Реализация алгоритма визуализация местности в режиме реального времени

Ерухимов В.Л., Капустин А.Д., Малашкина А.В.,
Нижегородская Лаборатория Программных Технологий, ЗАО, Нижний Новгород, Россия

Аннотация

В настоящей работе предлагается реализация алгоритма визуализация местности, изложенного в работе [1]. Этот алгоритм позволяет значительно снизить затраты памяти, т.к. вычисляет сетку для текущего положения камеры без использования предварительно полученных данных. Эффективность алгоритма достигается за счет использования регулярной сетки и применения заранее определенных простых способов ее огрубления при удалении камеры от отображаемого участка поверхности.

Предлагается реализация алгоритма и тестовое приложение, моделирующее полет над местностью в режиме реального времени. Набор опций, возможность задания произвольной траектории полёта, позволяют контролировать реалистичность работы алгоритма. Визуализация поверхности осуществляется с использованием текстур. Построение триангуляции и загрузка текстурных страниц в оперативную память производится только для видимого участка поверхности.

Ключевые слова: *LOD, визуализация, регулярное поле высот, strip.*

1. ВВЕДЕНИЕ

Задаче визуализации местности в режиме реального времени в последние годы уделено значительное внимание [1-3]. Приложения, использующие визуализацию местности, предъявляют жесткие требования как к качеству, так и к скорости визуализации. Этим требованиям отвечают алгоритмы, использующие понятия многоуровневых сеток (progressive meshes) или многоразрешаюших (multiresolution) моделей поверхностей [4-6]. Такие модели описывают множество триангуляций поверхности, начиная с самой грубой триангуляции и заканчивая самой подробной. Предложены различные способы эффективного построения множества таких триангуляций на шаге предобработки. На этапе визуализации в режиме реального времени выбирается триангуляция поверхности, соответствующая текущему положению камеры и заданной ошибке отображения и содержащая небольшое число треугольников. К сожалению, использование таких алгоритмов приводит к большим затратам памяти, несмотря на использование специально разработанных компактных структур данных. В настоящей работе предлагается реализация алгоритма, позволяющая значительно снизить затраты памяти засчет вычисления сетки для текущего положения камеры без использования предварительно полученных данных.

2. ОПИСАНИЕ АЛГОРИТМА ФИНЧА И БИШОПА

Предварительный обзор.

Данный алгоритм предназначен для визуализации местности, заданной в виде регулярного высотного поля произвольного размера. В силу специфики алгоритма ширина и высота поля должны быть степенью 2. Если это не так, то выделяется дополнительная память и поле достраивается до нужных ближайших размеров. Визуализируется только исходная область.

Алгоритм использует понятие уровня детализации (LOD). На каждом уровне LOD на поверхности задается равномерная квадратная сетка. Шаг разбиения свой для каждого уровня и представляет собой степень 2: для LOD=0 шаг является минимальным, для LOD=1 длина стороны каждой ячейки в два раза больше, чем для LOD=0, и т.д. Ячейки для LOD=LODmax являются самым грубым разбиением и называются *страницами.* Количество уровней является параметром алгоритма.

Результатом работы алгоритма является триангуляция, представленная в виде последовательности strip. Каждый отдельный strip направлен вдоль оси Ox поверхности (минорное направление), а последовательность strip упорядочена вдоль оси Oy (мажорное направление).

В процессе триангуляции происходит обработка не всей поверхности, а только области, видимой в данный момент на экране. Проекция этой области на плоскость Oxy называется *footprint.*

Strip может состоять из треугольников различных уровней детализации. Текущий LOD определяет размер квадрата, который затем разбивается на треугольники определенным образом. На каждом шаге текущий уровень детализации может быть повышен или понижен на 1, или оставлен прежним по сравнению с соседними в мажорном и минорном направлении квадратами, в соответствии с допустимой ошибкой в текущей точке. Функция ошибки является функцией расстояния от точки взгляда до визуализируемой точки поверхности (в данной реализации функция ошибки линейно зависит от этого расстояния). Затем треугольники пришиваются к strip, и процесс повторяется, пока strip не достигнет границы footprint.

