

# Компьютерная графика

Лекция 6: Ошибки OpenGL, расширения OpenGL,  
blending, stencil buffer, framebuffer, renderbuffer,  
пост-обработка

---

2025

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур
  - `glTexParameter`i устанавливающая `mag filter` в что-то отличное от `GL_LINEAR` или `GL_NEAREST`

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур
  - `glTexParameter`i устанавливающая `mag filter` в что-то отличное от `GL_LINEAR` или `GL_NEAREST`
  - `glBindBuffer` с ID буфера, который не был получен через `glGenBuffers`

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур
  - `glTexParameterI` устанавливающая `mag filter` в что-то отличное от `GL_LINEAR` или `GL_NEAREST`
  - `glBindBuffer` с ID буфера, который не был получен через `glGenBuffers`
  - etc.

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур
  - `glTexParameter`i устанавливающая `mag filter` в что-то отличное от `GL_LINEAR` или `GL_NEAREST`
  - `glBindBuffer` с ID буфера, который не был получен через `glGenBuffers`
  - etc.
- В таких случаях генерируется ошибка как особое значение типа `GLenum`

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур
  - `glTexParameter`i устанавливающая `mag filter` в что-то отличное от `GL_LINEAR` или `GL_NEAREST`
  - `glBindBuffer` с ID буфера, который не был получен через `glGenBuffers`
  - etc.
- В таких случаях генерируется ошибка как особое значение типа `GLenum`
- Как правило, вызвавшая ошибку операция не выполняется

# Ошибки OpenGL

- Часто драйвер может легко понять, что в определённом OpenGL-вызове содержится ошибка
  - `glTexImage2D` с параметром `target`, не являющимся одним из типов текстур
  - `glTexParameter`i устанавливающая `mag filter` в что-то отличное от `GL_LINEAR` или `GL_NEAREST`
  - `glBindBuffer` с ID буфера, который не был получен через `glGenBuffers`
  - etc.
- В таких случаях генерируется ошибка как особое значение типа `GLenum`
- Как правило, вызвавшая ошибку операция не выполняется
- `glGetError()` – получить значение ошибки, если она была

# Ошибки OpenGL

Возможные ошибки:

- GL\_NO\_ERROR – ошибки нет

# Ошибки OpenGL

Возможные ошибки:

- GL\_NO\_ERROR – ошибки нет
- GL\_INVALID\_ENUM – недопустимое значения перечисления  
(например, GL\_TEXTURE\_2D как первый параметр glBindBuffer)

# Ошибки OpenGL

Возможные ошибки:

- GL\_NO\_ERROR – ошибки нет
- GL\_INVALID\_ENUM – недопустимое значения перечисления (например, GL\_TEXTURE\_2D как первый параметр glBindBuffer)
- GL\_INVALID\_VALUE – недопустимое значение числового аргумента (например, отрицательная ширина текстуры в glTexImage2D)

# Ошибки OpenGL

Возможные ошибки:

- GL\_NO\_ERROR – ошибки нет
- GL\_INVALID\_ENUM – недопустимое значения перечисления (например, GL\_TEXTURE\_2D как первый параметр glBindBuffer)
- GL\_INVALID\_VALUE – недопустимое значение числового аргумента (например, отрицательная ширина текстуры в glTexImage2D)
- GL\_INVALID\_OPERATION – недопустимая операция (например, glUniform1i при отсутствии текущей шейдерной программы)

# Ошибки OpenGL

Возможные ошибки:

- GL\_NO\_ERROR – ошибки нет
- GL\_INVALID\_ENUM – недопустимое значения перечисления (например, GL\_TEXTURE\_2D как первый параметр glBindBuffer)
- GL\_INVALID\_VALUE – недопустимое значение числового аргумента (например, отрицательная ширина текстуры в glTexImage2D)
- GL\_INVALID\_OPERATION – недопустимая операция (например, glUniform1i при отсутствии текущей шейдерной программы)
- GL\_INVALID\_FRAMEBUFFER\_OPERATION – операция рисования/чтения пикселей с невалидным framebuffer'ом (о них чуть позже)

# Ошибки OpenGL

Возможные ошибки:

- GL\_NO\_ERROR – ошибки нет
- GL\_INVALID\_ENUM – недопустимое значения перечисления (например, GL\_TEXTURE\_2D как первый параметр glBindBuffer)
- GL\_INVALID\_VALUE – недопустимое значение числового аргумента (например, отрицательная ширина текстуры в glTexImage2D)
- GL\_INVALID\_OPERATION – недопустимая операция (например, glUniform1i при отсутствии текущей шейдерной программы)
- GL\_INVALID\_FRAMEBUFFER\_OPERATION – операция рисования/чтения пикселей с невалидным framebuffer'ом (о них чуть позже)
- GL\_OUT\_OF\_MEMORY – закончилась память на GPU (может быть вызвана любой OpenGL-командой, обычно не обрабатывается)

# Ошибки OpenGL

- Документация к каждой функции описывает все возможные ошибки, которые эта функция может вызвать, и в каких случаях

# Ошибки OpenGL

- Документация к каждой функции описывает все возможные ошибки, которые эта функция может вызвать, и в каких случаях
  - **N.B.** GL\_OUT\_OF\_MEMORY нигде не указан, потому что может появиться где угодно

# Ошибки OpenGL

- Документация к каждой функции описывает все возможные ошибки, которые эта функция может вызвать, и в каких случаях
  - **N.B.** GL\_OUT\_OF\_MEMORY нигде не указан, потому что может появиться где угодно
- Не все ошибки покрываются этим механизмом

# Ошибки OpenGL

- Документация к каждой функции описывает все возможные ошибки, которые эта функция может вызвать, и в каких случаях
  - **N.B.** `GL_OUT_OF_MEMORY` нигде не указан, потому что может появиться где угодно
- Не все ошибки покрываются этим механизмом
  - Например, `glDrawArrays` с правильно настроенным VAO но с пустым VBO с вершинами

# Ошибки OpenGL

- Документация к каждой функции описывает все возможные ошибки, которые эта функция может вызвать, и в каких случаях
  - **N.B.** `GL_OUT_OF_MEMORY` нигде не указан, потому что может появиться где угодно
- Не все ошибки покрываются этим механизмом
  - Например, `glDrawArrays` с правильно настроенным VAO но с пустым VBO с вершинами
- Нет информации о том, какая именно функция вызвала ошибку (любая из тех, что были вызваны до `glGetError`)

# Ошибки OpenGL

- Документация к каждой функции описывает все возможные ошибки, которые эта функция может вызвать, и в каких случаях
  - **N.B.** `GL_OUT_OF_MEMORY` нигде не указан, потому что может появиться где угодно
- Не все ошибки покрываются этим механизмом
  - Например, `glDrawArrays` с правильно настроенным VAO но с пустым VBO с вершинами
- Нет информации о том, какая именно функция вызвала ошибку (любая из тех, что были вызваны до `glGetError`)
  - ⇒ Придётся расставлять проверку после каждого OpenGL-вызова

# Ошибки OpenGL

- Драйвер может хранить один флаг ошибки, и пропускать все последующие ошибки до вызова `glGetError`

# Ошибки OpenGL

- Драйвер может хранить один флаг ошибки, и пропускать все последующие ошибки до вызова `glGetError`
- Драйвер может хранить несколько флагов, и `glGetError` возвращает и очищает любой из них

# Ошибки OpenGL

- Драйвер может хранить один флаг ошибки, и пропускать все последующие ошибки до вызова `glGetError`
- Драйвер может хранить несколько флагов, и `glGetError` возвращает и очищает любой из них
- Драйвер может хранить очередь ошибок, и `glGetError` возвращает их в порядке появления

# Ошибки OpenGL

- Драйвер может хранить один флаг ошибки, и пропускать все последующие ошибки до вызова `glGetError`
- Драйвер может хранить несколько флагов, и `glGetError` возвращает и очищает любой из них
- Драйвер может хранить очередь ошибок, и `glGetError` возвращает их в порядке появления
- ⇒ Чтобы очистить ошибки, нужно вызывать `glGetError` в цикле:

```
while (glGetError() != GL_NO_ERROR);
```

# Ошибки OpenGL

- OpenGL 4.3+ (или расширение GL\_ARB\_debug\_output): более удобный механизм, `glDebugMessageCallback`

# Ошибки OpenGL

- OpenGL 4.3+ (или расширение GL\_ARB\_debug\_output): более удобный механизм, `glDebugMessageCallback`
- [khronos.org/opengl/wiki/OpenGL\\_Error](http://khronos.org/opengl/wiki/OpenGL_Error)
- [docs.gl/gl3/glGetError](http://docs.gl/gl3/glGetError)

# Расширения OpenGL

- Предоставляют дополнительную функциональность за рамками возможностей конкретной версии OpenGL

# Расширения OpenGL

- Предоставляют дополнительную функциональность за рамками возможностей конкретной версии OpenGL
  - Конкретный производитель выпустил новую функциональность

