LOBACHEVSKY STATE UNIVERSITY
of NIZHNI NOVGOROD
National Research University

Computing Mathematics and Cybernetics faculty

Software department

**CS255. Computer Graphics** Introduction Course

# Моделирование окружающей среды на основе шумов

## 3D Models and Algorithms for Shaders.

Турлапов, Вадим Евгеньевич
проф. каф. МО ЭВМ

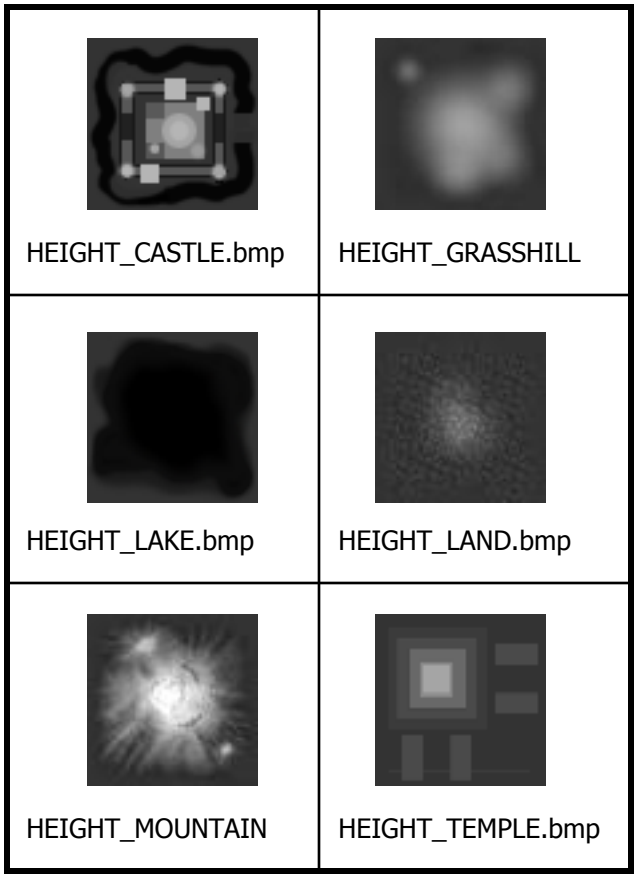
Изучению данного материала предшествует изучение основ языка программирования шейдеров GLSL или MS HLSL и приемов программирования на его основе моделей освещения и текстурирования

# ОСНОВЫ ЛАНДШАФТНОЙ ГЕОМЕТРИИ

**Карта высот** - простейшая и самая распространенная схема организации входных данных о геометрии ландшафта: двухмерный массив значений высот, вписанных в регулярную сетку, т.е. для каждой позиции *(x, y)* на сетке в карте хранится значение высоты (z).

z-информацию обычно хранят в байтах без знака:
0 обозначает самую малую, 255 - самую большую высоту. Полезным аспектом такого подхода является то, что двумерный массив, состоящий из байтов, идентичен битовой карте в оттенках серого.

DXSDK\Samples\C++\Demos:



# **Процедурные карты высот.** Гладкие шумы

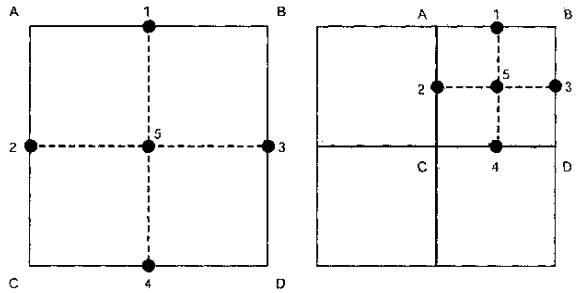
## Метод «Смещение средней точки»

Рекурсивный процесс, именуемый смещением средней точки (midpoint displacement):

•Изображение делится на четыре квадранта как показывает пунктир, что создает пять новых узлов, показанных пронумерованными точками. Базовое значение в каждой точке мы вычислим как среднее значений узлов, с которыми она связана. Например, базовое значение точки 1 мы примем равным среднему значений, находящихся в угловых точках A и B.

•Продолжив работу с точкой номер 1, сместим ее, используя случайное значение в диапазоне [-Дельта, Дельта]. Аналогично генерируем случайные значения смещений точек 2, 3 и 4. Точка 5 отличается тем, что ее базовое значение находится как среднее всех четырех углов.