Особенностью алгоритма является то, что уровни детализации соседних в минорном и мажорном направлениях ячеек не могут отличаться больше, чем на 1.

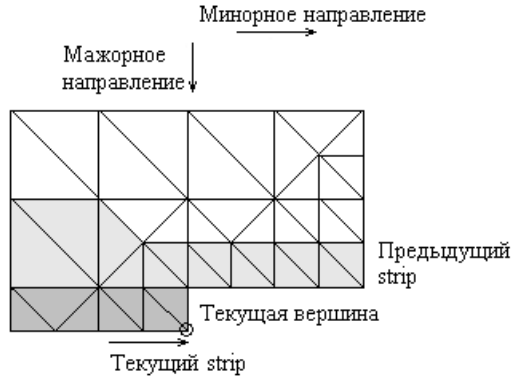

Рис.1 Генерирование стрипов.

Существует пять способов разбиения ячейки (рис.2).

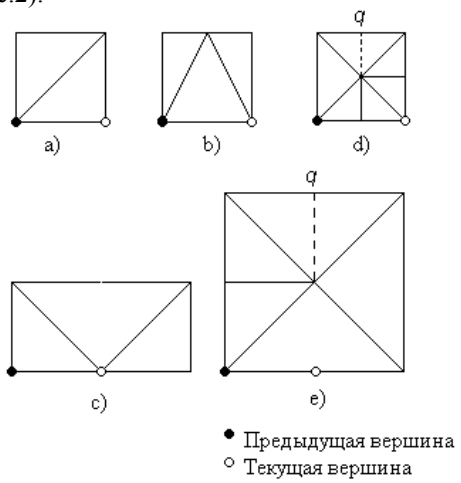

Рис.2. Способы разбиения ячеек.

Три из них (a, b, c) соответствуют случаю, когда уровень детализации текущего strip не меняется, а разбиение зависит от LOD предыдущего strip в данном месте. Четвертый способ (d) соответствует случаю повышения уровня детализации (рис.3). Следующая ячейка будет генерироваться со стороной, в два раза меньшей. Так как по высоте она тоже в два раза меньше, то возникает необходимость в создании еще одного strip. Его построение выполняется рекурсивно.

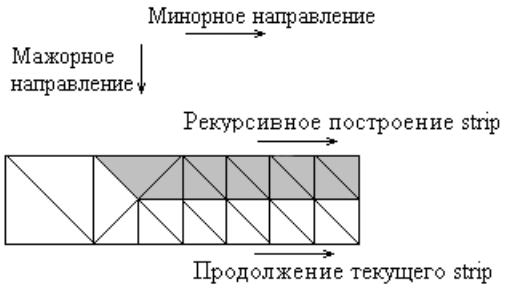

Рис.3. Улучшение детализации.

Пятый случай (e) – случай понижения уровня детализации. Текущий strip заканчивается, а следующий – в этом месте увеличивает шаг в два раза *(рис.4).*

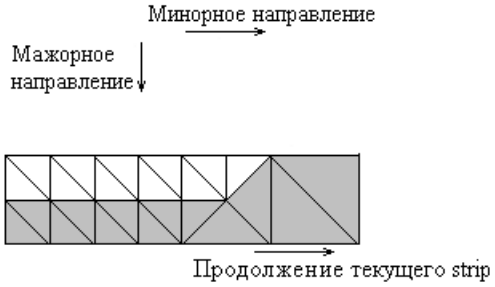

Рис.4. Огрубление детализации.

Выбор footprint.

Frustum – это характеристика камеры, через которую ведется наблюдение за высотным полем. Frustum представляет собой усеченную пирамиду с прямоугольником в основании. Плоскости, образующие основания пирамиды, являются максимально близким и максимально удаленным видимыми участками камеры. Боковые грани пирамиды ограничивают область видимости сверху, снизу, справа и слева. Таким образом, те и только те предметы, которые попали в данный момент во frustum, могут быть видимы наблюдателем.

