

# Компьютерная графика

## Лекция 5: Текстуры

---

2023

# Что такое текстуры?

- Текстуры – изображения в памяти GPU (чаще – несколько изображений)

# Что такое текстуры?

- Текстуры – изображения в памяти GPU (чаще – несколько изображений)
- Их можно читать из шейдеров

# Что такое текстуры?

- Текстуры – изображения в памяти GPU (чаще – несколько изображений)
- Их можно читать из шейдеров
- В них можно загружать пиксели из CPU

# Что такое текстуры?

- Текстуры – изображения в памяти GPU (чаще – несколько изображений)
- Их можно читать из шейдеров
- В них можно загружать пиксели из CPU
- В них можно рисовать (вместо рисования в окно – для этого нужны фреймбуферы)

# Что такое текстуры?

- Текстуры – изображения в памяти GPU (чаще – несколько изображений)
- Их можно читать из шейдеров
- В них можно загружать пиксели из CPU
- В них можно рисовать (вместо рисования в окно – для этого нужны фреймбуферы)
- Поддерживают много специфичных для изображений операций

# Что нужно знать про текстуры?

- Типы текстур в OpenGL

# Что нужно знать про текстуры?

- Типы текстур в OpenGL
- Как читать из текстур в шейдере

# Что нужно знать про текстуры?

- Типы текстур в OpenGL
- Как читать из текстур в шейдере
- Какие есть настройки чтения из текстур

# Что нужно знать про текстуры?

- Типы текстур в OpenGL
- Как читать из текстур в шейдере
- Какие есть настройки чтения из текстур
- Как загрузить данные в текстуру

# Что нужно знать про текстуры?

- Типы текстур в OpenGL
- Как читать из текстур в шейдере
- Какие есть настройки чтения из текстур
- Как загрузить данные в текстуру
- Как привязать текстуру к шейдеру

# Что нужно знать про текстуры?

- Типы текстур в OpenGL
- Как читать из текстур в шейдере
- Какие есть настройки чтения из текстур
- Как загрузить данные в текстуру
- Как привязать текстуру к шейдеру
- Как использовать текстуру для 3D моделей

# Что такое текстуры в OpenGL?

- Текстуры – объекты OpenGL

# Что такое текстуры в OpenGL?

- Текстуры – объекты OpenGL
- `glGenTextures`, `glDeleteTextures`, `glBindTexture`

# Что такое текстуры в OpenGL?

- Текстуры – объекты OpenGL
- `glGenTextures`, `glDeleteTextures`, `glBindTexture`
- `target` – тип текстуры (об этом чуть позже)

# Что такое текстуры в OpenGL?

- Текстуры – объекты OpenGL
- `glGenTextures`, `glDeleteTextures`, `glBindTexture`
- `target` – тип текстуры (об этом чуть позже)
- Если текстура сделана текущей для какого-то `target`, её никогда нельзя сделать текущей для другого `target`
  - $\Rightarrow$  Тип текстуры определяется первым вызовом `glBindTexture` для неё

# Что такое текстуры в OpenGL?

- Текстуры – объекты OpenGL
- `glGenTextures`, `glDeleteTextures`, `glBindTexture`
- `target` – тип текстуры (об этом чуть позже)
- Если текстура сделана текущей для какого-то `target`, её никогда нельзя сделать текущей для другого `target`
  - $\Rightarrow$  Тип текстуры определяется первым вызовом `glBindTexture` для неё
- ID текущей для конкретного `target` текстуры хранится не в контексте OpenGL, а в т.н. *texture unit'ax* (о них позже)

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)

## Одномерная текстура (GL\_TEXTURE\_1D)

- Градиенты цветов
- Предподсчитанные значения функций (напр. для освещения)



## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура

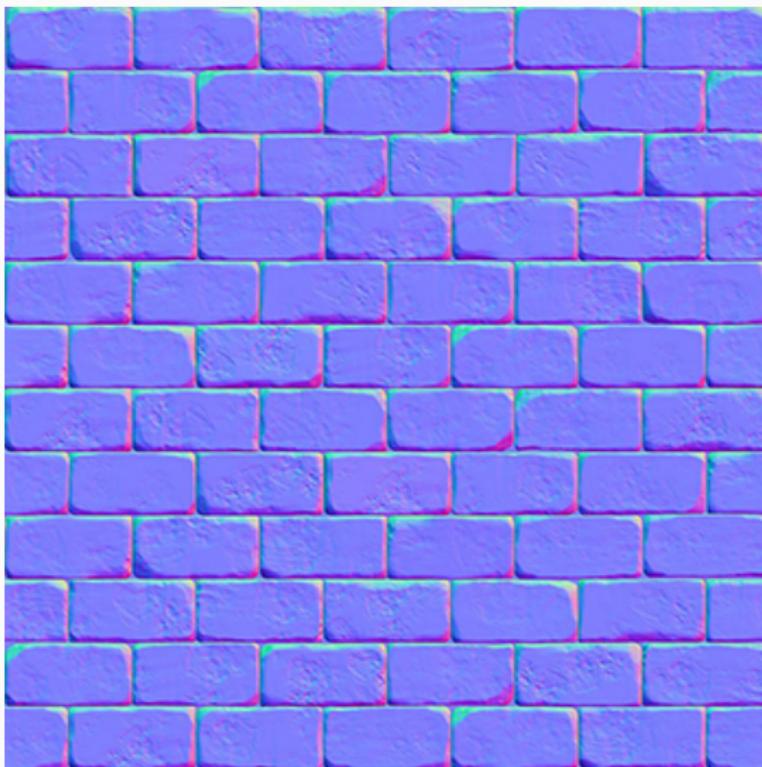
## Двумерная текстура (GL\_TEXTURE\_2D)



