Компьютерная Графика

Текстуры

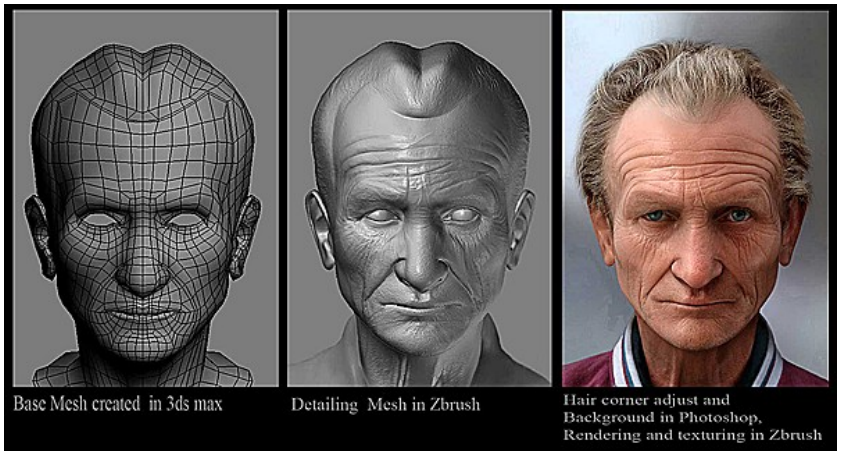
Текстуры

Текстура — это картинка, графическое изображение, которое наносится на объекты.

Благодаря текстурам, мы отличаем песчаную дорожку от асфальтированной, деревянный ящик от каменной глыбы, кирпичную стенку от скалы и т. д.

Текстуры

Позволяет воспроизвести малые объекты поверхности, создание которых полигонами оказалось бы чрезмерно ресурсоёмким. Например, шрамы на коже, складки на одежде, мелкие камни и прочие предметы на поверхности стен и почвы.



Пикселы и текселы

- **pixel**, or **picture element**

минимальная единица двумерного цифрового изображения, а так же элемент матрицы дисплеев

- **texel**, or ***tex****ture* ***el****ement* (also ***tex****ture pix****el***)

минимальная единица текстуры трёхмерного объекта.

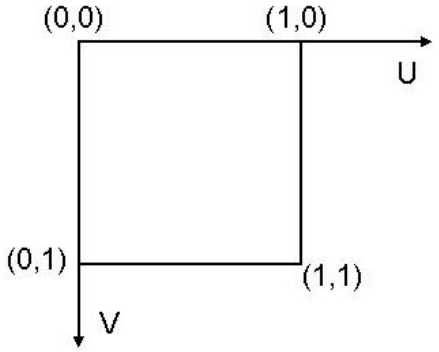
- Для идеального отображения текстуры количество текселов должно совпадать с количеством пикселов

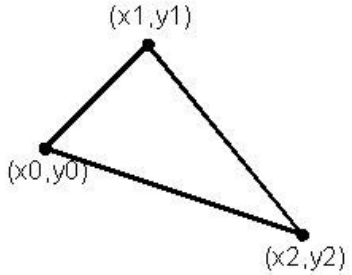
Количество текселей и пикселей **совпадает**

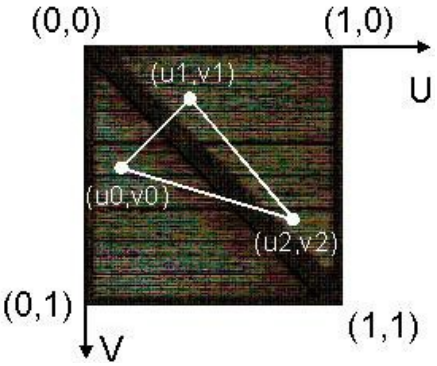
Увеличенное изображение
Один тексель отображается во много пикселей монитора

Текстурные координаты

- Процесс наложения текстуры на объект называют **текстурированием** или **мапированием** (от англ. **mapping**).
- Текстуры накладываются на объект с помощью так называемых **текстурных координат** (**texture coords**).



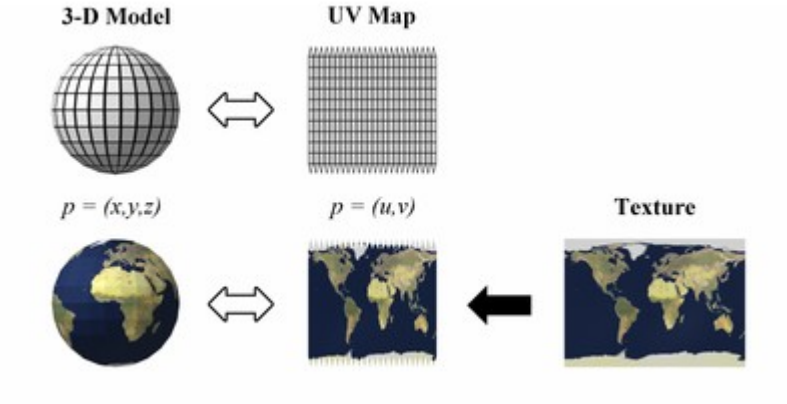




- Пара величин (u,v) однозначно указывает на элемент текстуры, называемый текселем.

Текстурные координаты

- **UV-преобразование** в трёхмерной графике (англ. ***UV map***) — соответствие между координатами на поверхности трёхмерного объекта (X, Y, Z) и координатами на текстуре (U, V).
- Развёртка может строиться как вручную, так и автоматически.



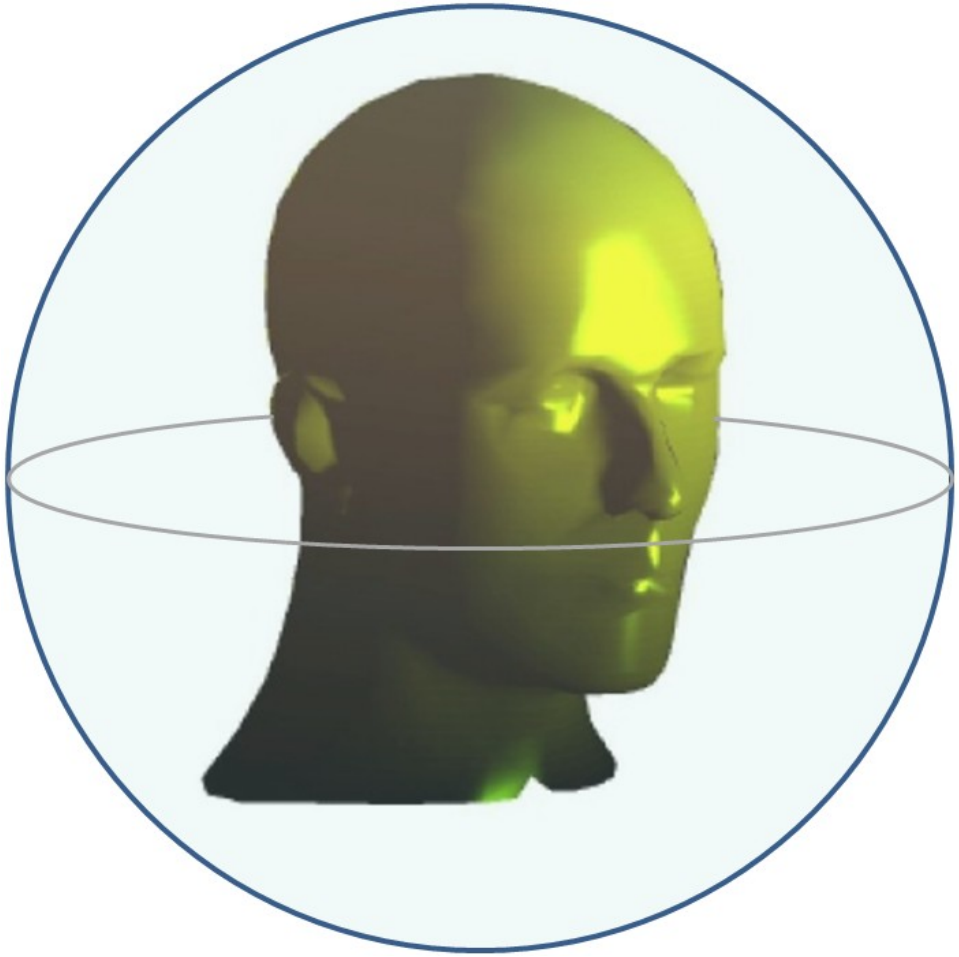
Текстурные координаты

Привязываются к каждой вершине сетки
Обычно обозначаются s, t, r, q
К текстурным координатам можно применять матричные преобразования
Правила повторения или обрезания (clamp/repeat)
Явное задание:

```
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0); glVertex3f(-1.0,-1.0,-1.0);
glTexCoord2f(0.0,1.0); glVertex3f(-1.0, 1.0,-1.0);
glTexCoord2f(1.0,1.0); glVertex3f( 1.0, 1.0,-1.0);
glTexCoord2f(1.0,0.0); glVertex3f( 1.0,-1.0,-1.0);
...
glEnd();
```