Все участки местности, являющиеся видимыми с конкретной точки наблюдения, лежат внутри многогранника, образованного пересечением frustum и плоскостей Z=Zmin и Z=Zmax, где Zmin и Zmax – это, соответственно, минимальная и максимальная координаты по оси Oz, достигаемые высотным полем. Проекция этого многогранника на плоскость Oxy является областью footprint. Так как минимальным объемом загружаемой в оперативную память информации является страница, то формируется список потенциально видимых страниц, включающий в себя все те страницы, которые хотя бы частично попадают в область footprint. Алгоритм триангулирует только область, состоящую из этих страниц.

Алгоритм генерации ячеек.

Данный алгоритм описывает способ разбиения ячейки для текущей вершины P и текущего уровня детализации L.

Положение точки P называется выровненным относительно уровня L в мажорном (минорном) направлении, если y-координата (x- координата) точки P кратна размеру квадрата уровня L.

Обозначения:

P+M (P-M) - обозначает точку, расположенную вправо (влево) от точки P на размер стороны ячейки (M) текущего уровня L в мажорном направлении;

P+m (P-m) - обозначает точку, расположенную вправо (влево) от точки P на размер стороны ячейки (m) текущего уровня L в минорном направлении;

LODFUNC(P) - функция, вычисляющая требуемый уровень детализации в P.
DrawA() - функция продолжения strip в случае Рис.2.a.
DrawB() - функция продолжения strip в случае Рис.2.b.
DrawC() - функция продолжения strip в случае Рис.2.c.
DrawD() - функция продолжения strip в случае Рис.2.d.
DrawE() - функция продолжения strip в случае Рис.2.e.
GenerateNextQuad() – функция, определяющая способ разбиения ячейки для текущей вершины P и текущего уровня детализации L.
NormalQuad() – функция, определяющая способ разбиения ячейки для случаев Рис.2.a,b,d.
AlignedInMajor(Q, L) – функция, возвращающая true, если координаты точки Q выровнены в мажорном направлении по отношению к LOD L; false в противном случае.
AlignedInMinor(Q, L) – функция, возвращающая true, если координаты точки Q выровнены в минорном направлении по отношению к LOD L; false в противном случае.
UsedInPrevStrip(Q) - функция, возвращающая true, если вершина Q была использована в предыдущем strip, и false в противном случае.
IsPreviousStripEnd() - функция, возвращающая true, если для текущей ячейки не существует ячейки предыдущего strip, расположенной непосредственно над ней, и все вершины предыдущего стрипа располагаются слева от текущей ячейки; false в противном случае.
IsPreviousStripNotBegun() - функция, возвращающая true, если для текущей ячейки не существует ячейки предыдущего strip, расположенной непосредственно над ней, и все вершины предыдущего стрипа располагаются справа от текущей ячейки; false в противном случае.

BOOL PermitToMerge – переменная, равная true, если LOD на текущем шаге при необходимости может быть увеличен на 1, и false в противном случае, т.е. если LOD одной из соседних ячеек не позволяет этого сделать.
BOOL IncPrevLOD1, IncPrevLOD2 – переменная, равная true, если LOD на текущем шаге при необходимости может быть уменьшен на 1, и false в противном случае, т.е. если LOD одной из соседних ячеек не позволяет этого сделать.