•$Дельта_{k+1}$ = $Дельта_k$ * Шероховатость
(идеальное значение шероховатости = 0,5).



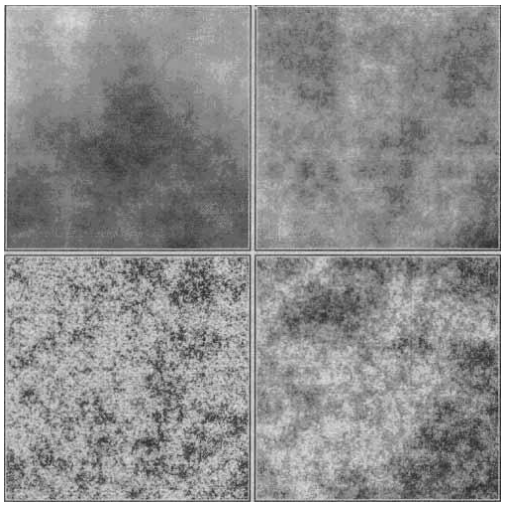
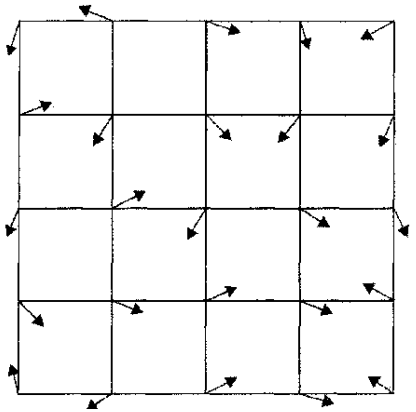



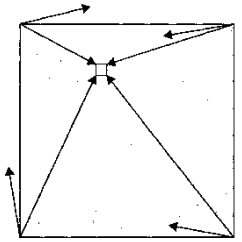
Рис. Карты высоты, созданные методом смещения средней точки (по часовой стрелке, начиная с верхнего левого угла, параметр уменьшения дельты равен 0.65, 0.75, 0.85, 0.95)

# **Шум Перлина** (Perlin Noise)

Введен в 1983 г. **Кеном Перлином (Ken Perlin).** Перлин предложил функцию порождения случайных значений, которая стала основой почти всех фильтров генерации мраморных, деревянных поверхностей и шумов, встроенных в программы рисования и пакеты 3D-рендеринга. Работает для $n$ измерений.

**1.Изображение накрывается сеткой**, представляющей диапазон вещественных чисел. На рис. шум создается на сетке, представляющей значения между 0 и 4. Каждое целое число порождает линию сетки, а значит, все стороны каждого квадрата последней имеют длину, равную единице.
Большее число квадратов на сетке (масштаб) создает более «плотно упакованный» шум.
2.**В каждом узле сетки строится случайный вектор** единичной длины, который указывает в случайном направлении в пределах каждого из квадратов: создается таблица из 256 векторов, которые охватывают полный круг, со случайным выбором одного из них для каждого узла сетки.
3.**Для каждого пиксела** изображения определяется ячейка, где он находится и **строится значение высоты**, которое основано на высотах и векторах узлов этой ячейки и 4-х диагональных векторах, соединяющих углы ячейки с текущим пикселом →