Проецирование текстурных координат

- Вместо задания каждой координаты объект мысленно проецируется на сферу, цилиндр или плоскость.
- Координаты в соответствующей системе координат используются в качестве текстурной:
  - Плоскость — x, y
  - Цилиндр — φ, z
  - Сфера — φ, θ

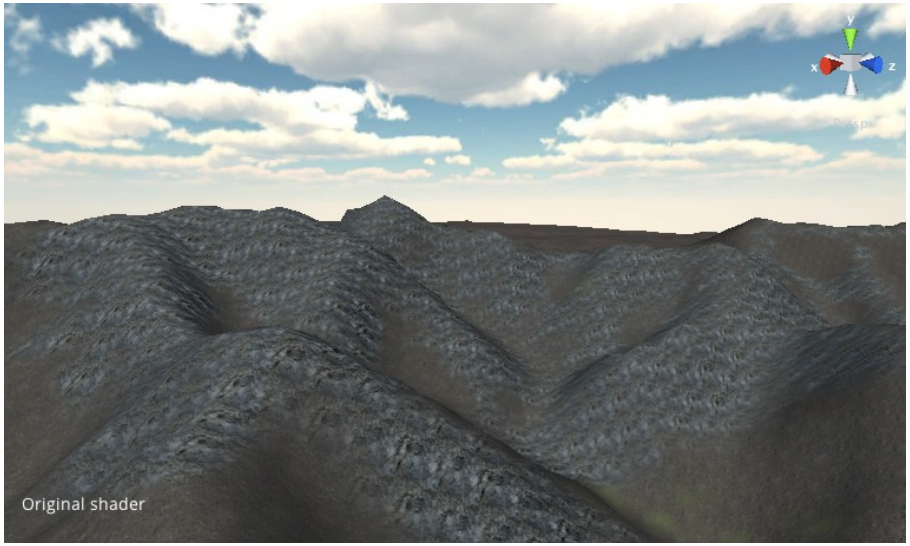


Тайлинг (Tiling)

- вид наложения текстур на объект «по типу поклейки обоев с рисунком»

Швов склейки не видно)

Тайлинг (Tiling)



Размерность текстур

Одномерные

Как двумерные, но в один пиксель высотой

Двумерные

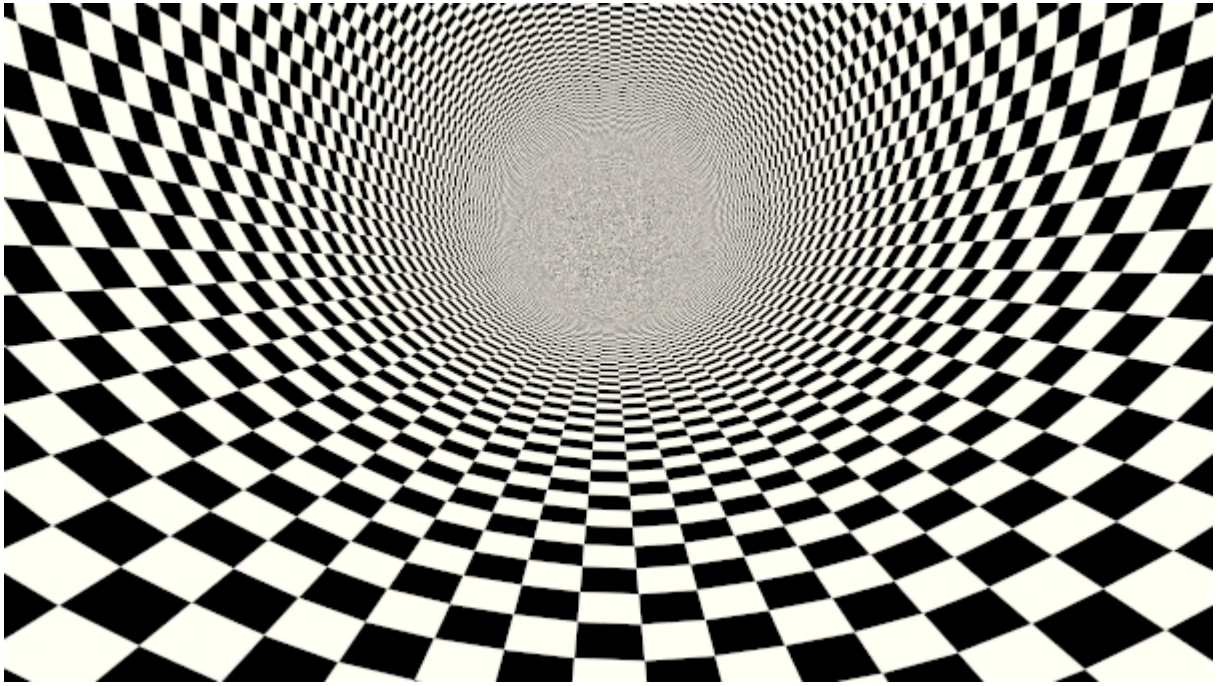
Трёхмерные

Объёмные плотности, томограмма и т. д.

256×256×256×4=64 Мб

Масштабирование

- Вдалеке много текселей претендуют попасть в 1 пиксель экрана



Масштабирование

Одному пикселю на экране может соответствовать как много меньше, так и много больше одного пикселя текстуры

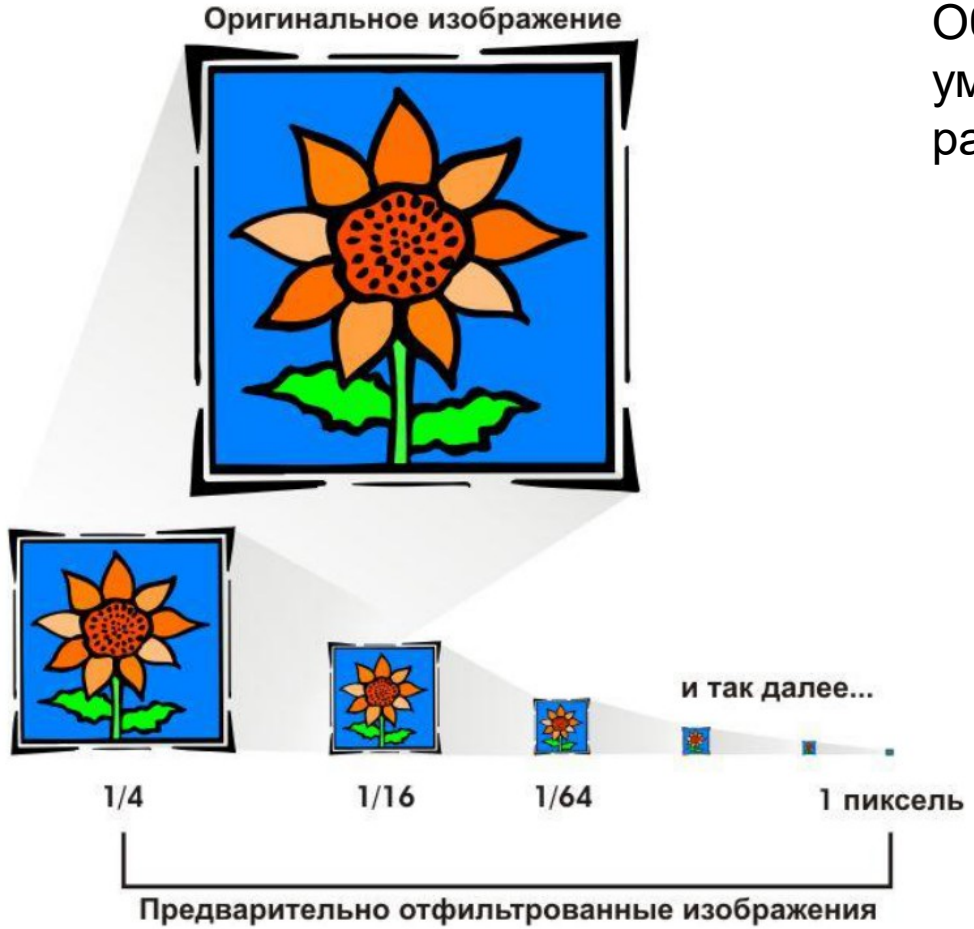
Увеличение и уменьшение

- Взять ближайший пиксель (nearest)
- Билинейно интерполировать четыре ближайших пикселя (linear)

Масштабирование

- Изображение лучше всего выглядит, когда детализация текстуры близка к разрешению экрана.
- Если разрешение экрана высокое (текстура слишком маленькая/объект очень близко), получается размытое изображение.
- Если же разрешение текстуры слишком высокое (текстура слишком велика/объект очень далеко), получаем случайные пиксели — а значит, потерю мелких деталей, мерцание.
- **Получается, что лучше иметь несколько текстур разной детализации и накладывать на объект ту, которая наиболее подходит в данной ситуации.**