GenerateNextQuad()

```
if (AlignedInMinor(P, L+1)) NormalQuad();
//Выровнено в минорном направлении по
//отношению к LOD L+1
else if (AlignedInMajor(P, L+1)) {
//Выровнено в мажорном направлении по
//отношению к LOD L+1)
    if (UsedInPreviousStrip(P-M) and not
    UsedInPreviousStrip(P-M+m))
    {
        DrawE();
        P=P+3m;
        L=L+1;
    }
    else
        NormalQuad();
    };
    else {
        // Не выровнено в обоих направлениях:
        if (IsPreviousStripNotBegun())
        {
            DrawA();
            P=P+m;
        } else {
        /*  Проверка, произошло ли в предыдущем
        strip на следующем шаге уменьшение LOD:  */
        PermitToMerge = (LODFUNC(P)>L) and not (
        UsedInPrevStrip(P-M+m/2)                and
        UsedInPrevStrip(P-M+1.5m)               and
        UsedInPrevStrip(P-M+2.5m));
        if (UsedInPrevStrip(P-M+3m) or PermitToMerge)
            EndStrip();
        else
        {
        if(UsedInPrevStrip(P-M)
            or IsPreviousStripNotBegun()
            or IsPreviousStripEnd())
            {
                NormalQuad();
            }
            else {
                /*  Проверка, закончился ли через шаг
                предыдущий  strip  для  увеличения
                LOD:  */
                if  not  AlignedInMinor(P-M+3m,  L+2)
                and not UsedInPrevStrip(P-M+5m) and
                IsPreviousStripEnd())
                        EndStrip();
                else {
                    DrawC();
                    P=P+2m;
                }
            }
        }
    }
}
```

NormalQuad()

```
if (IsPreviousStripNotBegun())
{
    DrawA();
    P=P+m;
}
/*  Проверка, произошло ли в предыдущем strip на
следующем шаге уменьшение LOD:  */
else {
    DecPrevLOD = UsedInPrevStrip(P-M+0.75m) or
    UsedInPrevStrip(P-M+0.25m);
    if ( not DecPrevLOD &&  LODFUNC(P)<L )
    {
```

```
        /* Проверка, закончился ли через шаг
        предыдущий strip для увеличения LOD:  */
        Присвоить значения переменной IncPrevLOD1;

        /* Проверка, закончился ли на следующем
        шаге предыдущий strip для увеличения LOD:*/
        Присвоить значения переменным IncPrevLOD2
        и PermitToSplit;

      if (DecPrevLOD or PermitToSplit )
      {
          //Уменьшение текущего LOD:
          DrawD();
          Выполнить рекурсивное построение стрипа
          уровня L-1, начиная с вершины P-M/2+m/2;
          L=L-1;
          P=P+m;
      }
      else {
                // Текущий уровень LOD не меняется:
                if (UsedInPrevStrip(P-M-m/2))
                {
                    DrawB();
                    P=P+m;
                }
                else {
                        DrawA();
                        P=P+m;
                }
          }
       };
};
```

Создание нового strip на границе footprint.

При создании нового strip на границе footprint возможны две ситуации. Если в предыдущем strip существует ячейка, которая расположена непосредственно над текущей ячейкой нового strip, то за текущий уровень детализации берется LOD этой ячейки.

В противном случае, то есть когда новый strip сдвинут вправо относительно предыдущего, за текущий уровень LOD ячейкам нового strip берется уровень детализации первого квадрата предыдущего strip. При этом текущий LOD можно увеличить или уменьшить на один, если требуемый уровень детализации соответственно больше или меньше полученного. Дальнейшее изменение шага запрещено, пока текущий strip не достигнет предыдущего в минорном направлении. Это делается для того, чтобы избежать нестыковки в уровнях детализации этих strip.

3. ОПИСАНИЕ СИСТЕМЫ ВИЗУАЛИЗАЦИИ FLIGHT.

Приложение Flight было создано для тестирования алгоритмов, упрощающих триангуляции поверхностей. Система визуализации имитирует полёт над поверхностью в реальном времени. При каждом изменении положения камеры происходит запуск тестируемого алгоритма. В такой ситуации пользователь тестовой системы получает возможность контролировать качество и скорость работы алгоритма. Набор свойств (выбор скорости полёта, различных методов отображения поверхности, в том числе в виде сетки, и т.д.) дают возможность детального анализа. Отображение поверхности реализовано на основе библиотеки OpenGL.