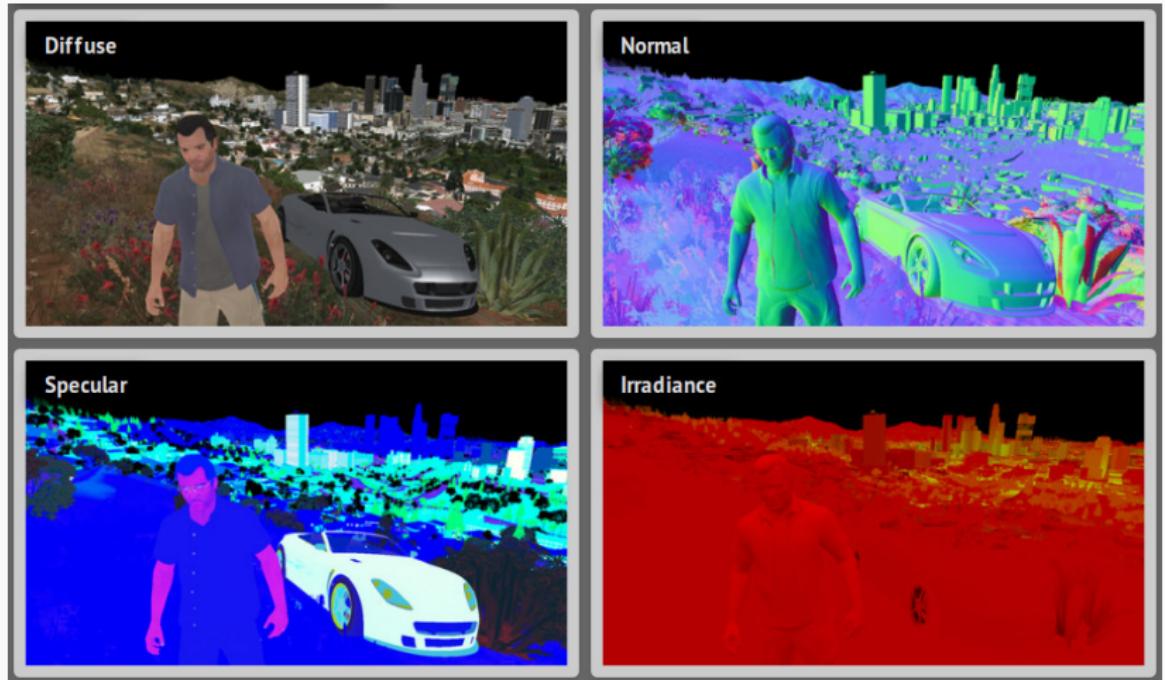
## Двумерная текстура (GL\_TEXTURE\_2D)

- Самый часто используемый тип
- Любые числовые свойства поверхностей: цвет (альбедо), нормаль, тип материала
- Промежуточные изображения при рендеринге: G-buffer, ID buffer, HDR buffer
- Карта высот ландшафта
- Элементы UI
- Предподсчитанные значения 2D функции
- etc.

## Карта нормалей



# G-buffer



# Карта высот



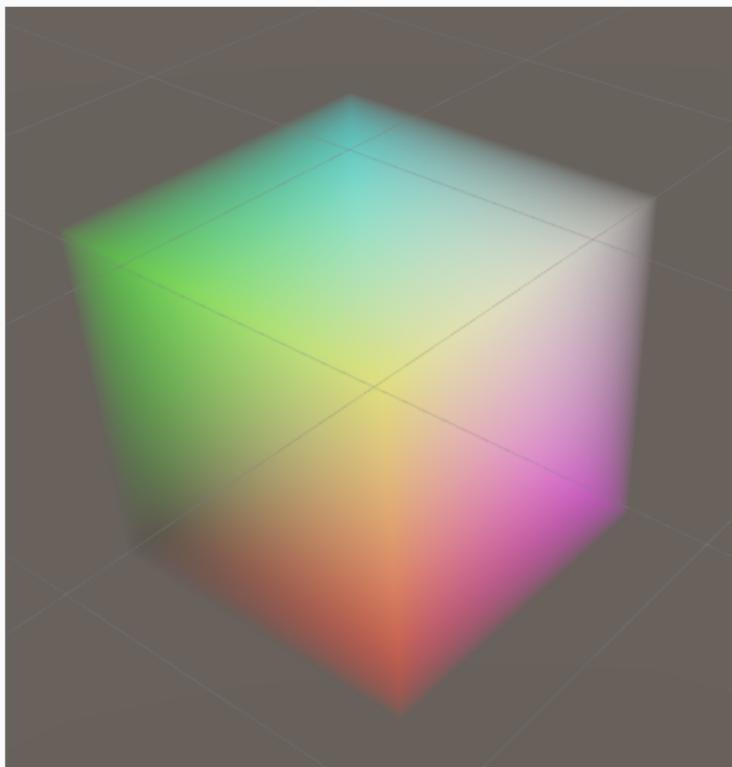
## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура

## Трёхмерная текстура (GL\_TEXTURE\_3D)



## Трёхмерная текстура (GL\_TEXTURE\_3D)

- Любые числовые свойства пространства: плотность, цвет, коэффициент рассеяния
- Объёмные (volumetric) данные: дым, туман, и т.п.
- Воксельные данные
- Предподсчитанные значения 3D функции

## Типы текстур

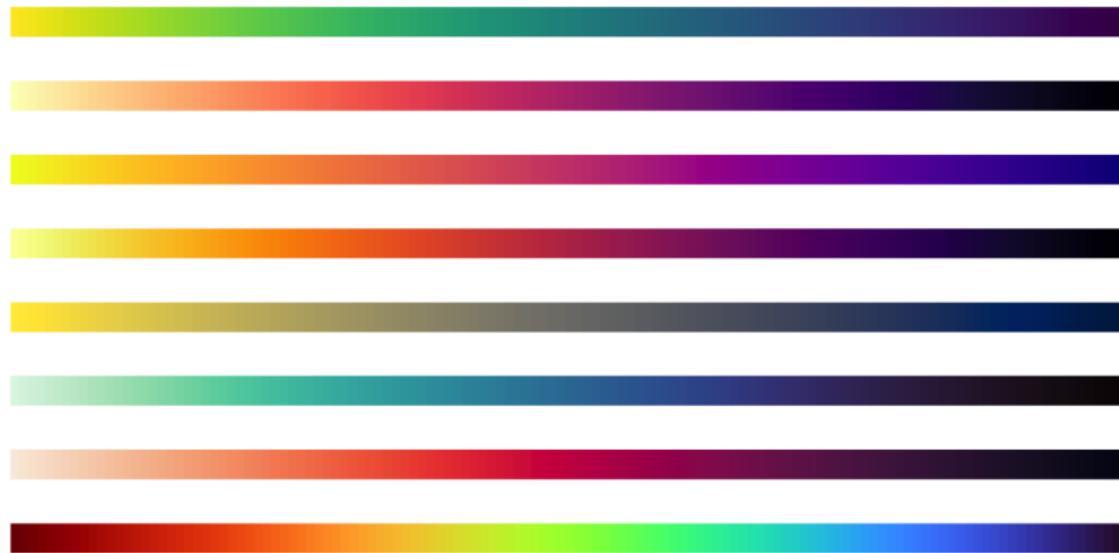
- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера

## Массив одномерных текстур (GL\_TEXTURE\_1D\_ARRAY)

- Набор градиентов цветов



## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера

## Массив двумерных текстур (GL\_TEXTURE\_2D\_ARRAY)

- Массив текстур для однотипных объектов, например, вoxelей



## Типы текстур

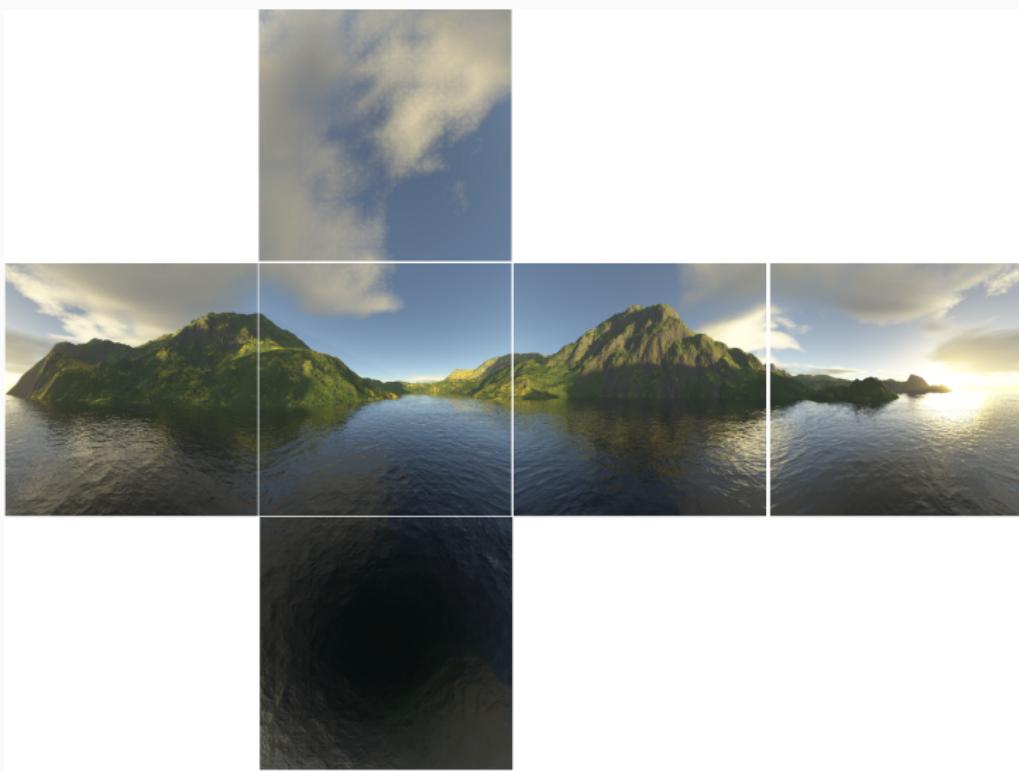
- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера
- GL\_TEXTURE\_CUBE\_MAP – текстуры, хранящиеся как 6 граней куба

## Сиветар-текстура (GL\_TEXTURE\_CUBEMAP)

- Небо (skybox), отражения (environment map, reflection map), предпосчитанные значения функции на сфере



## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера
- GL\_TEXTURE\_CUBE\_MAP – текстуры, хранящиеся как 6 граней куба

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера
- GL\_TEXTURE\_CUBE\_MAP – текстуры, хранящиеся как 6 граней куба
- GL\_TEXTURE\_CUBE\_MAP\_ARRAY – массив cubemap'ов (OpenGL 4.0)

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера
- GL\_TEXTURE\_CUBE\_MAP – текстуры, хранящиеся как 6 граней куба
- GL\_TEXTURE\_CUBE\_MAP\_ARRAY – массив cubemap'ов (OpenGL 4.0)
- GL\_TEXTURE\_RECTANGLE – прямоугольные текстуры

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера
- GL\_TEXTURE\_CUBE\_MAP – текстуры, хранящиеся как 6 граней куба
- GL\_TEXTURE\_CUBE\_MAP\_ARRAY – массив cubemap'ов (OpenGL 4.0)
- GL\_TEXTURE\_RECTANGLE – прямоугольные текстуры (плохое название, другие текстуры тоже могут быть прямоугольными; в современном OpenGL бесполезны)

## Типы текстур

- GL\_TEXTURE\_1D – одномерная текстура (полоска пикселей)
- GL\_TEXTURE\_2D – двумерная текстура
- GL\_TEXTURE\_3D – трёхмерная текстура
- GL\_TEXTURE\_1D\_ARRAY – массив одномерных текстур одинакового размера
- GL\_TEXTURE\_2D\_ARRAY – массив двумерных текстур одинакового размера
- GL\_TEXTURE\_CUBE\_MAP – текстуры, хранящиеся как 6 граней куба
- GL\_TEXTURE\_CUBE\_MAP\_ARRAY – массив cubemap'ов (OpenGL 4.0)
- GL\_TEXTURE\_RECTANGLE – прямоугольные текстуры (плохое название, другие текстуры тоже могут быть прямоугольными; в современном OpenGL бесполезны)
- GL\_TEXTURE\_BUFFER – текстуры, берущие данные из buffer object'a

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`
- `sampler2D/isampler2D/usampler2D`

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`
- `sampler2D/isampler2D/usampler2D`
- `sampler3D/isampler3D/usampler3D`

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`
- `sampler2D/isampler2D/usampler2D`
- `sampler3D/isampler3D/usampler3D`
- `sampler1DArray/isampler1DArray/usampler1DArray`

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`
- `sampler2D/isampler2D/usampler2D`
- `sampler3D/isampler3D/usampler3D`
- `sampler1DArray/isampler1DArray/usampler1DArray`
- `sampler2DArray/isampler2DArray/usampler2DArray`

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`
- `sampler2D/isampler2D/usampler2D`
- `sampler3D/isampler3D/usampler3D`
- `sampler1DArray/isampler1DArray/usampler1DArray`
- `sampler2DArray/isampler2DArray/usampler2DArray`
- `samplerCube/isamplerCube/usamplerCube`

## Текстуры в шейдере - samplers

- В шейдере текстура представлена uniform-переменной с особым типом, соответствующим типу текстуры и формату пикселей (floating-point / integer / unsigned integer)
- `sampler1D/isampler1D/usampler1D`
- `sampler2D/isampler2D/usampler2D`
- `sampler3D/isampler3D/usampler3D`
- `sampler1DArray/isampler1DArray/usampler1DArray`
- `sampler2DArray/isampler2DArray/usampler2DArray`
- `samplerCube/isamplerCube/usamplerCube`
- `samplerBuffer/isamplerBuffer/usamplerBuffer`

## Чтение из текстур

- `texture(sampler, coords)` – возвращает `vec4/ivec4/uvec4` в зависимости от типа текстуры (даже если в текстуре на самом деле меньше компонент)

## Чтение из текстур

- `texture(sampler, coords)` – возвращает `vec4/ivec4/uvec4` в зависимости от типа текстуры (даже если в текстуре на самом деле меньше компонент)
- `coords` – текстурные координаты, их тип и смысл зависят от типа текстуры

## Чтение из текстур

- `texture(sampler, coords)` – возвращает `vec4/ivec4/uvec4` в зависимости от типа текстуры (даже если в текстуре на самом деле меньше компонент)
- `coords` – текстурные координаты, их тип и смысл зависят от типа текстуры
  - Для 1D/2D/3D текстур `float/vec2/vec3`: нормированные координаты, [0..1] по каждой оси