# Расширения OpenGL

- Предоставляют дополнительную функциональность за рамками возможностей конкретной версии OpenGL
  - Конкретный производитель выпустил новую функциональность
  - Функциональность доступна на большинстве реализаций OpenGL, но ещё не успела войти в новую версию OpenGL

# Расширения OpenGL

- Предоставляют дополнительную функциональность за рамками возможностей конкретной версии OpenGL
  - Конкретный производитель выпустил новую функциональность
  - Функциональность доступна на большинстве реализаций OpenGL, но ещё не успела войти в новую версию OpenGL
  - Функциональность доступна в новой версии OpenGL, но крайне распространена и хочется ей пользоваться из старой версии

# Расширения OpenGL

- Имя расширения имеет особый префикс:

# Расширения OpenGL

- Имя расширения имеет особый префикс:
  - ARB (*Architectural Review Board*) – функциональность, которая (скорее всего) войдёт в следующий стандарт

# Расширения OpenGL

- Имя расширения имеет особый префикс:
  - ARB (*Architectural Review Board*) – функциональность, которая (скорее всего) войдёт в следующий стандарт
  - EXT – широко распространённая функциональность

# Расширения OpenGL

- Имя расширения имеет особый префикс:
  - ARB (*Architectural Review Board*) – функциональность, которая (скорее всего) войдёт в следующий стандарт
  - EXT – широко распространённая функциональность
  - NV – расширение от Nvidia (возможно, доступно и на других видеокартах)

# Расширения OpenGL

- Имя расширения имеет особый префикс:
  - ARB (*Architectural Review Board*) – функциональность, которая (скорее всего) войдёт в следующий стандарт
  - EXT – широко распространённая функциональность
  - NV – расширение от Nvidia (возможно, доступно и на других видеокартах)
  - AMD – расширение от AMD (возможно, доступно и на других видеокартах)

# Расширения OpenGL

- Имя расширения имеет особый префикс:
  - ARB (*Architectural Review Board*) – функциональность, которая (скорее всего) войдёт в следующий стандарт
  - EXT – широко распространённая функциональность
  - NV – расширение от Nvidia (возможно, доступно и на других видеокартах)
  - AMD – расширение от AMD (возможно, доступно и на других видеокартах)
  - APPLE – расширение от APPLE (возможно, доступно и на других видеокартах)

# Расширения OpenGL

- Имя расширения имеет особый префикс:
  - ARB (*Architectural Review Board*) – функциональность, которая (скорее всего) войдёт в следующий стандарт
  - EXT – широко распространённая функциональность
  - NV – расширение от Nvidia (возможно, доступно и на других видеокартах)
  - AMD – расширение от AMD (возможно, доступно и на других видеокартах)
  - APPLE – расширение от APPLE (возможно, доступно и на других видеокартах)
  - И т.д.

# Расширения OpenGL

- Расширение – набор констант и функций (как и конкретная версия OpenGL)

# Расширения OpenGL

- Расширение – набор констант и функций (как и конкретная версия OpenGL)
- Функции нужно загружать так же, как и функции самого OpenGL

# Расширения OpenGL

- Расширение – набор констант и функций (как и конкретная версия OpenGL)
- Функции нужно загружать так же, как и функции самого OpenGL
- Константы и перечисления заканчиваются на суффикс в зависимости от типа расширения:
  - ARB\_debug\_output: glDebugMessageCallbackARB,  
GL\_DEBUG\_SEVERITY\_HIGH\_ARB
  - EXT\_texture\_filter\_anisotropic:  
GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT
  - NV\_texture\_barrier: glTextureBarrierNV

# Расширения OpenGL

- Расширение – набор констант и функций (как и конкретная версия OpenGL)
- Функции нужно загружать так же, как и функции самого OpenGL
- Константы и перечисления заканчиваются на суффикс в зависимости от типа расширения:
  - ARB\_debug\_output: glDebugMessageCallbackARB,  
GL\_DEBUG\_SEVERITY\_HIGH\_ARB
  - EXT\_texture\_filter\_anisotropic:  
GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT
  - NV\_texture\_barrier: glTextureBarrierNV
- Исключение: *core extensions* – предоставляют функциональность новых версий OpenGL в старых версиях OpenGL
  - Имеют префикс ARB
  - Не имеют суффиксов в названиях функций и констант

# Расширения OpenGL

- Получить список поддерживаемых расширений:

```
GLint numExtensions;
glGetIntegerv(GL_NUM_EXTENSIONS, &numExtensions);
for (GLint i = 0; i < numExtensions; ++i) {
    std::cout << glGetStringi(GL_EXTENSIONS, i) << "\n";
}
```