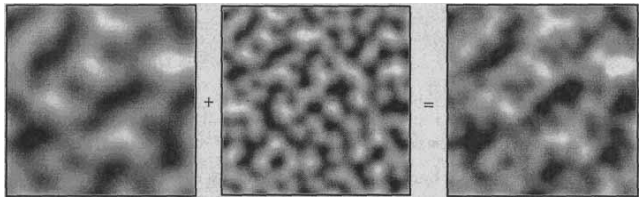


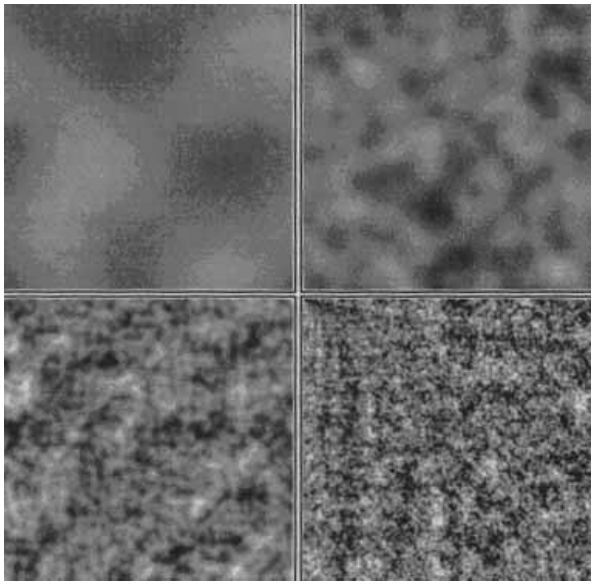


# **Шум Перлина** (Perlin Noise, lab)

4. **Порядок вычисления высоты для пиксела** (x,y):
•вычислить по x и y веса $w_x$ и $w_y$, заменив в формуле $w=6t^5-15t^4+10t^3$, t координатами x и y, найденными относительно вершин сетки;
•смешаем по формуле $h = w \cdot h_a + (1 - w) \cdot h_b$ с весом $w=w_x$ пару значений высот в верхних углах квадрата, затем в нижних и, наконец, смешаем результаты с весом $w=w_y$

**Примеры** изображений шума Перлина, созданные при разных размерах сетки, различном числе октав и разных значениях параметра масштаба →





**Функция выборки случайных узловых нормалей**:
cVector2 RandomGridNormal(unsigned char x, unsigned char y)
{return V[(T[x] + T[y])%256];] };
V – регулярная таблица узловых нормалей;
T – таблица случайных индексов векторов.

# Perlin Noise vertex shader

**Perlin Noise Class, C++**

```
// File: PerlinNoise.vsh
//-----------------------------------------------------------------------------
// Global variables
float4x4 mWorldViewProj;  // World * View * Projection transformation
float fTime;              // Time parameter. This keeps increasing
//-----------------------------------------------------------------------------
// Vertex shader output structure
struct VS_OUTPUT
{
    float4 Position   : POSITION;   // vertex position
    float4 Diffuse    : COLOR0;     // vertex diffuse color
};
//-----------------------------------------------------------------------------
// Name: Perlin Vertex shader

VS_OUTPUT Perlin( in float3 vPosition : POSITION )
{
    VS_OUTPUT Output;
    Output.Position = mul( float4( vPosition.x, vPosition.y, 0.2f*vPosition.z, 1.0f ), mWorldViewProj );
    Output.Diffuse =vPosition.z;
    //Output.Diffuse =float4(0,vPosition.z,0,1.0f);
    return Output;
}
```

# Базовые классы ландшафтной геометрии

**Рендеринг ландшафта в целом как одного гигантского множества треугольников неэффективен, поскольку ландшафт может выходить далеко за пределы видимости камеры во всех направлениях**.
Ландшафт делится на поддающиеся контролю участки (класс TerrainSection). К примеру, для представления ландшафта из 256 x 256 вершин можно создать Terrain, который будет хранить весь набор данных. Затем объект Terrain поделит ландшафт на участки по 32 x 32 вершины и создаст в общей сложности 64 объекта TerrainSection, представляющие каждый из этих участков.

## Индексные буферы в ландшафтной геометрии

Если все объекты TerrainSection имеют одинаковые размеры и содержат одинаковое число вершин, мы можем создать один объект типа «индексный буфер» (IndexBuffer).

## Вершинные буферы в ландшафтной геометрии

В общем случае вероятность найти два участка с одинаковой топографией крайне мала, что не позволяет упаковать эти данные. Если участки будут одинакового размера (например 32x32), то можно существенно улучшить хранение ландшафта и оперирование с ним, можно:
•привести интервалы изменения x,y-координат и u,v-координат текстуры к интервалу [0,1] – общему для всех участков;
•использовать один оптимально построенный индексный буфер для всех участков.
Носителем унифицированных данных является объект класса cTerrain.

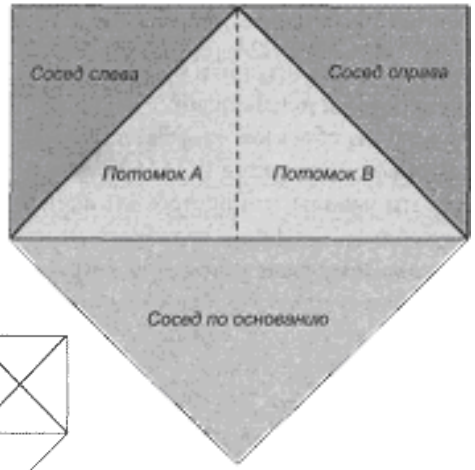
# Ландшафтная система ROAM

Главная цель - больше треугольников на тех участках ландшафта, где это необходимо (вблизи камеры), и меньше - там, где острой необходимости в этом нет (на более отдаленных участках).