# Чтение из текстур

- `texture(sampler, coords)` – возвращает `vec4/ivec4/uvec4` в зависимости от типа текстуры (даже если в текстуре на самом деле меньше компонент)
- `coords` – текстурные координаты, их тип и смысл зависят от типа текстуры
  - Для 1D/2D/3D текстур `float/vec2/vec3`: нормированные координаты, [0..1] по каждой оси
  - Для 1D/2D array текстур `vec2/vec3`: первые одна/две координаты – нормированные, последняя – номер слоя (не нормированный)

# Чтение из текстур

- `texture(sampler, coords)` – возвращает `vec4/ivec4/uvec4` в зависимости от типа текстуры (даже если в текстуре на самом деле меньше компонент)
- `coords` – текстурные координаты, их тип и смысл зависят от типа текстуры
  - Для 1D/2D/3D текстур `float/vec2/vec3`: нормированные координаты, [0..1] по каждой оси
  - Для 1D/2D array текстур `vec2/vec3`: первые одна/две координаты – нормированные, последняя – номер слоя (не нормированный)
  - Для cubemap текстур `vec3`: вектор направления из центра куба, возвращается значение из пересечения луча с поверхностью куба

# Чтение из текстур

- `texture(sampler, coords)` – возвращает `vec4/ivec4/uvec4` в зависимости от типа текстуры (даже если в текстуре на самом деле меньше компонент)
- `coords` – текстурные координаты, их тип и смысл зависят от типа текстуры
  - Для 1D/2D/3D текстур `float/vec2/vec3`: нормированные координаты, [0..1] по каждой оси
  - Для 1D/2D array текстур `vec2/vec3`: первые одна/две координаты – нормированные, последняя – номер слоя (не нормированный)
  - Для cubemap текстур `vec3`: вектор направления из центра куба, возвращается значение из пересечения луча с поверхностью куба
  - Частое название в случае 2D текстур: *UV-координаты*

## Чтение из текстур: примеры

- `sampler2D`: `texture(sampler, vec2(0.6, 0.5))` – пиксель чуть правее центра текстуры

## Чтение из текстур: примеры

- `sampler2D: texture(sampler, vec2(0.6, 0.5))` – пиксель чуть правее центра текстуры
- `sampler2DArray: texture(sampler, vec3(0.1, 0.9, 12.0))` – пиксель в левом нижнем углу на 12ом слое

## Чтение из текстур: примеры

- `sampler2D`: `texture(sampler, vec2(0.6, 0.5))` – пиксель чуть правее центра текстуры
- `sampler2DArray`: `texture(sampler, vec3(0.1, 0.9, 12.0))` – пиксель в левом нижнем углу на 12ом слое
- `samplerCube`: `texture(sampler, vec3(0.1, 1.0, -0.1))` – пиксель на +Y грани куба, около центра грани

## Чтение из текстур

- Текстура хранит набор дискретных пикселей, но обращение к ней происходит по floating-point координатам

## Чтение из текстур

- Текстура хранит набор дискретных пикселей, но обращение к ней происходит по floating-point координатам
- ⇒ режимы фильтрации текстур
  - GL\_NEAREST – использовать ближайший к указанным координатам пиксель
  - GL\_LINEAR – использовать линейную/билинейную/трилинейную интерполяцию между соседними 2/4/8 пикселями (для 1D/2D/3D соответственно)
  - **NB:** Для array-текстур номер слоя всегда ближайший!

## Чтение из текстур

- Текстура хранит набор дискретных пикселей, но обращение к ней происходит по floating-point координатам
- ⇒ режимы фильтрации текстур
  - GL\_NEAREST – использовать ближайший к указанным координатам пиксель
  - GL\_LINEAR – использовать линейную/билинейную/трилинейную интерполяцию между соседними 2/4/8 пикселями (для 1D/2D/3D соответственно)
  - **NB:** Для array-текстур номер слоя всегда ближайший!
- Настраиваются отдельно для случая, когда
  - Пиксель текстуры больше пикселя на экране (magnification) – GL\_TEXTURE\_MAG\_FILTER
  - Пиксель текстуры меньше пикселя на экране (minification) – GL\_TEXTURE\_MIN\_FILTER

# Чтение из текстур

- Текстура хранит набор дискретных пикселей, но обращение к ней происходит по floating-point координатам
- ⇒ режимы фильтрации текстур
  - GL\_NEAREST – использовать ближайший к указанным координатам пиксель
  - GL\_LINEAR – использовать линейную/билинейную/трилинейную интерполяцию между соседними 2/4/8 пикселями (для 1D/2D/3D соответственно)
  - **NB:** Для array-текстур номер слоя всегда ближайший!
- Настраиваются отдельно для случая, когда
  - Пиксель текстуры больше пикселя на экране (magnification) – GL\_TEXTURE\_MAG\_FILTER
  - Пиксель текстуры меньше пикселя на экране (minification) – GL\_TEXTURE\_MIN\_FILTER