Траектория полёта выбирается опционально. Один из вариантов - генерация пути заранее, другой - свободный полёт с использованием инструментов управления. В приложении Flight разработан интуитивный способ генерации траектории полёта. Путь привязан к конечному числу точек. Пользователь имеет возможность задавать точки как относительно узла поверхности, так и в объектных координатах. Привязка к узлам даёт возможность легко менять высоту траектории над поверхностью. Пользователю предоставляется возможность выбрать метод, по которому проводится траектория через точки. Полученная траектория может быть сохранена, а затем загружена.

Взаимодействие с алгоритмом происходит через универсальный COM-интерфейс, что даёт возможность работать с разными алгоритмами триангуляции, не изменяя исходный код приложения. Кроме положения камеры, на вход алгоритму подаются параметры клиппирующих плоскостей; обрабатывается только видимая часть поверхности, что существенно ускоряет работу.

4. ПОДГОТОВКА ДАННЫХ ДЛЯ ВИЗУАЛИЗАЦИИ.

Данные представлены в виде высотного поля, записанного в формате bmp, 512x512, а так же набора из 16 текстур, также в формате bmp, 128x128.

Высотное поле создано с помощью генератора высотных полей Terragen 0.6, а затем отредактировано в Adobe PhotoShop 5.0. Карты текстур созданы также при помощи графического редактора Adobe PhotoShop 5.0.

Алгоритм реализован на Microsoft Visual C++, данные визуализировались посредством OpenGL.

5. РЕЗУЛЬТАТ РАБОТЫ.

На рис.5 изображена поверхность, триангулированная с помощью вышеописанного алгоритма и визуализированная посредством Flight.

На рис.6 можно увидеть триангуляцию того же участка поверхности без наложения текстуры.