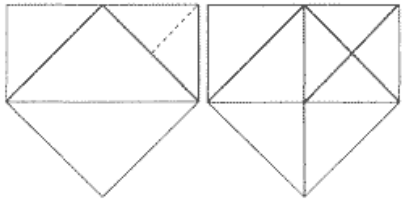
**Мэши с оптимальной подгонкой в реальном времени**
**Real-Time Optimal Adapting Mesh**

Алгоритм Real-Time Optimal Adapting Mesh больше известен как ROAM. Впервые представлен в работе Марка Дюшено (Mark Duchaineau) и др. в 1997
[M.Duchaineau, M.Wolinski, D.Sigeti, M.Miller, C.Aldrich, M.Mineev-Weinshtein “ROAMing Terrain: Real-Time Optimally Adapting Meshes” www.llnl.gov/graphics/ROAM]

1.Всякий треугольник должен не только знать о дочерних узлах на нижележащем уровне, но и о трех соседях, смежных с ним по его сторонам. Связи треугольников между собой показаны на рис.→
2.Если сосед по основанию не имеет общей с разбитым треугольником гипотенузы, он должен рекурсивно подвергаться делению до тех пор, пока не образуется треугольник, смежный по гипотенузе с тем, который мы разбиваем.




3.Выполняется процедура пересоздания индексов рекурсивным спуском по дереву до каждого листового узла, учитывающая расстояние до вершин. Используется межкадровая когерентность.



# Методы мозаичной геометрии (агрегатные)

**Недостатки ROAM**: 1. Необходимость построения динамических индексных буферов. Перестройка и загрузка такого буфера в видеокарту на каждом кадре существенно сказывается на производительности. 2. Бинарное дерево треугольников затрудняет построение strips & fans, позволяющих аппаратуре использовать когерентность кэш-памяти.

**Альтернативой являются агрегатные методы** управления ландшафтной геометрией, рассматривающие LOD сразу для групп треугольников. В качестве групп принимаются квадраты на регулярной сетке ландшафта. Это позволяет заранее строить множество геометрических форм, отвечающих разным LOD. **Достоинство:** меньше объем реальных расчетов; предварительно созданные геометрические конструкции можно строить с учетом образования strips & fans.

**Два агрегатных метода для работы с ландшафтом в приращениях сетки (grid increments):**
1)Chunked Terrain (Блочный ландшафт), автор – Thatcher Ulrich из Oddworld Inhabitants®. Метод использует ячеистую структуру, напоминающую квадродеревья.
2)Метод, созданной Snook-ом, - использует единую сетку ячеек, каждая из которых может отображаться с заранее рассчитанным LOD.
Общая проблема методов – **скрыть щели**, которые возникают между блоками → оба метода предусматривают дополнительные геометрические построения для сокрытия или устранения щелей между блоками с разным уровнем детализации.

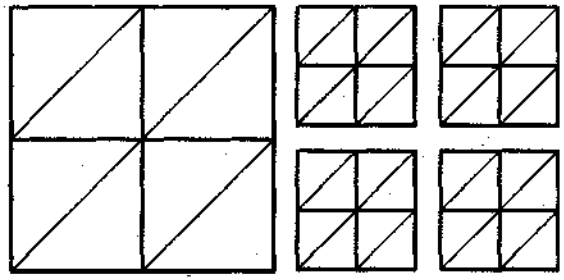
# Блочный ландшафт

Идея разбить ландшафт на квадратные блоки по регулярным линиям сетки - прекрасный ход, упрощающий бинарное дерево треугольников из ROAM. Там, где ROAM пользовался метриками ошибок для рекурсивного спуска по бинарному дереву на искомую глубину, Chunked Terrain использует подобную метрику для перехода на нужную глубину квадродерева.

