаемой геометрической модели и возможности масштабирования подсистемы визуализации.

Все описанные алгоритмы поддерживают морфинг вершин, что уменьшает визуальные артефакты, но алгоритму основанному на методе регулярного прореживания, требуется производить морфинг целого блока, что является ресурсоемким. Кроме того, способ сопряжения блоков ведет к сильным скачкам в освещении при смене детальности блоков.

VDPM генерирует геометрическую модель с количеством треугольников на 50-70% меньше, чем *ROAM* при одинаковом визуальном качестве [1]. Алгоритм *VIPM* в силу того, что уровень детальности выбирается для всего блока, а не для каждого треугольника в отдельности, генерирует геометрическую модель еще более избыточную, чем *ROAM*. А алгоритм, основанный на методе регулярного прореживания дает, настолько избыточную геометрическую модель, что делает использование алгоритма вместе со слабыми 3D ускорителем нерациональным. Таким образом, из-за избыточности геометрической модели и других присущих этому алгоритму недостатков, он не подходит для использования в подсистеме визуализации земной поверхности.

Алгоритм *VDPM*, хотя и использует межкадровую когерентность, но скорость генерации одного кадра зависит от сложности получаемой геометрической модели, кроме того, алгоритм требует большого количества оперативной памяти. Время же генерации одного кадра в *ROAM* и *VIPM* пропорционально количеству изменений в геометрической модели, что обычно составляет 5% от общего объема геометрической модели. Таким образом, при использовании алгоритмов *VIPM* или *ROAM* освобождается процессорное время для других частей приложения, что делает применение этих алгоритмов более предпочтительным.

Использование *VDPM* оптимально в случае систем с быстрым графическим процессором, т.к. алгоритм сильно избыточную геометрическую модель и почти не нагружает CPU, а основная нагрузка ложиться на систему подготовки. Использование же *ROAM* оптимально в случае, когда CPU быстрее графического процессора, т.к. за счет использования процессорного времени алгоритм сильнее упрощает геометрическую модель. Но из-за избыточности геометрической модели использование *VDPM* на машинах со слабым графическим процессором является невозможным, а значит, используя этот алгоритм в подсистеме визуализации рельефа земной поверхности, мы теряем широкий класс платформ, на которых может запускаться приложение, что делает использование *VDPM* не приемлемым.

7 ЗАКЛЮЧЕНИЕ

В результате сравнительного анализа существующих алгоритмов в качестве основы для подсистемы визуализации рельефа земной поверхности выбран алгоритм *ROAM*. Авторами данной работы предложена новая операция управления детализацией, позволяющая существенно уменьшить время генерации геометрической модели в случае малокогерентных кадров. На основе модернизированного алгоритма *ROAM* авторами реализована и оптимизирована, для получения возможно большего быстродействия, масштабируемая подсистема визуализации рельефа земной поверхности.

При тестировании подсистемы на примере некоторого фрагмента земной поверхности 3000x3000 $м^2$(шаг 1 м), генерация одного кадра изображения занимала ≈15 мс., причем ≈8 мс. из них уходило на адаптацию геометрической поверхности к положению и ориентации наблюдателя (тестирование проходило на Pentium III – 800 МГц; RAM – 256 Мб; GForce 2 GTS). Такой скорости в большинстве случаев оказывается достаточно для эффективного применения библиотеки в системах синтеза визуальной обстановки.

В настоящее время реализованная авторами подсистема, успешно используется в ряде систем виртуальной реальности.

8 ЛИТЕРАТУРА:

[1]. 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 Proc. SIGGRAPH ’96, pages 109-118,Aug. 1996.

[2]. M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, M. B. Mineev Veinstein, *ROAMing Terrain: Real time optimally adapting meshes*, IEEE Visualisation’97, Nov 1998 pp.81-88.

[3]. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer graphics: principles andpractic*e, Sec. Ed. in C, Addison-Wesley, 1996.

[4]. Willem H. de Boer,*Fast Terrain Rendering Using Geometrical MipMapping,*E-mersion Project, October 2000*,* http://www.connectii.net/emersion

[5]. H. Hoppe, *Smooth view-dependent level-of-detail control and its application to terrain rendering*, IEEE Visualization ’98, Oct. 1998.

[6]. S. Melax, A Simple, *Fast and Effective Polygon Reduction Algorithm*, Game Developer Magazine, Nov 1998, pp. 44-49

Об авторах:

Все авторы работают в компании Софтлаб-НСК, Новосибирск.

e-mail:

Н.А. Елыков - nicolas@ softlab-nsk.com

М.М.Лаврентьев – lavr@softlab-nsk.com

И.В.Белаго – bel@softlab-nsk.com

С.А.Кузиковский – stas@softlab-nsk.com

Ю.Ю. Некрасов - nek@ softlab-nsk.com