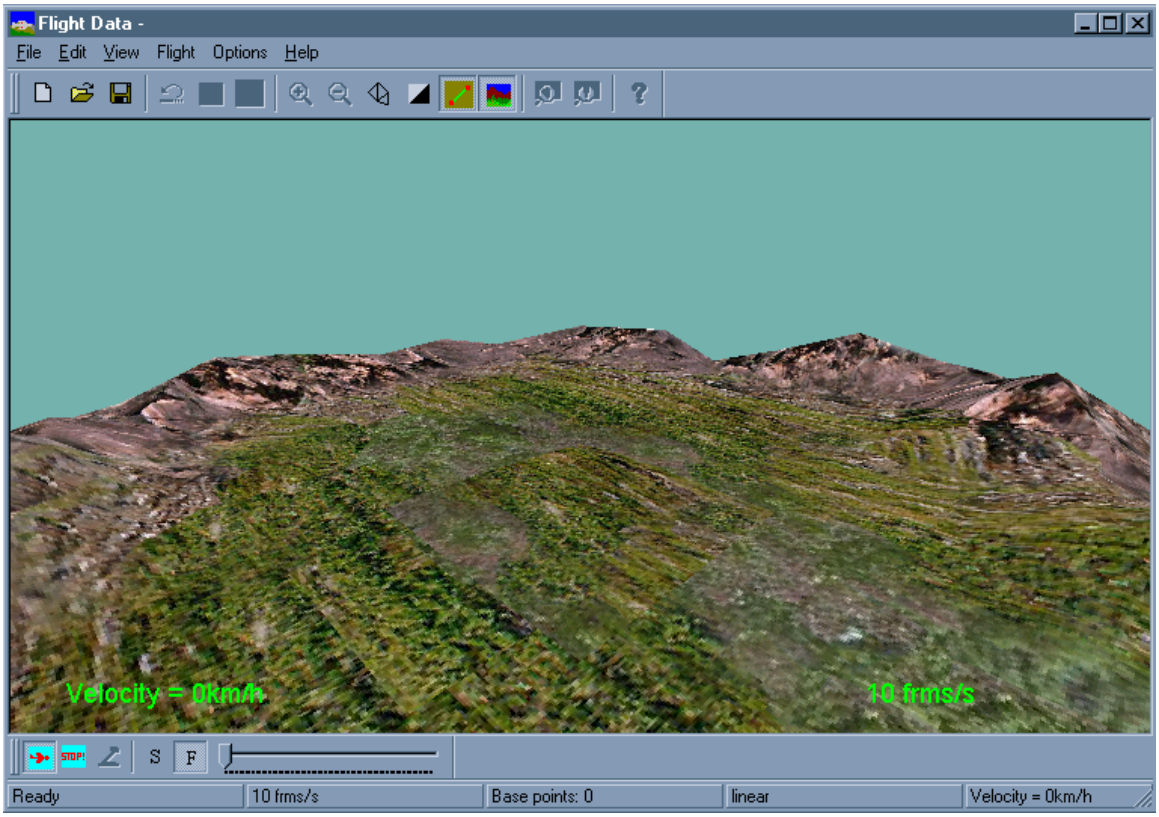

Рис.5. Поверхность, триангулированная с помощью вышеописанного алгоритма и визуализированная посредством Fligh

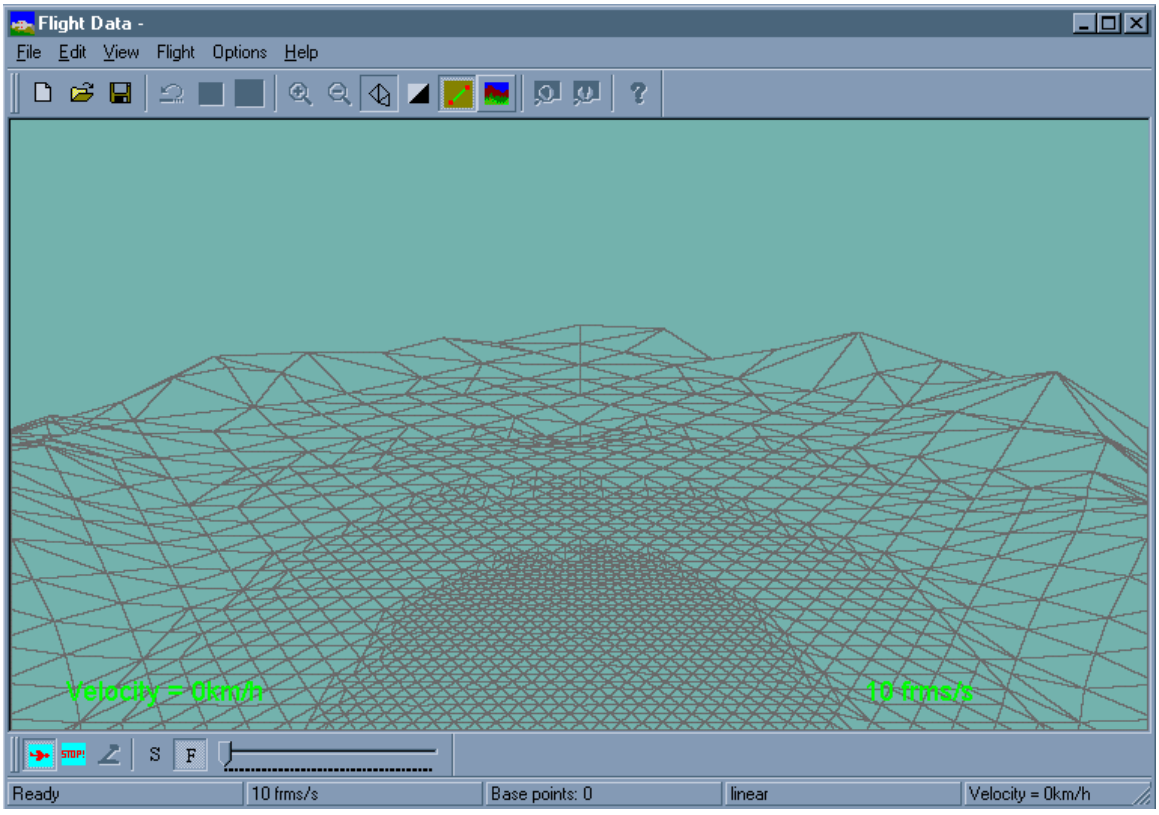

Рис.6. Триангуляция того же участка поверхности без наложения текстуры.