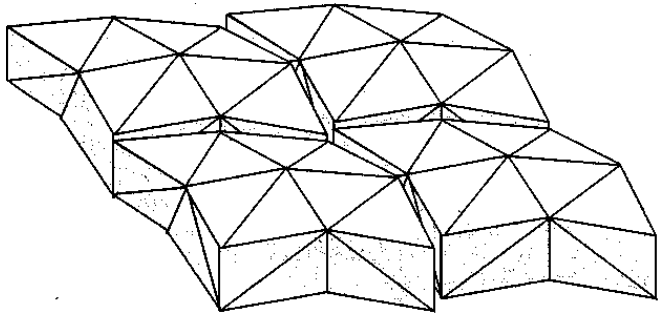
•на верхнем уровне создан один квадрат, образованный двумя треугольниками

•второй уровень дерева содержит четыре члена, каждый из которых задает более высокий уровень детализации, чем его прямой предок

•конструкция повторяется до тех пор, пока лежащая в основе карта высот не будет с заданной точностью представлена треугольниками (уточнение высоты станет меньше заданного).



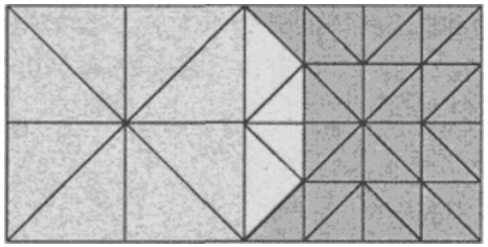
Вместо принудительного деления соседей Ульрих → предлагает новый подход, использующий полигональную кромку вокруг каждой отображаемой ячейки на сетке



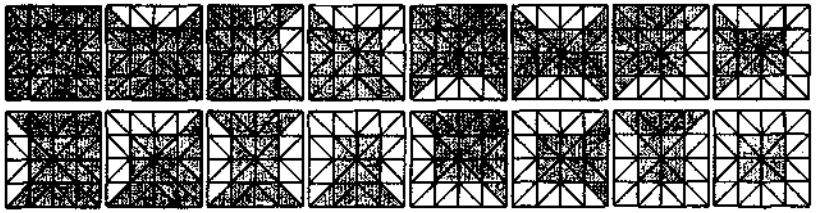
**Недостаток:** происходит удвоение объема вершинного и индексного буферов + индексы треугольников кромки.

# Взаимосвязанная (сцепленная) ландшафтная мозаика. Snook

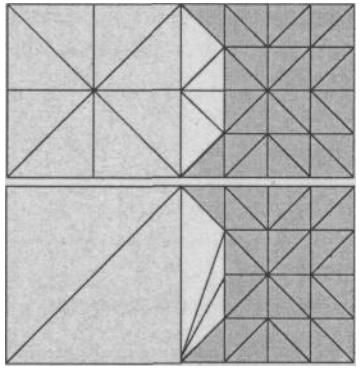
Два элемента ландшафта вкупе с объединяющим их фрагментом геометрии связи →



16 основных форм, необходимых для представления основы любого ландшафта. Открытые зоны - это пространство, где могут размещаться связующие фрагменты, соединяющие смежные ячейки сетки →



Два варианта одного связующего фрагмента. Каждый из них предназначен для соединения со смежным меньшим уровнем LOD.→



Лучший способ справиться с артефактами всех трех методов - анимация, или временной морфинг изменений, наблюдаемых на границе различных LOD.

**Реализация морфинга в методе взаимосвязанных ландшафтных мозаик представляется нереальной**.

Дополнительный материал.

# Методы текстурирования ландшафтов

Будем говорить о нескольких текстурах: 1)траве, 2)грунте, 3)скальной породе, 4)снежном покрове. Если число доступных текстур в расчете на один пиксельный шейдер окажется существенно ограничено, то мы можем еще сильнее сузить этот набор.

Ландшафт в один  квадратный километр, потребовал бы при масштабе 1 пиксел на квадратный дециметр использовать текстуру в 10000x10000 пикселов! **Наша цель** - создать ландшафт, который растянется на километры. → Покрытие множеством карт с текстурой, а не одной большой картой; генерация текстур. Главная сложность: существенная неоднородность и неповторимость текстуры, которую реально нужно генерировать.