Несколько уровней детализации текстуры (mipmap)

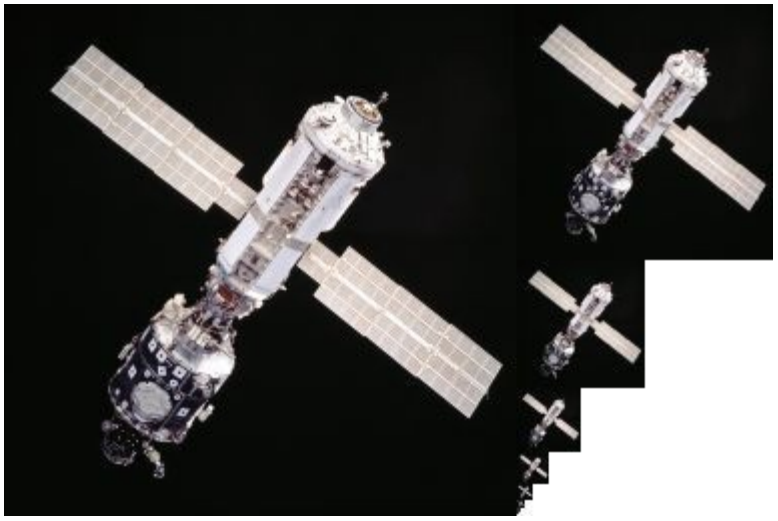


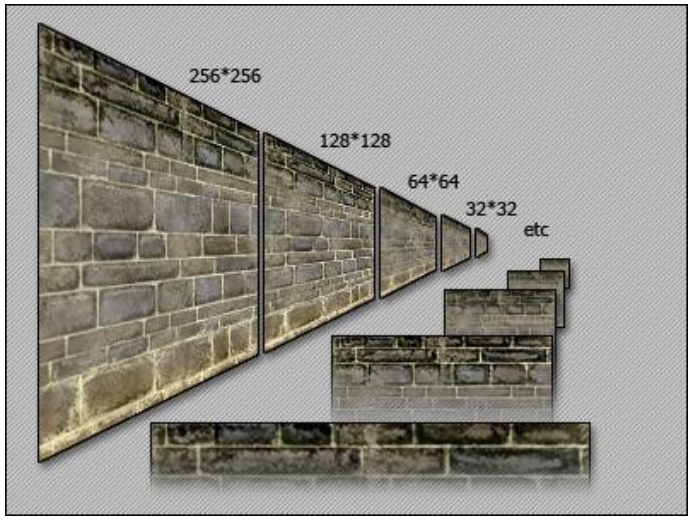
Обычно генерируются с уменьшением линейных размеров вдвое:

256×64
128×32
64×16
32×8
16×4
8×2
4×1
2×1
1×1

MIP-текстурирование (MIP mapping)

- метод текстурирования, использующий несколько копий одной текстуры сразной детализацией.
- Название происходит от лат. *multum in parvo* — «много в малом».
- Обычно следующий уровень детализации меньше в два раза по разрешению к предыдущему (256x256, 128x128, 64x64, …)





MIP-текстурирование (MIP mapping)

- Изображение лучше всего выглядит, когда детализация текстуры близка к разрешению экрана.
- Если разрешение экрана высокое (текстура слишком маленькая/объект очень близко), получается размытое изображение.
- Если же разрешение текстуры слишком высокое (текстура слишком велика/объект очень далеко), получаем случайные пиксели — а значит, потерю мелких деталей, мерцание.
- **Получается, что лучше иметь несколько текстур разной детализации и накладывать на объект ту, которая наиболее подходит в данной ситуации.**

MIP-текстурирование (MIP mapping)

Границы применения:

- Расход видеопамяти увеличивается натреть
- Не решает проблему текстур, находящихсяпод острым углом к зрителю - видна чёткая граница между MIP-уровнями.

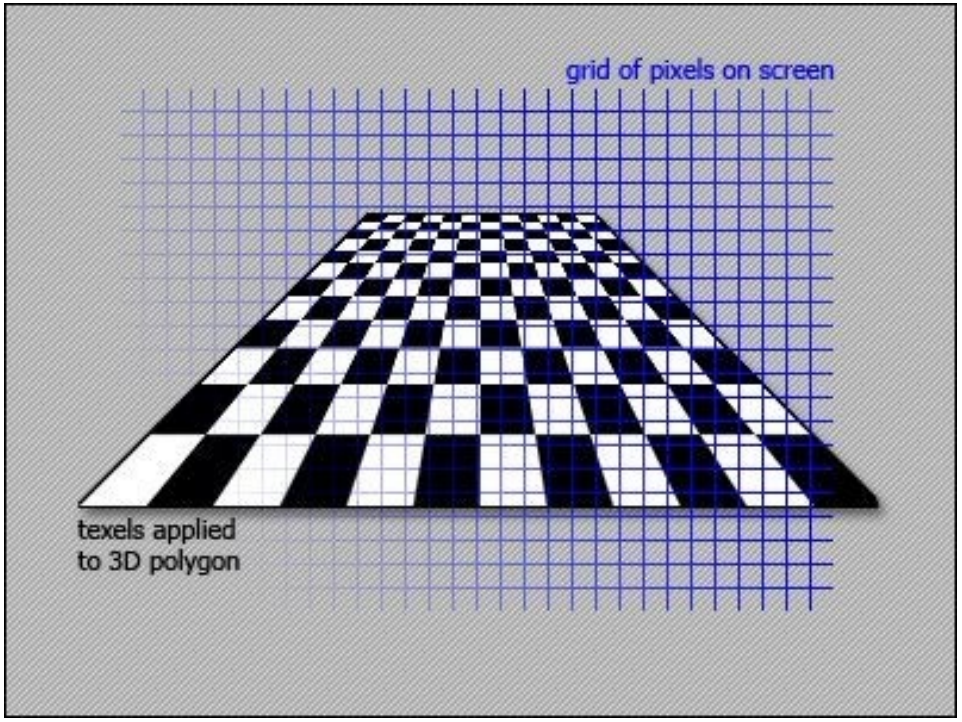
Однако с ним лучше, чем без него

Уменьшение с использованием mipmaps

Вычисляем ширину (высоту, площадь) пикселя в текстурных координатах и определяем коэффициент масштабирования текстуры
Выбираем ближайший пиксель из ближайшего по масштабу мипмэпа (nearest-mipmap-nearest)
Интерполируем четыре пикселя из ближайшего по масштабу мипмэпа (nearest-mipmap-linear)
Выбираем ближайший пиксель из двух ближайших по мастшабу мипмэпов и интерполируем их (linear-mipmap-nearest)
Интерполируем по четыре пикселя из двух ближайших по мастшабу мипмэпов и интерполируем результат (linear-mipmap-linear)

Фильтрация

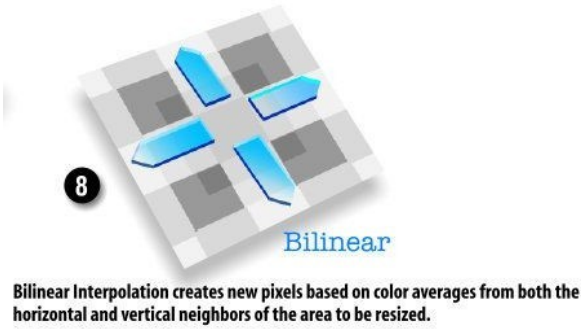
- Фильтрация текстур необходима, т.к. позиции текселей обычно не совпадают с пикселями экрана.
- Какой из текселей выбрать для отрисовки пиксела ?