В таблицах 1 и 2 приведены зависимости скорости визуализации местности в кадрах в секунду от размера окна на экране и от количества треугольников в кадре. Результаты приведены для поверхности размера 512x512, размер страницы 128x128.

Первый тест проводился на PentiumII, 266MHz, видеокарта не поддерживает двумерные операции.

Размер окна	**240x84**	**754x406**	**1020x617**
Кол-во треугольников			
157	20	6	3
280	20	6	3
640	20	6	3
1060	20	6	3
1650	20	6	3

Таблица 1. Результаты тестирования на машине без аппаратной поддержки функций OpenGL

По результатам теста видно, что скорость отображения практически не зависит от числа треугольников. Это происходит потому, что основная часть времени тратится на отрисовку.

Следующий тест проводилось на PentiumII, 333MHz, видеокарта FireGL 4000 (Real Image tech gy, E&S) с поддержкой аппаратной реализации двумерных операций библиотеки OpenGL.

Размер окна	**240x84**	**754x406**	**1020x617**
Кол-во треугольников			
157	100	50	50
280	100	50	50
640	53	50	50
1060	50	50	50
1650	34	34	33
2400	34	34	33
8500	17	17	17
25500	8	8	7

Таблица 2. Результаты тестирования на машине с аппаратной поддержкой функций OpenGL

Приведенные результаты свидетельствуют, что предложенная реализация алгоритма позволяет эффективно построить аппроксимацию исходного высотного поля и отобразить в режиме реального времени полученную триангуляцию, содержащую около 20 000 текстурированных треугольников на машине PENTIUM II.

Литература.

[1] Mark Finch and Lars Bishop Sorted Height Field Rendering. http://www.ndl.com/wpapers/ndlter.html

[2] Cohen-Or D., Levanoni Y. Temporal Continuity of Levels of Detail in Delaunay Triangulated Terrain http://www.math.tau.ac.il/~daniel/

[3] Lindstrom P., Koller D., Ribarsky W., Hodges 3.L. F., Turner G. A**.** Real-Time, Continuous Level of Detail Rendering of Height Fields SIGGRAPH'96 Proceedings, pp. 109 -118

[4] Hoppe H**.** Progressive Meshes SIGGRAPH '96 Proc., pp. 99-108

[5] De Floriani L., Magillo P., Puppo E. Building and Traversing a Terrain at Variable Resolution (Extended Abstract) Proceedings IEEE Visualization'97, October 19-24, 1997, Phoenix, AZ

[6] Klein R., Huttner T. Simple camera-dependent approximation of terrain surfaces for fast visualization and animation In R. Yagel, editor, Visualization 96. ACM, November 1996

Об авторах

Капустин Александр, (e-mail: kap@nstl.nnov.ru)?
Ерухимов Виктор, студент 6 курса факультета Высшая школа общей и прикладной физики Нижегородского государственного университета, (e-mail: vic@nstl.nnov.ru),
Малашкина Анна, студентка 6 курса факультета Вычислительной математики и кибернетики, (e-mail: Anna_Malashkina@nstl.nnov.ru).

Abstract

The implementation of the realtime terrain visualization algorithm

A. Kapustin, V. Eroukhimov, A. Malashkina

The algorithm implementation, suggested in [1], is presented in the paper. This algorithm provides a significant reduction of memory loss since it calculates the mesh for a current position of a camera directly, without using precalculations. The use of a regular mesh as the most detailed representation and the application of simple predefined methods of making a mesh coarser along with a camera moving away from a surface makes the algorithm very efficient.

The algorithm implementation and the test application that simulates a realtime flight above a terrain are presented here. An advanced method of generating a flight trajectory and a set of options allow a user to check if the algorithm gives a realistic surface. Textures are used when visualizing a terrain. The triangulation is calculated and texture pages are loaded into memory only for a visible part of a surface.