**Комбинирование текстур**

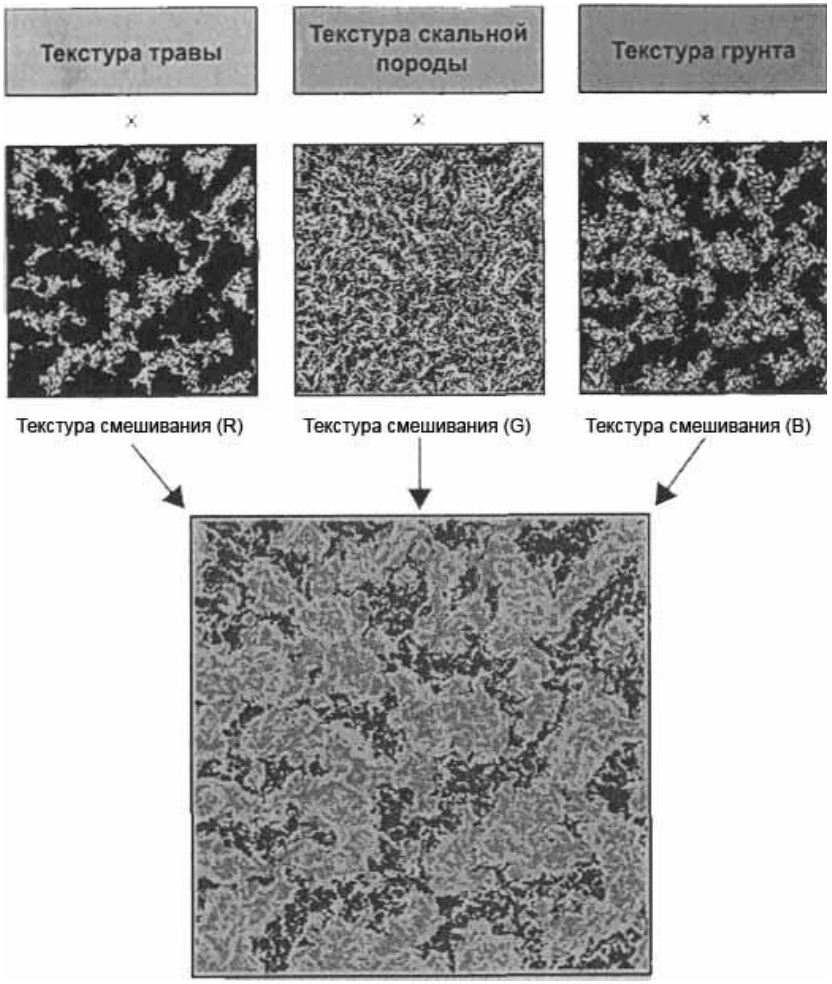
Покрытие всего ландшафта одним и тем же изображением - бесспорный фаворит в плане потенциального качества и отношения числа пикселов в текстуре и на экране, необходимого объема памяти и простоты применения. Если мы сможем включить в него работу с поверхностями различных типов, то получим более привлекательный метод.

Представьте, что мы создали отдельные слои ландшафта, один из которых будет покрыт грунтом, другой - скальной породой, а третий - травой. Каждый из них - это всего лишь одна текстура, которая повторяется до бесконечности, но их смесь дает очень органичные результаты. Идея состоит в том, чтобы на разных участках ландшафта обеспечить неодинаковое отображение этих слоев. Для ее реализации нужно средство управления долей каждого слоя в окончательном результате.

# Комбинирование текстур

Используем взвешенное среднее, рассчитанное для каждого пиксела. Припишем каждому слою весовое значение от 0.0 до 1.0, определяющее, в какой мере этот слой виден в данной конкретной точке, так, чтобы сумма весов по всем слоям = 1.

Графически это показано на рис. →
Здесь представлены четыре входные текстуры. Среди них - текстуры поверхностей трех типов и текстура, управляющая смешиванием изображений.
В управляющей текстуре коэффициент видимости каждой ландшафтной поверхности закодирован красным, зеленым и синим каналами соответственно.
Шейдер смешивания



# Генерация текстуры смешивания

**Параметры смешивания текстур**

```
struct  elevationData
{
float minElevation; //  низшая высотная отметка
float maxElevation; //  высшая высотная отметка
float minNormalZ;   //  мин. z-координата нормали к поверхности
float maxNormalZ;   //  макс.z-координата нормали к поверхности
float strength;     //  "интенсивность" (приоритет применения)
};
```

Структура elevationData дает возможность каждой ландшафтной поверхности указать свой минимум и максимум как в диапазоне высот, так и в диапазоне значений наклона. Наклон измеряется z-составляющей нормали к поверхности. Задание этих значений, в [0.0 , 1.0], позволяет отобразить поверхность ландшафта в хребты скал (малые значения z) и участки равнины (большие значения z).