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
```

## Nearest vs Linear

**NEAREST**

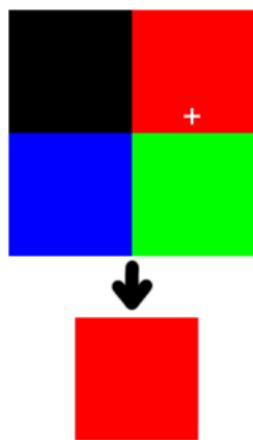


**LINEAR**

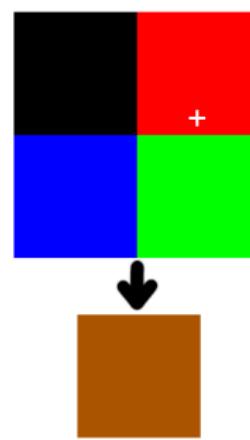


# Nearest vs Linear

**NEAREST**



**LINEAR**



# Minification vs Magnification

**MINIFICATION**



**MAGNIFICATION**



# Mipmaps

- Когда пиксель текстуры меньше пикселя на экране (minification), текстура выглядит плохо:
  - `GL_NEAREST` – часть пикселей текстуры не попадают на экран

# Mipmaps

- Когда пиксель текстуры меньше пикселя на экране (minification), текстура выглядит плохо:
  - GL\_NEAREST – часть пикселей текстуры не попадают на экран
  - GL\_LINEAR – нормально до  $x2$  уменьшения, потом пиксели тоже не попадают на экран

# Mipmaps

- Когда пиксель текстуры меньше пикселя на экране (minification), текстура выглядит плохо:
  - GL\_NEAREST – часть пикселей текстуры не попадают на экран
  - GL\_LINEAR – нормально до  $x2$  уменьшения, потом пиксели тоже не попадают на экран
- Решение: *mipmap-уровни*

# Mipmaps

- Mipmaps – уменьшенные копии текстуры для использования при *минификации*

# Mipmaps

- Mipmaps – уменьшенные копии текстуры для использования при минификации
- Для 2D текстуры  $W \times H$  первый уровень имеет размер  $\lceil \frac{W}{2} \rceil \times \lceil \frac{H}{2} \rceil$ , второй уровень имеет размер  $\lceil \frac{W}{4} \rceil \times \lceil \frac{H}{4} \rceil$ , и т.д.

# Mipmaps

- Mipmaps – уменьшенные копии текстуры для использования при минификации
- Для 2D текстуры  $W \times H$  первый уровень имеет размер  $\lceil \frac{W}{2} \rceil \times \lceil \frac{H}{2} \rceil$ , второй уровень имеет размер  $\lceil \frac{W}{4} \rceil \times \lceil \frac{H}{4} \rceil$ , и т.д.
- Последний mipmap-уровень имеет размер  $1 \times 1$

# Mipmaps

- Mipmaps – уменьшенные копии текстуры для использования при минификации
- Для 2D текстуры  $W \times H$  первый уровень имеет размер  $\lceil \frac{W}{2} \rceil \times \lceil \frac{H}{2} \rceil$ , второй уровень имеет размер  $\lceil \frac{W}{4} \rceil \times \lceil \frac{H}{4} \rceil$ , и т.д.
- Последний міртмар-уровень имеет размер  $1 \times 1$
- Для 1D или 3D текстур – аналогично по всем осям

# Mipmaps

- Mipmaps – уменьшенные копии текстуры для использования при минификации
- Для 2D текстуры  $W \times H$  первый уровень имеет размер  $\lceil \frac{W}{2} \rceil \times \lceil \frac{H}{2} \rceil$ , второй уровень имеет размер  $\lceil \frac{W}{4} \rceil \times \lceil \frac{H}{4} \rceil$ , и т.д.
- Последний міртмар-уровень имеет размер  $1 \times 1$
- Для 1D или 3D текстур – аналогично по всем осям
- Для array-текстур – отдельный набор міртмар'ов для каждого слоя

# Mipmaps

- Mipmaps – уменьшенные копии текстуры для использования при минификации
- Для 2D текстуры  $W \times H$  первый уровень имеет размер  $\lceil \frac{W}{2} \rceil \times \lceil \frac{H}{2} \rceil$ , второй уровень имеет размер  $\lceil \frac{W}{4} \rceil \times \lceil \frac{H}{4} \rceil$ , и т.д.
- Последний міртмар-уровень имеет размер  $1 \times 1$
- Для 1D или 3D текстур – аналогично по всем осям
- Для array-текстур – отдельный набор міртмар'ов для каждого слоя
- Для сиветар-текстур – отдельный набор міртмар'ов для каждой грани

# Mipmaps



# Mipmaps

- Мармар'ы можно загрузить отдельно, так же, как саму текстуру

# Mipmaps

- Мірмар'ї можно загрузить отдельно, так же, как саму текстуру
- Попросить OpenGL сгенерировать мірмар'ї на основе самой текстуры автоматически (линейным усреднением)

# Mipmaps

- Мірмар'ї можно загрузить отдельно, так же, как саму текстуру
- Попросить OpenGL сгенерировать мірмар'ї на основе самой текстуры автоматически (линейным усреднением)
- Мірмар'ї можно не использовать

# Mipmaps

- ОПЦИИ ДЛЯ GL\_TEXTURE\_MIN\_FILTER:

# Mipmaps

- Опции для GL\_TEXTURE\_MIN\_FILTER:
  - **GL\_NEAREST\_MIPMAP\_NEAREST**
    - выбирается **ближайший** мірмар уровень, с него выбирается **ближайший** пиксель

# Mipmaps

- Опции для GL\_TEXTURE\_MIN\_FILTER:
  - GL\_NEAREST\_MIPMAP\_NEAREST
    - выбирается ближайший мірмар уровень, с него выбирается ближайший пиксель
  - GL\_LINEAR\_MIPMAP\_NEAREST
    - выбирается ближайший мірмар уровень, с него делается интерполяция ближайших пикселей

# Mipmaps

- Опции для GL\_TEXTURE\_MIN\_FILTER:
  - GL\_NEAREST\_MIPMAP\_NEAREST
    - выбирается ближайший mipmap уровень, с него выбирается ближайший пиксель
  - GL\_LINEAR\_MIPMAP\_NEAREST
    - выбирается ближайший mipmap уровень, с него делается интерполяция ближайших пикселей
  - GL\_NEAREST\_MIPMAP\_LINEAR
    - выбираются два ближайших mipmap уровня, с них выбираются ближайшие пиксели, между ними происходит линейная интерполяция (включено по умолчанию)

# Mipmaps

- Опции для GL\_TEXTURE\_MIN\_FILTER:
  - GL\_NEAREST\_MIPMAP\_NEAREST
    - выбирается ближайший mipmap уровень, с него выбирается ближайший пиксель
  - GL\_LINEAR\_MIPMAP\_NEAREST
    - выбирается ближайший mipmap уровень, с него делается интерполяция ближайших пикселей
  - GL\_NEAREST\_MIPMAP\_LINEAR
    - выбираются два ближайших mipmap уровня, с них выбираются ближайшие пиксели, между ними происходит линейная интерполяция (включено по умолчанию)
  - GL\_LINEAR\_MIPMAP\_LINEAR
    - выбираются два ближайших mipmap уровня, в них по отдельности делается интерполяция ближайших пикселей, между результатами снова происходит интерполяция

# Mipmaps

- По умолчанию включен GL\_NEAREST\_MIPMAP\_LINEAR

# Mipmaps

- По умолчанию включен GL\_NEAREST\_MIPMAP\_LINEAR
- Если вы забыли сгенерировать/загрузить міртмар-уровни, текстура считается невалидной, и будет рисоваться чёрной

# Mipmaps

- По умолчанию включен `GL_NEAREST_MIPMAP_LINEAR`
- Если вы забыли сгенерировать/загрузить мірмар-уровни, текстура считается невалидной, и будет рисоваться чёрной
- Не забывайте выставлять правильные min/mag filter и/или генерировать мірмар'ы!

## Би/трилинейная фильтрация

- Размерность текстуры (1D/2D/3D) и тип фильтрации влияют на общее число линейных интерполяций, отсюда названия *билинейная фильтрация* и *трилинейная фильтрация*

## Би/трилинейная фильтрация

- Размерность текстуры (1D/2D/3D) и тип фильтрации влияют на общее число линейных интерполяций, отсюда названия *билинейная фильтрация* и *трилинейная фильтрация*
- Часто под трилинейной фильтрацией имеют в виду конкретно 2D текстуру с `GL_LINEAR_MIPMAP_LINEAR`

## Анизотропная фильтрация

- Текстура после проецирования на экран может быть нормального размера по одной оси, и маленькая по другой

## Анизотропная фильтрация

- Текстура после проецирования на экран может быть нормального размера по одной оси, и маленькая по другой
- Большой тіртмар уровень хорош для одной оси, маленький – для другой; ни один уровень не будет выглядеть хорошо

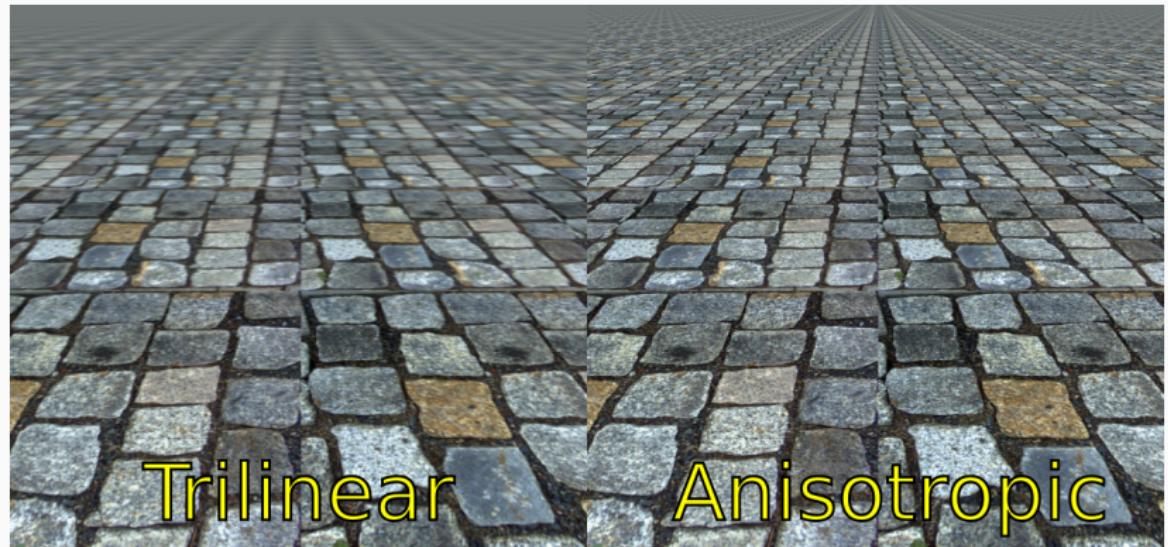
## Анизотропная фильтрация

- Текстура после проецирования на экран может быть нормального размера по одной оси, и маленькая по другой
- Большой тіртар уровень хорош для одной оси, маленький – для другой; ни один уровень не будет выглядеть хорошо
- Решение: *анизотропная фильтрация* – читать текстуру в нескольких местах сразу и усреднять (поддерживается аппаратно)

# Анизотропная фильтрация

- Текстура после проецирования на экран может быть нормального размера по одной оси, и маленькая по другой
- Большой тіртар уровень хорош для одной оси, маленький – для другой; ни один уровень не будет выглядеть хорошо
- Решение: *анизотропная фильтрация* – читать текстуру в нескольких местах сразу и усреднять (поддерживается аппаратно)
- Есть в OpenGL 4.6, но почти везде доступно через расширение `EXT_texture_filter_anisotropic`
  - (подробнее про расширения OpenGL поговорим позже)

# Анизотропная фильтрация



## Wrapping mode

- При чтении из текстуры координаты могут выходить за пределы диапазона

## Wrapping mode

- При чтении из текстуры координаты могут выходить за пределы диапазона
- При `GL_LINEAR` для точек у границы текстуры может не оказаться достаточно количество соседних пикселей

## Wrapping mode

- При чтении из текстуры координаты могут выходить за пределы диапазона
- При GL\_LINEAR для точек у границы текстуры может не оказаться достаточно количество соседних пикселей
- ⇒ Wrapping mode:
  - GL\_CLAMP\_TO\_EDGE – брать пиксель с границы
  - GL\_CLAMP\_TO\_BORDER – брать значение фиксированного цвета (настраивается для каждой текстуры)
  - GL\_REPEAT – повторять текстуру (включен по умолчанию)
  - GL\_MIRRORED\_REPEAT – повторять текстуру, отражая её на каждый повтор

## Wrapping mode

- При чтении из текстуры координаты могут выходить за пределы диапазона
- При GL\_LINEAR для точек у границы текстуры может не оказаться достаточно количество соседних пикселей
- $\Rightarrow$  Wrapping mode:
  - GL\_CLAMP\_TO\_EDGE – брать пиксель с границы
  - GL\_CLAMP\_TO\_BORDER – брать значение фиксированного цвета (настраивается для каждой текстуры)
  - GL\_REPEAT – повторять текстуру (включен по умолчанию)
  - GL\_MIRRORED\_REPEAT – повторять текстуру, отражая её на каждый повтор
- Настраивается отдельно для каждой текстурной координаты: S, T, R

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)
```