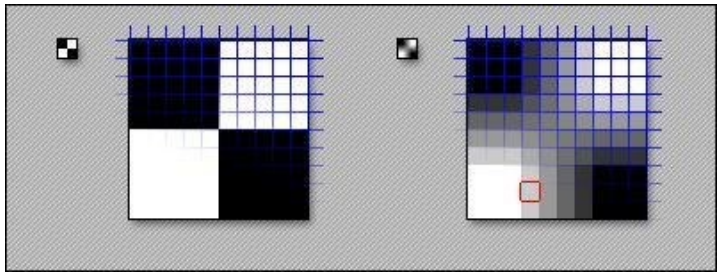
Фильтрация. Билинейная

Среднее значение от суммы четырех соседних текселей

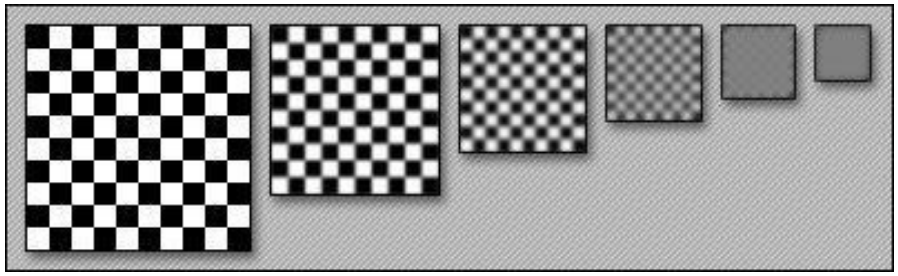
Увеличенное изображение без фильтрации



После применения билинейного фильтра



Работа билинейного фильтра



Уменьшение изображение с применением билинейного фильтра

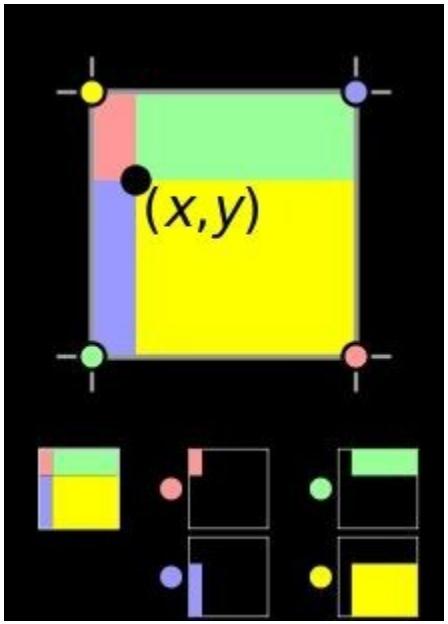
Фильтрация. Билинейная

Границы применения:

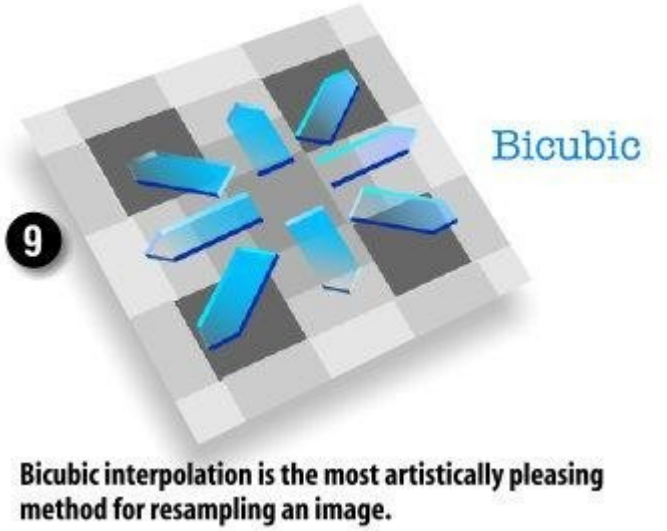
- Хорошо подходит для отображения, когда на отрисовку каждого пиксела претендуют [0.5, 2] тексела.
- Т.е. исходное изображение уменьшено в два раза, или увеличено в два раза
- За границами применимости происходит потеря текселей или размытие текстуры

Фильтрация. Билинейная

Модификации билинейной фильтрации



Можно складывать цвета соседних текселей, умноженные на их площадь



Бикубическая фильтрация использует 8 соседей

Фильтрация. Трилинейная

- Билинейная фильтрация + MIP mapping
- цвет пикселя = средневзвешенное восьми текселей: по четыре на двух соседних MIP-текстурах.
- В случае, если формулы MIP-текстурирования дают самую крупную или самую маленькую из MIP-текстур, трилинейная фильтрация вырождается в билинейную.

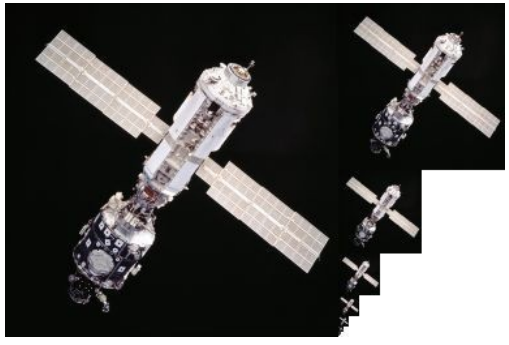
Фильтрация. Трилинейная

Фильтрация. Анизотропная

- метод улучшения качества изображения текстур на поверхностях, сильно наклонённых относительно камеры
- вносит меньше размытия и поэтому позволяет сохранить бóльшую детальность изображения

Фильтрация. Анизотропная

(Билинейная или трилинейная фильтрация)
+ RIP mapping (развитие MIP mapping)



x2 x4 x8

x2

x4

x8

MIP mapping
256x256,
128x128,
…

RIP mapping
256x256, 256x128, 128x256, …
128x128, 128x64, 64x128, …
…

Фильтрация. Анизотропная

Границы применения:

- Требует больше видеопамяти и скорости её работы (десятки гигабайт в секунду)
- Столь большие требования кпамяти уменьшают за счёт сжатия текстур и кэширования

Взаимодействие с материалом

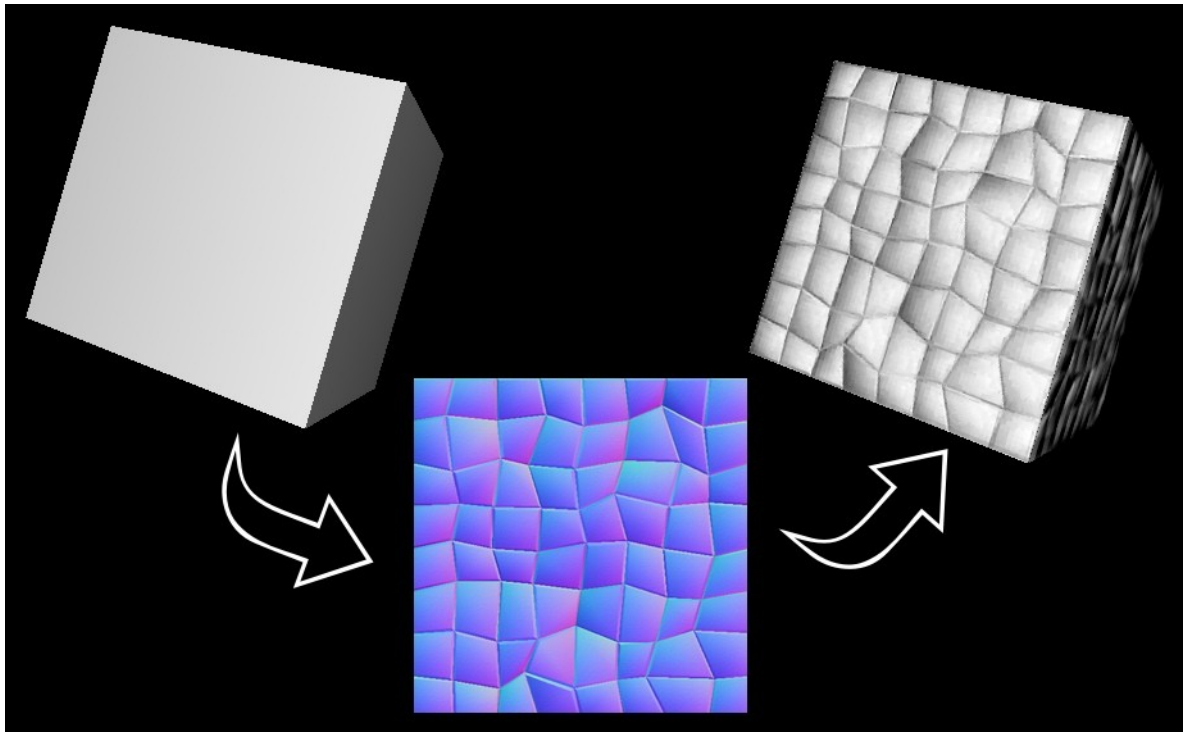
- Игнорировать материал; использовать только цвет текстуры (расчёт освещённости выключен)
- Смешивать рассчитанный цвет цветом текстуры (blend)
- Модулировать (умножать) цвет цветом текстуры
- Ограничиться обработкой части компонент:
  - Только яркость
  - Только RGB
  - Только альфа
  - Альфа и яркость

Компьютерная Графика

Normal mapping

Модуляция нормалей

Простой способ создания эффекта рельефной поверхности с детализацией большей, чем позволяет полигональная поверхность.