Класс cTerrain содержит функцию-член для создания текстуры смешивания, используя в качестве входа до четырех структур elevationData (по 1 для каждого типа ).
Расчет весовых значений основан на минимальных и максимальных значениях диапазона, заданных для наклона и высоты.

Дополнительный материал.

# **Генерация текстуры смешивания.** Пример

# **Природа и шум.** Многопроходные техники

## Природа и шум

Ограничением однопроходной ландшафтной системы является четыре текстуры на проход --> чтобы сохранить совместимость с устаревшими видеокартами.
Простейший способ побороть периодическую природу текстур наших поверхностей - ввести некий случайный шум. Так использован четвертый канал нашей текстуры смешивания.

Ландшафтному шейдеру, чтобы просто смешать изображения поверхностей, нужно три канала: четвертый  альфа-канал мы заполняем случайным шумом. Альфа-значение служит для модуляции итоговых цветов, которые создает шейдер. Чтобы усилить этот эффект, можно расширить диапазон случайных значений.

## Многопроходные техники

На картах с поддержкой более четырех текстур на проход мы можем смешать больше типов поверхностей и ввести больше случайностей в итоговый результат.
Но даже при ограничении в четыре текстуры на 1 проход мы можем затем смешать набор из четырех текстур, если перейдем на многопроходный рендеринг. Дополнительные проходы понизят общую скорость рендеринга, но результат окупит издержки. Во втором проходе добавим в ландшафт чуть-чуть снега.

Можно подумать о создании графической накладки на весь ландшафт. Подобную текстуру мы могли бы использовать для нанесения на ландшафт тени от облаков.

# Многопроходные техники

В программе мы введем в текстуру четыре отдельные картины распределения шума. Это позволит добавить к траве, породе, грунту и снегу тот шум, который лучше подходит к каждому типу поверхности. В случае с травой воспользуемся точечно-пунктирным шаблоном, который отражает вид отдельных травинок. На скалы нанесем более крупный, ямчатый шум. Для грунта и снега будут подобраны такие шумовые каналы, которые подходят именно для этих поверхностей.

В пиксельном шейдере (версии 1.1) выполним два прохода. На первом отобразим грунт и траву, на втором - скалы и снег. При каждом проходе для усиления полученного в итоге цвета используем отдельные каналы для текстуры смешивания и для текстуры шума. Код пиксельного шейдера показан в листинге.

# Дополнительные разделы

В материалах к лекции представлены также следующие дополнительные методы моделирования на основе шейдеров (в том числе шейдеров написанных на ассемблере):

1. Моделирование окружающей среды: метод окружающего куба; моделирование движущихся и уходящих за горизонт облаков; линзовые эффекты (doc)
2. Моделирование поверхности воды (doc).
3. Моделирование меха (doc).
4. Моделирование полупрозрачности и радужных переливов на тонких поверхностях (Simulation of Iridescence and Translucency on Thin Surfaces, pdf)
5. Шейдер многослойной визуализации поверхности автомобиля (Layered Car Paint Shader, pdf)

# Источники


1. Снук Г. Создание 3D-ландшафтов в реальном времени с использованием C++ и DirectX9/ Пер. с англ. –М.: КУДИЦ-ОБРАЗ, 2006. -368с.
2. Ulrich, Thatcher. «Chunked LOD» (работа доступна по адресу http://tnlrich.com/ geekstuff/ chunklod.html).
3. MS HLSL. Light Models. Bump-mapping// GameDev.ru: http://www.gamedev.ru/articles/?id=10109
4. Natalya Tatarchuk, Chris Brennan (ATI). Simulation of Iridescence and Translucency on Thin Surfaces. 2002 (pdf, local)
5. Chris Oat, Natalya Tatarchuk, John Isidoro (ATI). Layered Car Paint Shader . (http://www.ati.com/developer/) (pdf, local)
6. Евченко Александр. OpenGL и DirectX. Программирование графики. –СПб: Питер, 2006. -350с.
7. Андре Ламот . Программирование трехмерных игр для Microsoft Windows. Советы профессионала по трехмерной графике и растеризации: пер. с англ.- М.: Издат.дом «Вильямс», 2006. -1424 с.