# Wrapping



GL\_REPEAT



GL\_MIRRORED\_REPEAT



GL\_CLAMP\_TO\_EDGE



GL\_CLAMP\_TO\_BORDER

## Выбор mipmap уровня: shader derivatives

- Как GPU понимает, какой выбрать mipmap-уровень, и происходит ли minification/magnification?

## Выбор тіртар уровня: shader derivatives

- Как GPU понимает, какой выбрать тіртар-уровень, и происходит ли minification/magnification? По значениям в соседних пикселях

## Выбор тіртар уровня: shader derivatives

- Как GPU понимает, какой выбрать тіртар-уровень, и происходит ли minification/magnification? По значениям в соседних пикселях
- Фрагментный шейдер всегда запускается на группах 2x2 пикселя (даже если часть из них не были растеризованы)

## Выбор тіртар уровня: shader derivatives

- Как GPU понимает, какой выбрать тіртар-уровень, и происходит ли minification/magnification? По значениям в соседних пикселях
- Фрагментный шейдер всегда запускается на группах 2x2 пикселя (даже если часть из них не были растеризованы)
- В GLSL есть функции  $dFdx/dFdy$  – вычисляют разницу некой величины для пары соседних пикселей в группе 2x2

## Выбор тіртар уровня: shader derivatives

- Как GPU понимает, какой выбрать тіртар-уровень, и происходит ли minification/magnification? По значениям в соседних пикселях
- Фрагментный шейдер всегда запускается на группах 2x2 пикселя (даже если часть из них не были растеризованы)
- В GLSL есть функции  $dFdx/dFdy$  – вычисляют разницу некой величины для пары соседних пикселей в группе 2x2
  - Например,  $dFdx(value)$  – разница между значением  $value$  в пикселе справа и значением в пикселе слева

## Выбор тіртар уровня: shader derivatives

- Как GPU понимает, какой выбрать тіртар-уровень, и происходит ли minification/magnification? По значениям в соседних пикселях
- Фрагментный шейдер всегда запускается на группах 2x2 пикселя (даже если часть из них не были растеризованы)
- В GLSL есть функции  $dFdx/dFdy$  – вычисляют разницу некой величины для пары соседних пикселей в группе 2x2
  - Например,  $dFdx(value)$  – разница между значением  $value$  в пикселе справа и значением в пикселе слева
- По  $dFdx(coords)$  и  $dFdy(coords)$  вычисляется тіртар-уровень (где  $coords$  – текстурные координаты)

## Выбор тіртар уровня: shader derivatives

- Как GPU понимает, какой выбрать тіртар-уровень, и происходит ли minification/magnification? По значениям в соседних пикселях
- Фрагментный шейдер всегда запускается на группах 2x2 пикселя (даже если часть из них не были растеризованы)
- В GLSL есть функции  $dFdx/dFdy$  – вычисляют разницу некой величины для пары соседних пикселей в группе 2x2
  - Например,  $dFdx(value)$  – разница между значением  $value$  в пикселе справа и значением в пикселе слева
- По  $dFdx(coords)$  и  $dFdy(coords)$  вычисляется тіртар-уровень (где  $coords$  – текстурные координаты)
- Интерпретируя  $dFdx/dFdy$  как аппроксимации производных, можно делать много интересных трюков: восстанавливать нормаль к поверхности по z-буферу, рисовать линии фиксированной толщины даже с перспективной проекцией, и т.п.