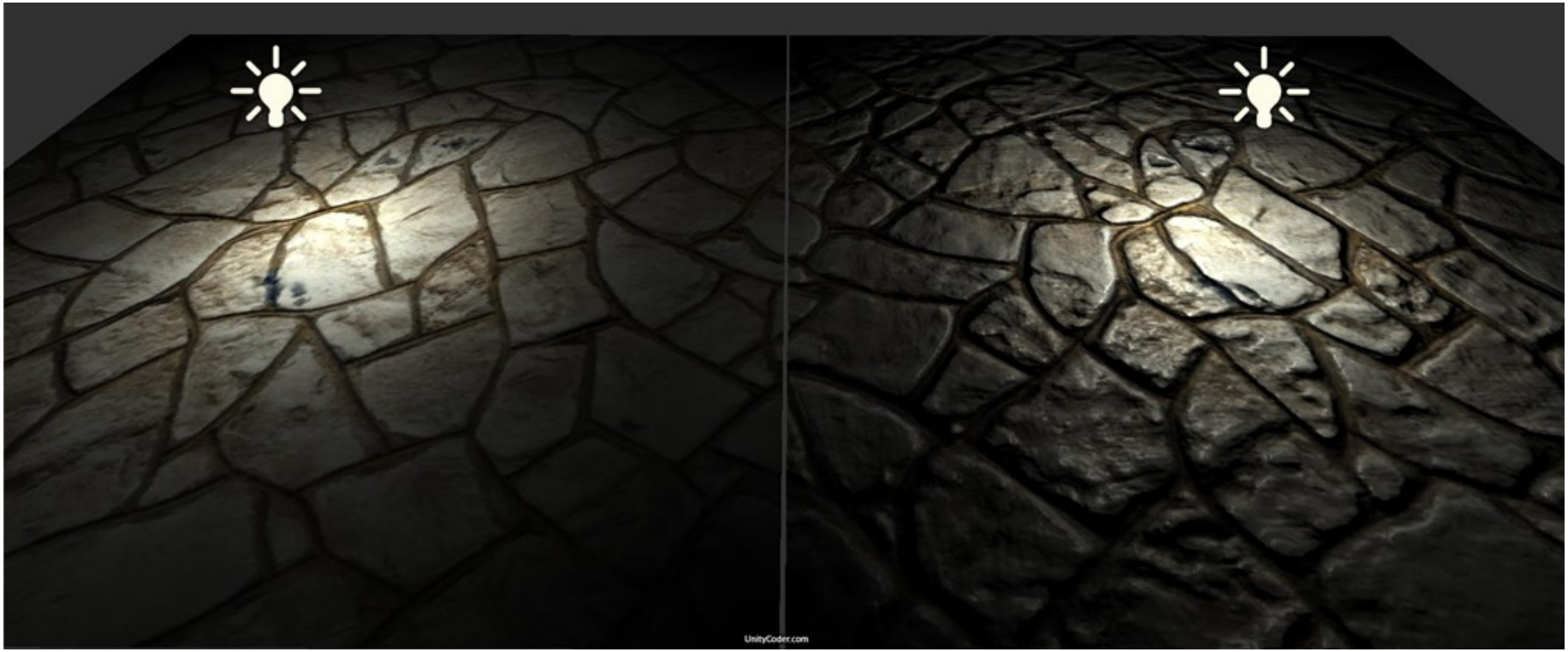


Модуляция нормалей

Позволяет низкополигональной модели выглядить высокополигонально (с хорошей детализацией)

Модуляция нормалей

Обязательное условие: эффект достигается за счет освещения поверхности источником света

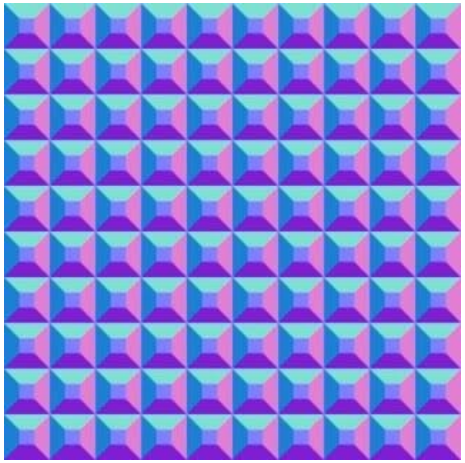


без карты нормалей

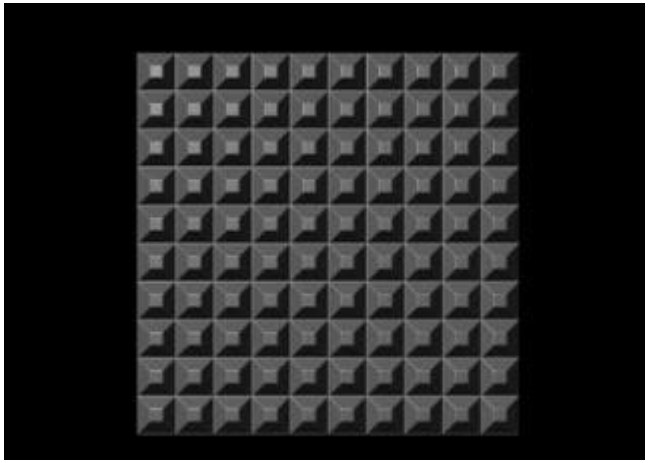
с картой нормалей

Основная идея

- Возьмём обычную плоскость (2 треугольника)
- В каждой вершине нормаль одинаковая - перпендикулярна плоскости
- При отрисовке будем брать нормаль не из вершины, а из специальной текстуры – карты нормалей



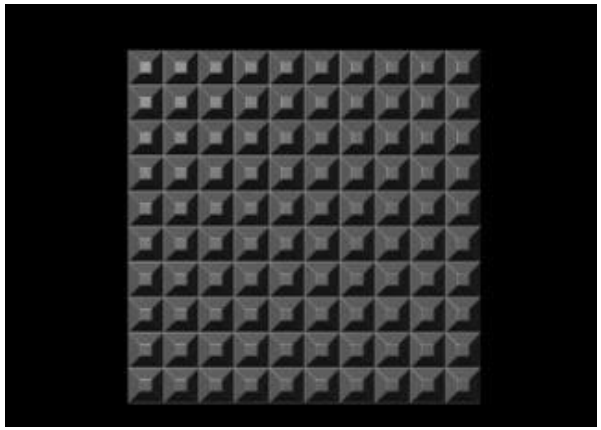
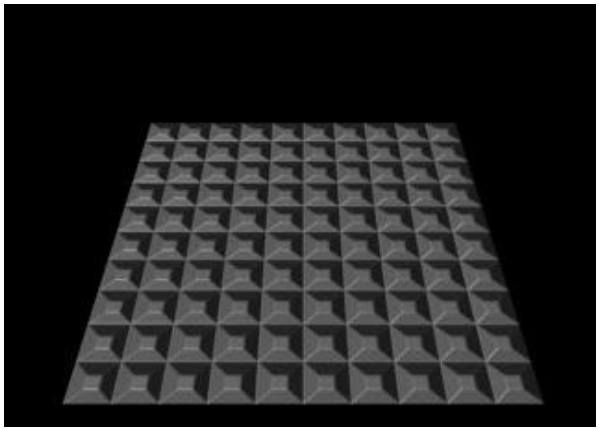
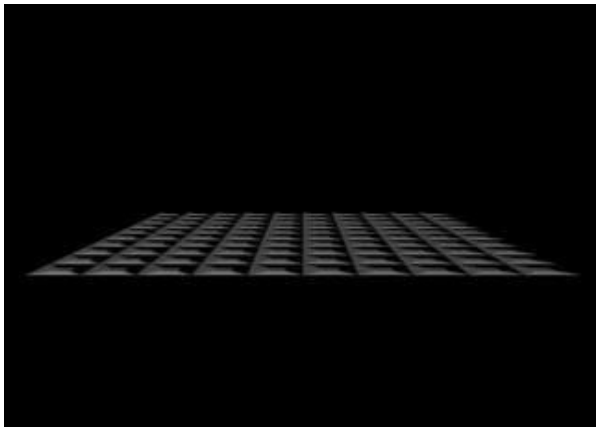
карта нормалей



результат

Основная идея

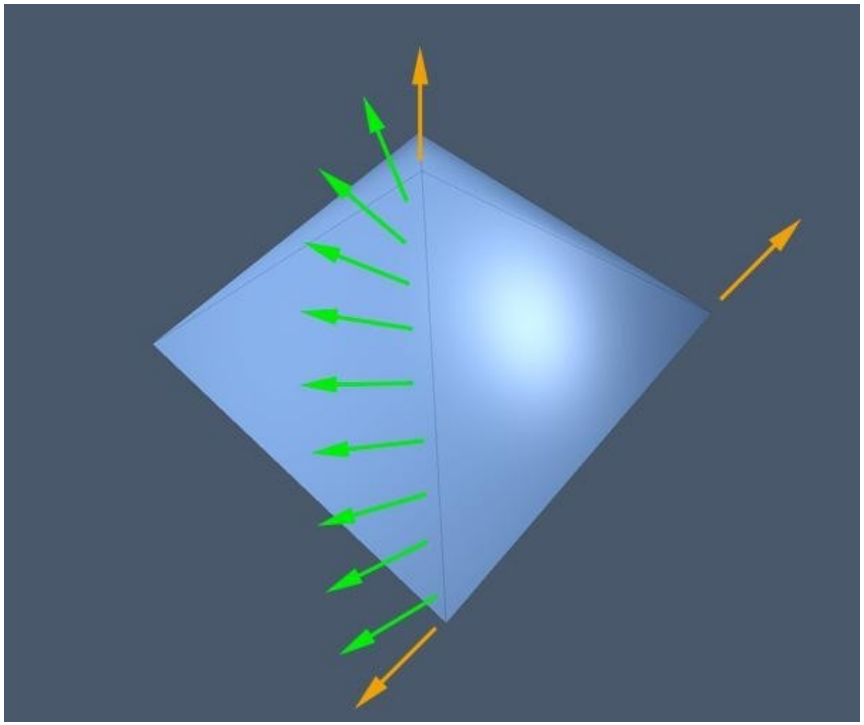
- На отрисовку объекта влияет положение источников света и камеры
- При большом наклоне камеры к поверхности видно, что это плоскость
- Т.е. на видеокарте не генерируются новые треугольники (их всего два) - идёт игра с нормалями, причём в пиксельном шейдере.

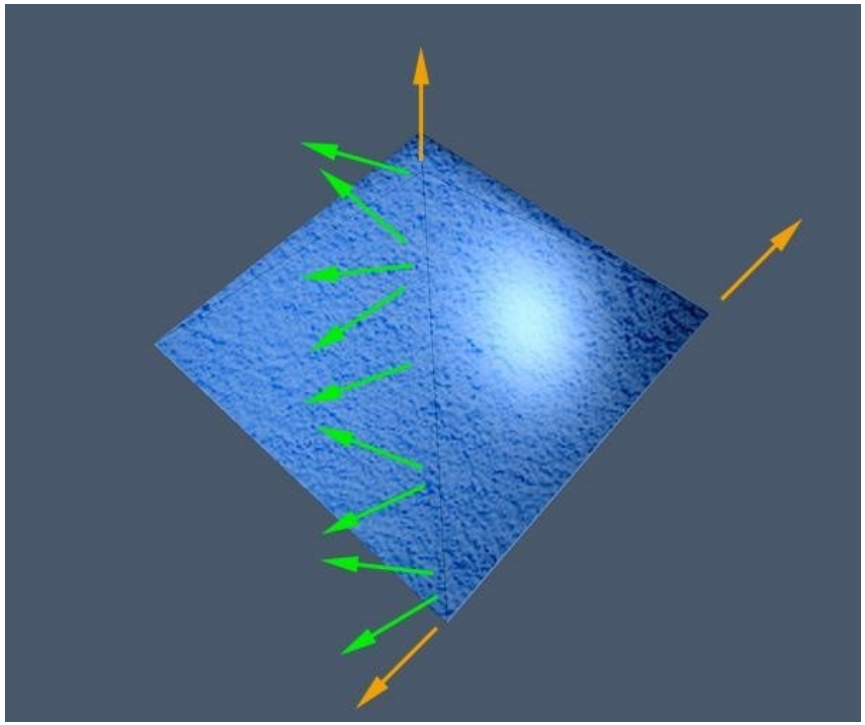




Основная идея

Ещё раз:

- Если брать нормаль из вершины, то получается плоскость
- Если брать нормаль из карты нормалей, то получается рельефная поверхность





Карта нормалей

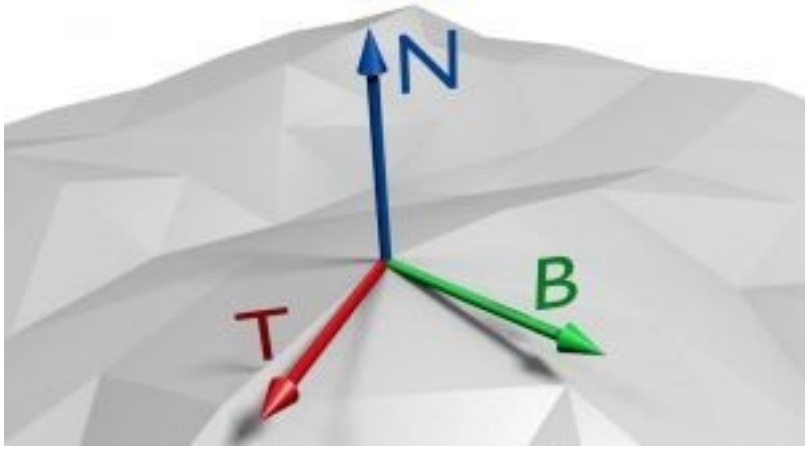
В ней хранится нормаль, причём в специфичной виде:

- т.к. это текстура, то в ней хранится цвет (R,G,B)
- (R,G,B) в диапазоне [0.0, 1.0]
- нормаль имеет (x,y,z) в диапазоне[-1.0, 1.0]
- значит (x,y,z) = (R,G,B) * 2 - 1
- и наоборот: (R,G,B) = ((x,y,z) - 1) / 2

Нормаль, которую можно извлечь из карты нормалей, ((x,y,z) = (R,G,B) * 2 - 1) указана в специальной системе координат- **tangent space**

Tangent space

- Локальная сиситема координат
  (для каждой вершины - своя)
- Строится на основе нормали вершины:
  N (Normal) - нормаль вершины
  T (Tangent) - касательная
  B (Binormal) - бинормаль
- Все вектора единичные и ортогональны друг другу



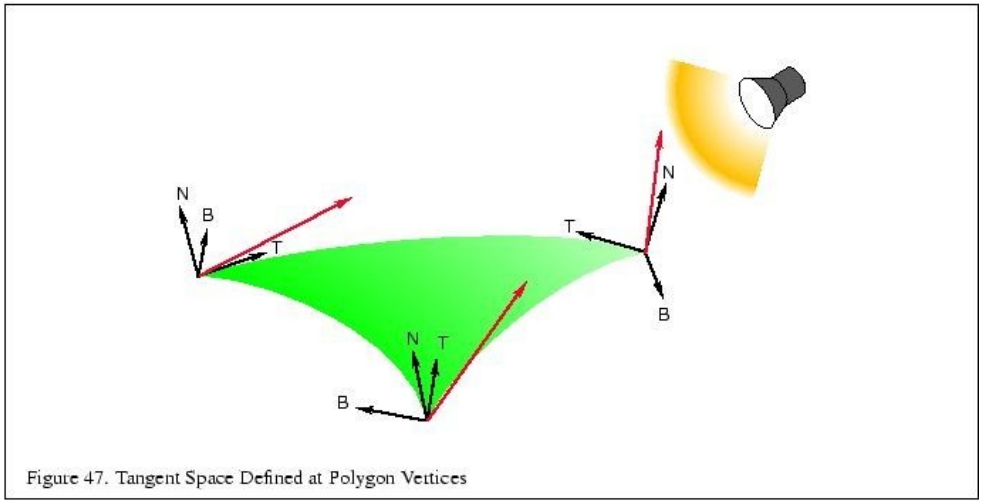


Figure 47. Tangent Space Defined at Polygon Vertices

Tangent space

Пусть:

- $\mathbf{n_{tex}}$ – нормаль из карты нормалей $\mathbf{n_{tex} = (R,G,B) * 2 - 1}$
- **n** – нормаль из вершины (в локальной с.к.)
- **t** – касательная (любой вектор, ортогональный к нормали вершины) **t = cross(n, float3(1,1,1))**
- **b** – бинормаль (вектор, ортогональный к нормали и касательной) **b = cross(n, t)**

  (функция cross() доступна в шейдерах)

- $\mathbf{n_{res}}$- результирующая нормаль (для расчёта освещения после перевода ее в глобальную систему координат)

  $\mathbf{n_{res} = n_{tex}.x * t + n_{tex}.y * b + n_{tex}.z * n}$

Шейдер GLSL

```
uniform sampler2D textureNormalMap;

vec3 calcNormalFromNormalmap(vec3 localNormal, vec2 uv)
{
	// Рассчитываем базисные вектора с.о. нормали (tangent space)
	vec3 n = normalize( localNormal.xyz );
	vec3 t = normalize( cross(n, vec3(1,1,1)) );
	vec3 b = cross(n, t);

	// Достаём нормаль из карты высот
	vec3 normal = texture( textureNormalMap, uv ).rgb;
	normal = normalize( normal * 2.0 - 1.0 );

	// Рассчитываем результирующую нормаль
	vec3 resultingNormal = normalize( normal.x * t + normal.y * b + normal.z * n );

	return resultingNormal;
}
```

Почему карта нормалей синяя ?