## Функции чтения из текстур

- `texture` – учитывает mipmap и настройки фильтрации

## Функции чтения из текстур

- `texture` – учитывает mipmaps и настройки фильтрации
- `textureLod` – обратиться напрямую к указанному mipmap-уровню (`LOD = level of detail`)

## Функции чтения из текстур

- `texture` – учитывает mipmap и настройки фильтрации
- `textureLod` – обратиться напрямую к указанному mipmap-уровню ( $LOD = \textit{level of detail}$ )
- `texelFetch` – обратиться напрямую к указанному пикслю, минуя всю фильтрацию

# Формат текстуры

- Текстура – набор (1D/2D/3D массив или 6 2D массивов) пикселей

# Формат текстуры

- Текстура – набор (1D/2D/3D массив или 6 2D массивов) пикселей
- Пиксель – 1, 2, 3 или 4 компоненты в определённом формате

# Формат текстуры

- Текстура – набор (1D/2D/3D массив или 6 2D массивов) пикселей
- Пиксель – 1, 2, 3 или 4 компоненты в определённом формате
- Форматов очень много, вот часто встречающиеся:
  - GL\_RGB8 – 3 нормированных 8-битных канала
  - GL\_RGBA8 – 4 нормированных 8-битных канала
  - GL\_RGBA32F – 4 32-битных floating-point канала
  - GL\_DEPTH\_COMPONENT24 – специальный формат для z-буфера

## Загрузить данные в текстуру

- 1D: `glTexImage1D`

## Загрузить данные в текстуру

- 1D: `glTexImage1D`
- 2D, 1D array или грань cubemap: `glTexImage2D`

## Загрузить данные в текстуру

- 1D: `glTexImage1D`
- 2D, 1D array или грань cubemap: `glTexImage2D`
- 3D или 2D array: `glTexImage3D`

## Загрузить данные в текстуру

```
glTexImage2D(GLenum target, GLint level, GLint internalFormat,  
             GLsizei width, GLsizei height, GLint border,  
             GLenum format, GLenum type, const GLvoid * data);
```

- `target` – тип текстуры (например, `GL_TEXTURE_2D`)
- `level` – mipmap level, который мы хотим загрузить (0 для основной текстуры)
- `internalFormat` – формат хранения пикселей (например, `GL_RGBA8`)
- `width, height` – размер в пикселях
- `border` – легаси, должно быть 0
- `format` – какие компоненты есть в массиве `data` (например, `GL_RGB`)
- `type` – какого типа компоненты в массиве `data` (например, `GL_UNSIGNED_BYTE`)
- `data` – указатель на массив пикселей (сначала первая строка пикселей, потом вторая, и т.д.)

## Загрузить данные в текстуру

- Если `data = nullptr`, под текстуру выделится память, но данные не будут записаны

## Загрузить данные в текстуру

- Если `data = nullptr`, под текстуру выделится память, но данные не будут записаны
- Есть ограничения на возможные пары `format` и `type` (например, `type = GL_UNSIGNED_SHORT_5_6_5` требует `format = GL_RGB`)

## Загрузить данные в текстуру

- Если `data = nullptr`, под текстуру выделится память, но данные не будут записаны
- Есть ограничения на возможные пары `format` и `type` (например, `type = GL_UNSIGNED_SHORT_5_6_5` требует `format = GL_RGB`)
- `format + type` может не совпадать с `internalFormat`, тогда происходит преобразование в `internalFormat`

## Выравнивание данных текстуры

- По умолчанию ожидается, что начало каждой строки пикселей в массиве `data` выровнено на 4 байта

## Выравнивание данных текстуры

- По умолчанию ожидается, что начало каждой строки пикселей в массиве `data` выровнено на 4 байта
- Это может нарушиться, например, с `format = GL_RGB` и `type = GL_UNSIGNED_BYTE` с нечётной шириной текстуры

## Выравнивание данных текстуры

- По умолчанию ожидается, что начало каждой строки пикселей в массиве `data` выровнено на 4 байта
- Это может нарушиться, например, с `format = GL_RGB` и `type = GL_UNSIGNED_BYTE` с нечётной шириной текстуры
- Настраивается с помощью  
`glPixelStorei(GL_UNPACK_ALIGNMENT, 1)`

## Генерация міртар

- После загрузки нулевого міртар-уровня, можно сгенерировать все остальные с помощью `glGenerateMipmap(target)`

## Генерация міртар

- После загрузки нулевого міртар-уровня, можно сгенерировать все остальные с помощью `glGenerateMipmap(target)`
- Делает линейное усреднение пикселей  $\Rightarrow$  не подходит, если в текстуре хранятся нелинейные данные (цвет в формате sRGB, нормали, параметры материала)

# Как связать текстуру и sampler в шейдере?

- ID текущей текстуры для каждого target хранится в т.н. texture unit'e
- Texture unit'ы – **не объекты OpenGL**, их нельзя создавать/удалять
- Их фиксированное число –  
`glGet(GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS)`, как минимум 48
- Есть текущий texture unit (по умолчанию – 0)
- `glActiveTexture(GL_TEXTURE0 + i)` – сделать текущим texture unit с номером i
- `glBindTexture(...)` делает текстуру текущей для данного target в *текущем texture unit'e*

# Как связать текстуру и sampler в шейдере?

- В качестве значения uniform-переменной типа `sampler2D` (и т.п.) указывается номер *texture unit'a*

# Как связать текстуру и sampler в шейдере?

- В качестве значения uniform-переменной типа `sampler2D` (и т.п.) указывается номер *texture unit'a*
- Если для `sampler2D` установлено значение *i*, то в качестве текстуры будет взята `GL_TEXTURE_2D` из *texture unit'a* номер *i*