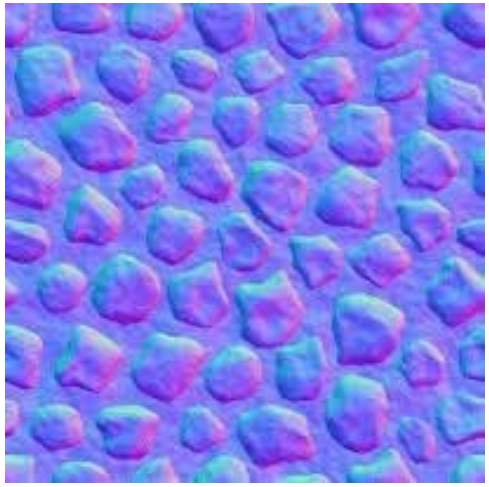
Исходя из формул

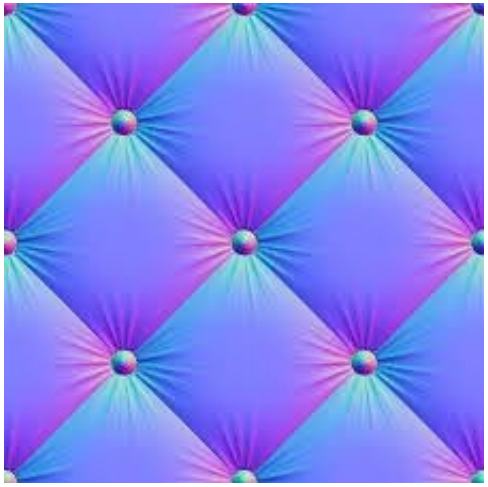
$n_{tex} = (R,G,B) * 2 - 1$

$\mathbf{n_{res} = n_{tex}.x * t + n_{tex}.y * b + n_{tex}.z * n}$

Обычно отклонения от нормали небольшие, то есть проекция вектора вдоль нормали существенно больше, чем проекции вдоль других осей: $n_{tex}.z > n_{tex}.x$ и $n_{tex}.y$,

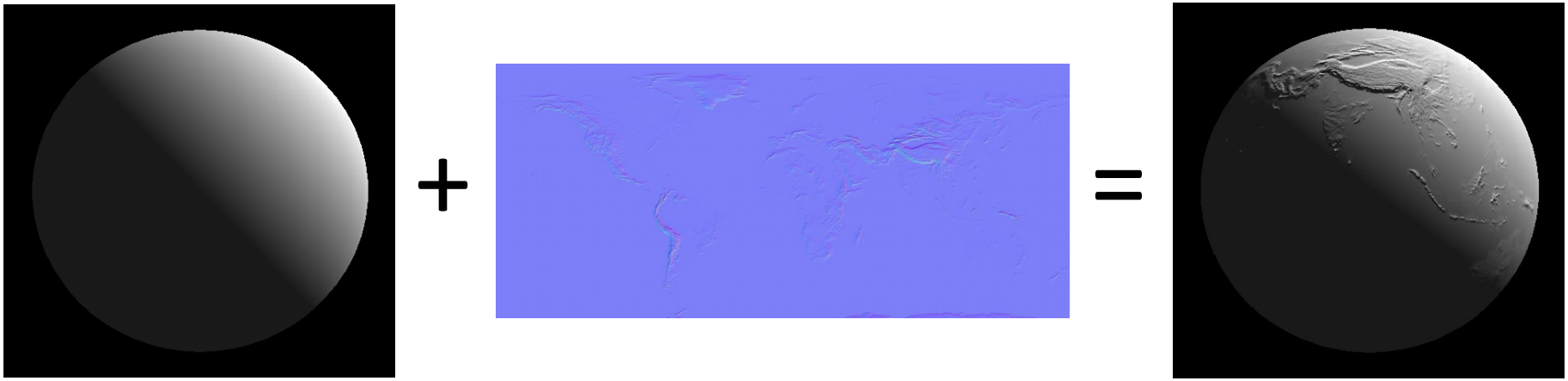
А проекция вдоль нормали и есть синяя компонента



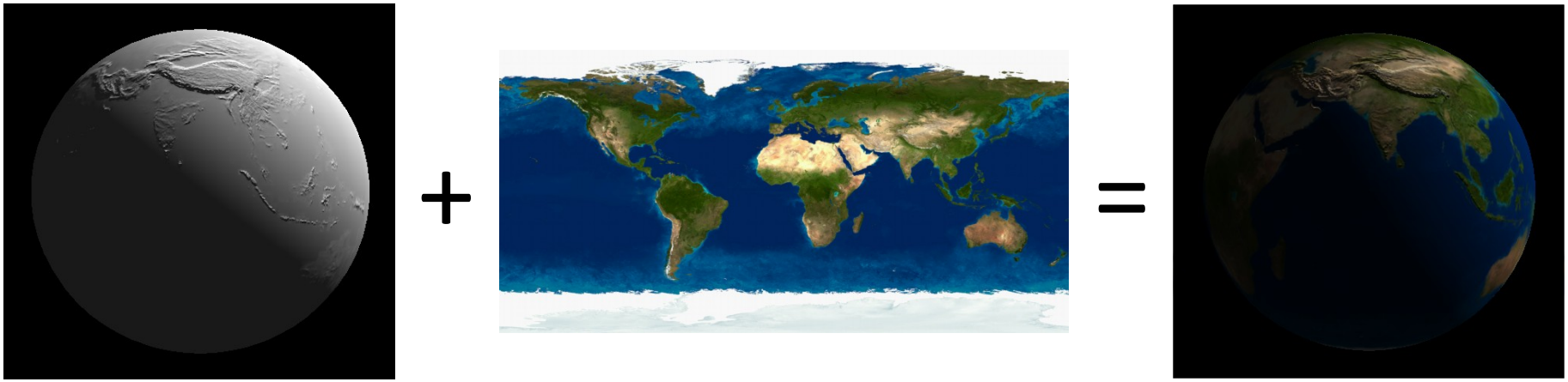


Модуляция нормалей (пример)

1. Сначала наложить карту нормалей на сферу



2. Потом добавить текстуру Земли



Модуляция нормалей (пример)

Компьютерная Графика

Normal mapping

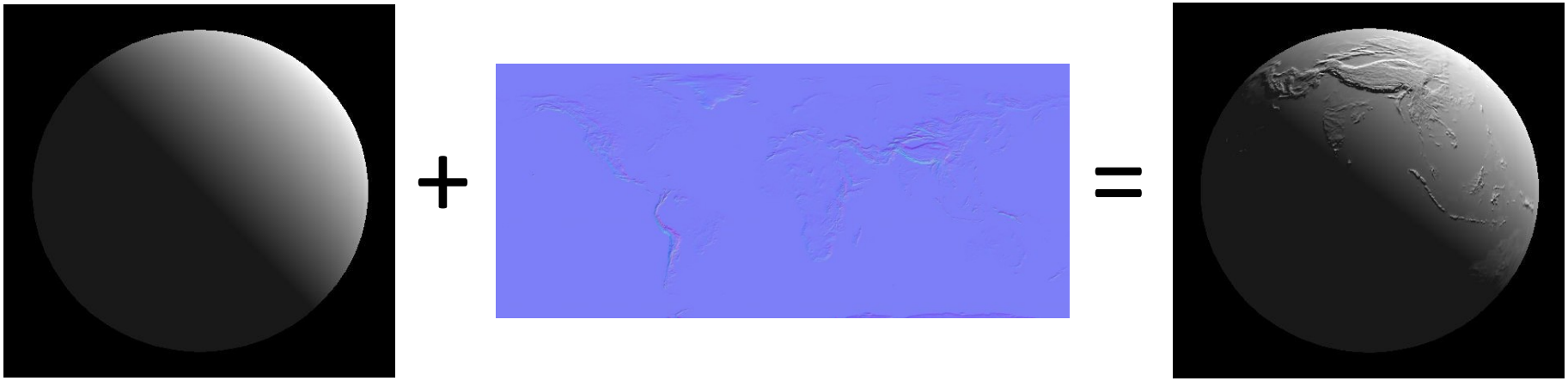
Задача #6

Нарисовать Землю, используя карту нормалей

в верхней части хорошо видно горы (монгольские горы над Индией)

Задача #6. Коротко.

1. Сначала наложить карту нормалей на сферу



2. Потом добавить текстуру Земли

Задача #6. Подробно.

Создать материал MaterialEarth.
Для этого взять код освещения из вершинного шедера в Задаче #3 и перенести его в пиксельный шейдер.

Почему ?

1. Т.к. чтобы достать нормаль из карты нормалей (текстуры), нужно вызвать функцию tex2D() – а она доступна только в пиксельном шейдере.
2. Перенос освещения в пиксельный шейдер позволяет использовать низкополигональные модели (mesh-и). А выглядеть они будут как высокополигональные - за счёт того, что освещения будет рассчитываться попиксельно.

Задача #6. Подробно.

Задача вершинного шейдера - передать данные в пиксельный.
Ниже пердставлен кусок вершинного шейдера.

```
struct VS_INPUT
{
   float4    Position : POSITION;
   float4    Normal   : NORMAL;
   float4    Color    : COLOR;
   float4    uv1      : TEXCOORD0;
};

struct VS_OUTPUT
{
   float4    Position       : POSITION;
   float4    Color          : COLOR0;
   float4    uv1            : TEXCOORD0;
   float4    LocalPos       : TEXCOORD1;
   float4    LocalNorm      : TEXCOORD2;
};
```

Задача #6. Подробно.

Задача вершинного шейдера - передать данные в пиксельный.
Ниже пердставлен кусок вершинного шейдера.

```
struct VS_OUTPUT
{
   float4   Position      : POSITION;
   float4   Color         : COLOR0;
   float4   uv1           : TEXCOORD0;
   float4   LocalPos      : TEXCOORD1;
   float4   LocalNorm     : TEXCOORD2;
};
```

Обратите внимание, что нам нужно положить в текстурные координаты (это просто регистры, которых доступно 8 штук) позицию и нормаль вершины. Нормаль нужна для diffuse и specular освещения в пиксельном шедере. Позиция нужна для specular освещения в пиксельном шедере.

Задача #6. Подробно.

Тогда в пиксельном шейдере можно будет использовать эти данные.
Ниже пердставлен кусок пиксельного шейдера.

```
struct PS_INPUT
{
    float4   Position       : POSITION;
    float4   Color          : COLOR0;
    float4   uv1            : TEXCOORD0;
    float4   LocalPos       : TEXCOORD1;
    float4   LocalNorm      : TEXCOORD2;
};
```

Номера регистров (TEXCOORD) должны совпадать в вершинном и пиксельном шейдерах.

Задача #6. Подробно.

Сначала нужно реализовать осещение в пиксельном шейдере на основе обычной нормали (нормали из вершины).
Причём достаточно взять из Задачи #3 код освещения diffuse и только от одного источника - directional.

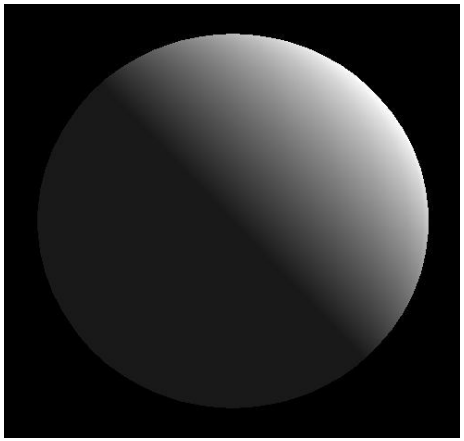
Когда освещение будет работать правильно, можно приступить к нормали из карты нормалей
$\mathbf{n_{res} = n_{tex}.x * t + n_{tex}.y * b + n_{tex}.z * n}$
Рассчитать новую нормаль по этой формуле и использовать в расчёте освещения вместо нормали из вершины.

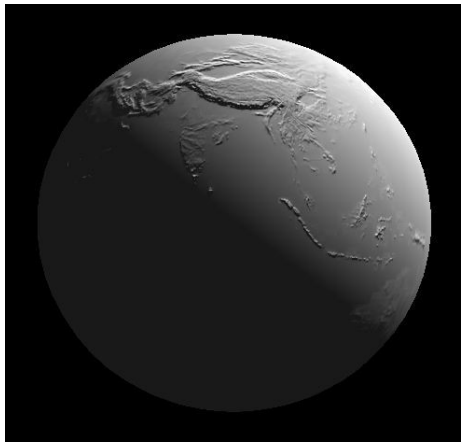
Задача #6. Подробно.

$n_{res} = n_{tex}.x * t + n_{tex}.y * b + n_{tex}.z * n$

Обратите внимание, что для расчета освещения нормали нужно привести в **глобальную систему координат**.

Ещё раз: сначала осветить сферу в пиксельном шейдере (достаточно directional источника света). Потом взять нормаль из карты нормалей и наложить текстуру Земли.





Компьютерная Графика

Mesh Import

Импорт mesh-ей

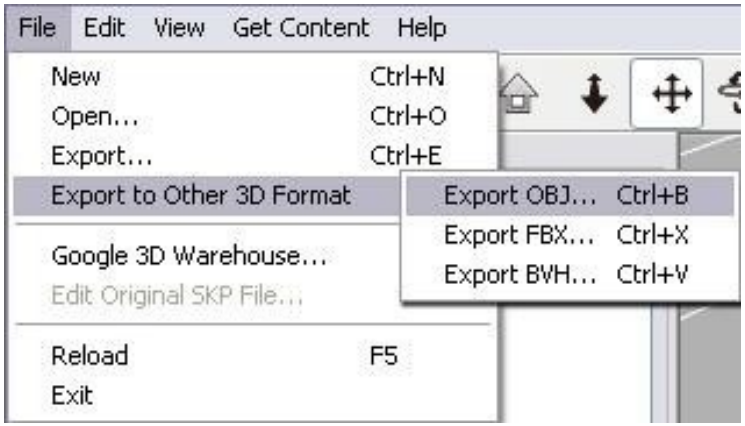
3d-приложения состоят из mesh-ей и кода

- mesh создаётся в 3d-редакторе (3ds Max, Maya, Blender, и т.д.)
- mesh экспортируют из 3d-редактора в бинарный или текстовый файл
- mesh импортируют в своё приложение из этого файла

Форматы файлов

Каждый 3d-редактор (в нём дизайнер создаёт mesh-и) естественно имеет свой формат файла для экспорта

3ds Max	*.max
Maya	*.ma, *.mb
Blender	*.blend



Некоторые форматы поддерживают многие 3d-редакторы

*.fbx

*.obj (простой текстовый формат)

Формат *.obj

- открытый текстовый формат
  https://ru.wikipedia.org/wiki/Obj
  http://www.martinreddy.net/gfx/3d/OBJ.spec
- принят разработчиками многих 3d-редакторов
  e-Frontier's Poser, **Maya**, XSI, **Blender**, MeshLab, Misfit Model 3D, **3D Studio Max** и Rhinoceros 3D, Hexagon, CATIA, Newtek Lightwave, Art of Illusion, milkshape 3d,Modo, Cinema 4D, Zanoza Modeller и т. д.
- хранит только геометрию

вершины
- позиция
- нормаль
- текстурные координаты

полигоны
- треугольники
- квадраты


```
v    -5.000000   5.000000   0.000000
v    -5.000000  -5.000000   0.000000
v     5.000000  -5.000000   0.000000
v     5.000000   5.000000   0.000000
vt   -5.000000   5.000000   0.000000
vt   -5.000000  -5.000000   0.000000
vt    5.000000  -5.000000   0.000000
vt    5.000000   5.000000   0.000000
vn    0.000000   0.000000   1.000000
vn    0.000000   0.000000   1.000000
vn    0.000000   0.000000   1.000000
vn    0.000000   0.000000   1.000000
```

Формат *.obj. Описание.

```
# Это комментарий
...
# Список вершин, с координатами (x,y,z[,w]), w является не обязательным и по умолчанию
1.0.
v 0.123 0.234 0.345 1.0
v ...
...
# Список текстурных координат (u,v[,w]), w является не обязательным и по умолчанию 0.
vt 0.500 -1.352 [0.234]
vt ...
...
# Список нормалей (x,y,z); нормали могут быть не нормированными.
vn 0.707 0.000 0.707
vn ...
...
# Треугольники и квадраты хранятся так
# f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 v4/vt4/vn4
# v1    - индекс из списка вершин
# vt1 - индекс из списка текс. координат
# vn1 - индекс из списка нормалей
f 1/1/1 2/2/2 3/3/3
f ...
...
```

Формат *.obj. Описание.

```
# Треугольники
# могут храниться так

# индексы позиции/текс. коорд./нормали ...
# f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3
f 1/1/1 2/2/2 3/3/3

# либо

# индексы позиции/текс. коорд. ...
# f v1/vt1 v2/vt2 v3/vt3
f 1/1 2/2 3/3

# либо

# индексы позиции/нормали ...
# f v1//vt1 v2//vt2 v3//vt3
f 1//1 2//2 3//3
```

```
# Квадраты
# могут храниться так

# индексы позиции/текс. коорд./нормали ...
# f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 v4/vt4/vn4
f 1/1/1 2/2/2 3/3/3 4/4/4

# либо

# индексы позиции/текс. коорд. ...
# f v1/vt1 v2/vt2 v3/vt3 v4/vt4
f 1/1 2/2 3/3 4/4

# либо

# индексы позиции/нормали ...
# f v1//vt1 v2//vt2 v3//vt3 v4//vt4
f 1//1 2//2 3//3 4//4
```

Задача #7

Сделать класс MeshObjFile, которому в конструктор передаётся путь до obj-файла - он его загружает.

Загрузить 3 объекта: куб, сферу и дом
(файл GraphicsEngine_2016-11-20_Obj_only.zip в DropBox)