# Как связать текстуру и sampler в шейдере?

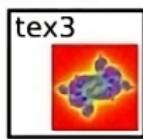
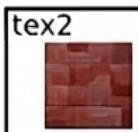
- В качестве значения uniform-переменной типа `sampler2D` (и т.п.) указывается номер *texture unit'a*
- Если для `sampler2D` установлено значение *i*, то в качестве текстуры будет взята `GL_TEXTURE_2D` из texture unit'a номер *i*
- `glUniform1i(color_texture_location, i)`

# Как связать текстуру и sampler в шейдере?

- В качестве значения uniform-переменной типа `sampler2D` (и т.п.) указывается номер *texture unit'a*
- Если для `sampler2D` установлено значение *i*, то в качестве текстуры будет взята `GL_TEXTURE_2D` из texture unit'a номер *i*
- `glUniform1i(color_texture_location, i)`
- Если нужны несколько текстур одновременно в одном шейдере, нужно их сделать текущими для разных *texture unit'ов*, и передать номера этих *texture unit'ов* в uniform-переменные шейдера

# Texture units

## Texture Objects



### Created with:

```
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, ...);
```

## Texture Units

Texture Unit 0  
(gl.TEXTURE0)

Texture Unit 1  
(gl.TEXTURE1)

### Association set up with:

```
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
```

## Sampler Variables in Fragment Shader

```
uniform sampler2D x;
```

```
uniform sampler2D y;
```

### Association set up with:

```
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);
```

# Как использовать текстуру для 3D-модели?

- Обычно каждая вершина 3D-модели хранит пару (нормированных) текстурных координат (UV-развёртка)

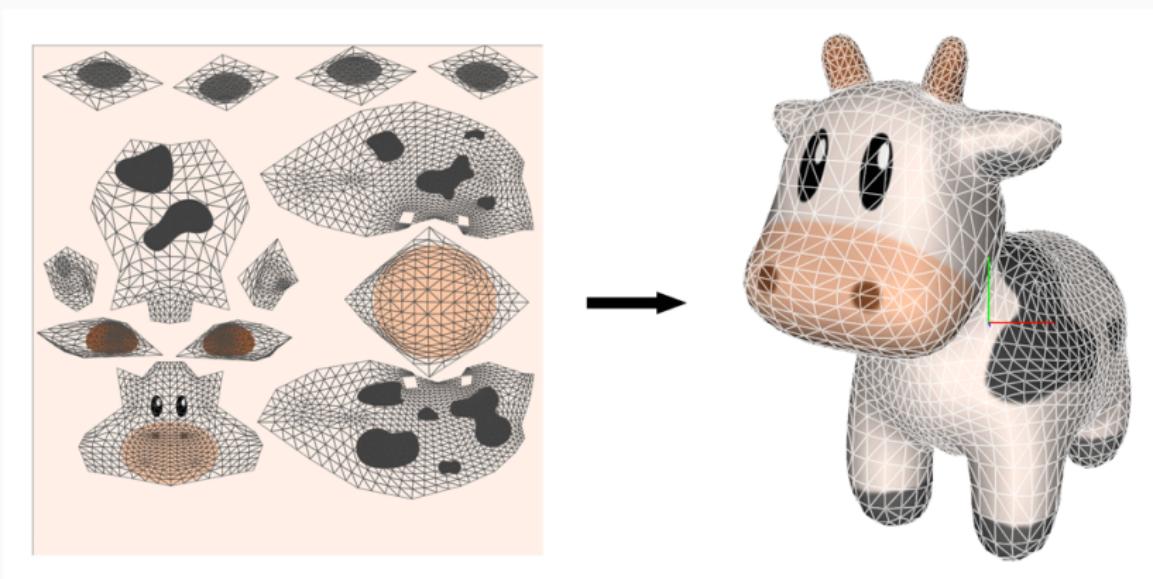
# Как использовать текстуру для 3D-модели?

- Обычно каждая вершина 3D-модели хранит пару (нормированных) текстурных координат (UV-развёртка)
- Они передаются из вершинного шейдера во фрагментный и интерполируются

# Как использовать текстуру для 3D-модели?

- Обычно каждая вершина 3D-модели хранит пару (нормированных) текстурных координат (UV-развёртка)
- Они передаются из вершинного шейдера во фрагментный и интерполируются
- Интерполированные значения текстурных координат используются во фрагментном шейдере для чтения из текстуры

# Как использовать текстуру для 3D-модели?



# Форматы изображений

- `glTexImage2D` ожидает на вход массив готовых пикселей

# Форматы изображений

- `glTexImage2D` ожидает на вход массив готовых пикселей
- Обычно изображения хранятся в более сложных форматах: PNG, JPG, и т.д.

# Форматы изображений

- `glTexImage2D` ожидает на вход массив готовых пикселей
- Обычно изображения хранятся в более сложных форматах: PNG, JPG, и т.д.
- Есть формат Netpbm, хранящий сырье пиксели и простой для чтения/записи

# Форматы изображений

- `glTexImage2D` ожидает на вход массив готовых пикселей
- Обычно изображения хранятся в более сложных форматах: PNG, JPG, и т.д.
- Есть формат Netpbm, хранящий сырые пиксели и простой для чтения/записи
- Есть библиотеки для чтения разных форматов – libpng, libjpeg

# Форматы изображений

- `glTexImage2D` ожидает на вход массив готовых пикселей
- Обычно изображения хранятся в более сложных форматах: PNG, JPG, и т.д.
- Есть формат Netpbm, хранящий сырье пиксели и простой для чтения/записи
- Есть библиотеки для чтения разных форматов – `libpng`, `libjpeg`
- `stb_image.h` – single-header библиотека для чтения разных форматов

# Форматы изображений

- `glTexImage2D` ожидает на вход массив готовых пикселей
- Обычно изображения хранятся в более сложных форматах: PNG, JPG, и т.д.
- Есть формат Netpbm, хранящий сырые пиксели и простой для чтения/записи
- Есть библиотеки для чтения разных форматов – libpng, libjpeg
- `stb_image.h` – single-header библиотека для чтения разных форматов
- Boost GIL (Generic Image Library)

## Псевдокод типичного использования текстуры

```
// инициализация
image = loadImage('image.png')
texture = createTexture()
texture.setMinFilter(GL_LINEAR_MIPMAP_NEAREST)
texture.setMagFilter(GL_NEAREST)
texture.load(image)
texture.generateMipmap()

// рендеринг
program.use()
vao.bind()
program.uniform("color_texture") = 0;
glActiveTexture(GL_TEXTURE0);
texture.bind()
glDrawArrays(...)
```

## Ссылки

- [khronos.org/opengl/wiki/Texture](http://khronos.org/opengl/wiki/Texture)
- [khronos.org/opengl/wiki/Sampler\\_Object](http://khronos.org/opengl/wiki/Sampler_Object)
- [khronos.org/opengl/wiki/Image\\_Format](http://khronos.org/opengl/wiki/Image_Format)
- [learnopengl.com/Getting-started/Textures](http://learnopengl.com/Getting-started/Textures)
- [opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube](http://opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube)
- [open.gl/textures](http://open.gl/textures)