# Расширения OpenGL

- Получить список поддерживаемых расширений:

```
GLint numExtensions;
glGetIntegerv(GL_NUM_EXTENSIONS, &numExtensions);
for (GLint i = 0; i < numExtensions; ++i) {
    std::cout << glGetStringi(GL_EXTENSIONS, i) << "\n";
}
```

- GLEW загружает их автоматически:

```
if (GLEW_EXT_texture_filter_anisotropic) {
    std::cout << "Anisotropic filtering is supported\n";
}
```

## Расширения OpenGL

- [khronos.org/opengl/wiki/OpenGL\\_Extension](http://khronos.org/opengl/wiki/OpenGL_Extension)
- [khronos.org/registry/OpenGL/index\\_gl.php](http://khronos.org/registry/OpenGL/index_gl.php) – список всех зарегистрированных расширений

## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого

## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания

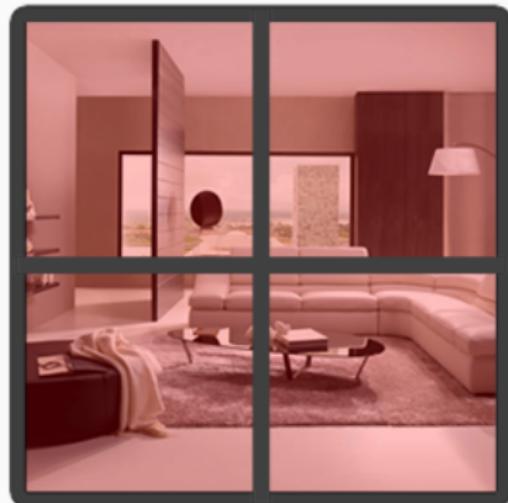
## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты

# Полупрозрачность

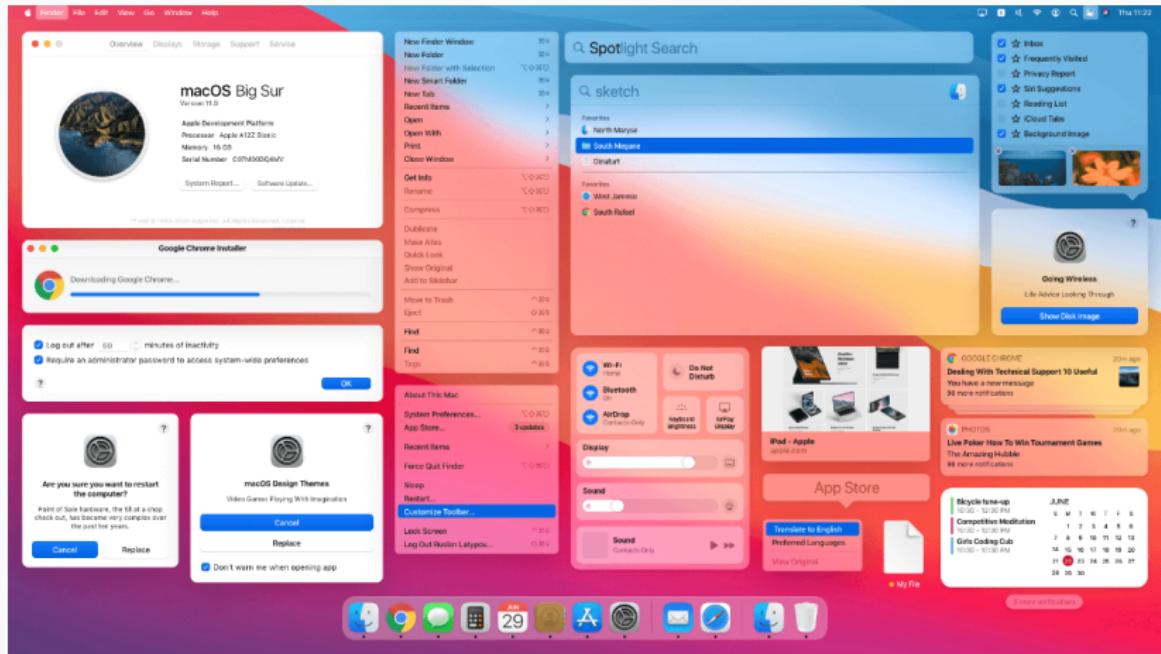


Full transparent window



Partially transparent window

# Полупрозрачность



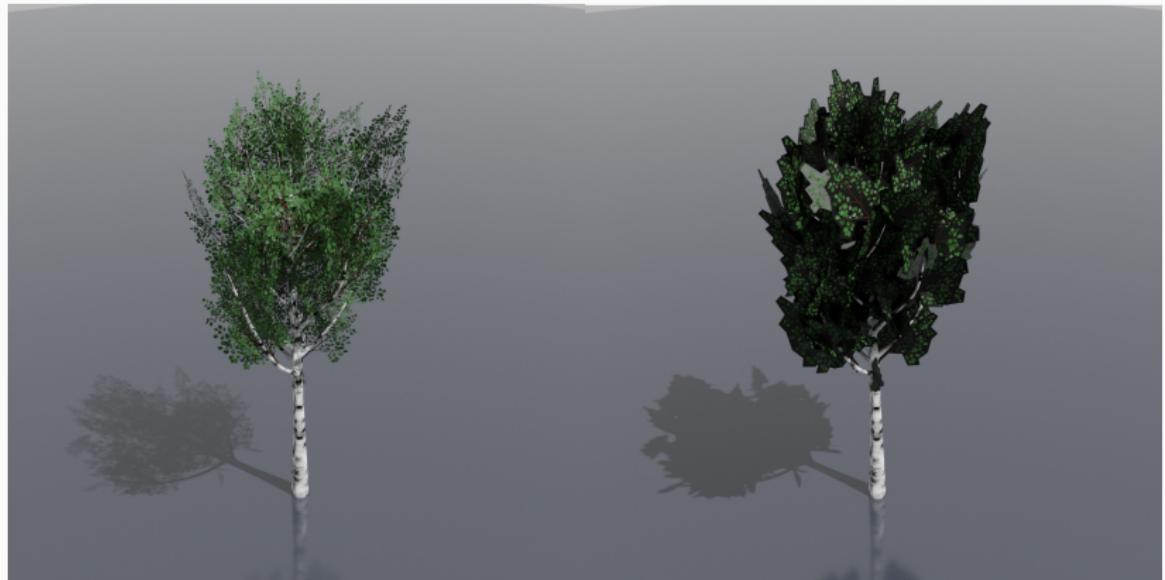
## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты

## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность

# Деревья



## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность

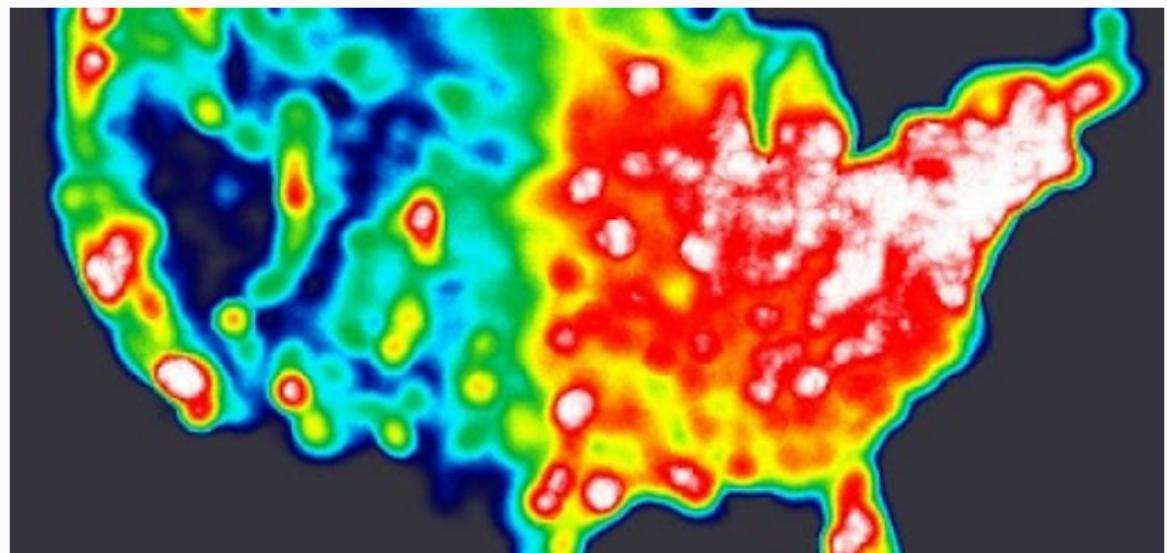
## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность
  - Алгоритмы освещения и теней

## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность
  - Алгоритмы освещения и теней
  - Heatmaps

# Heatmap

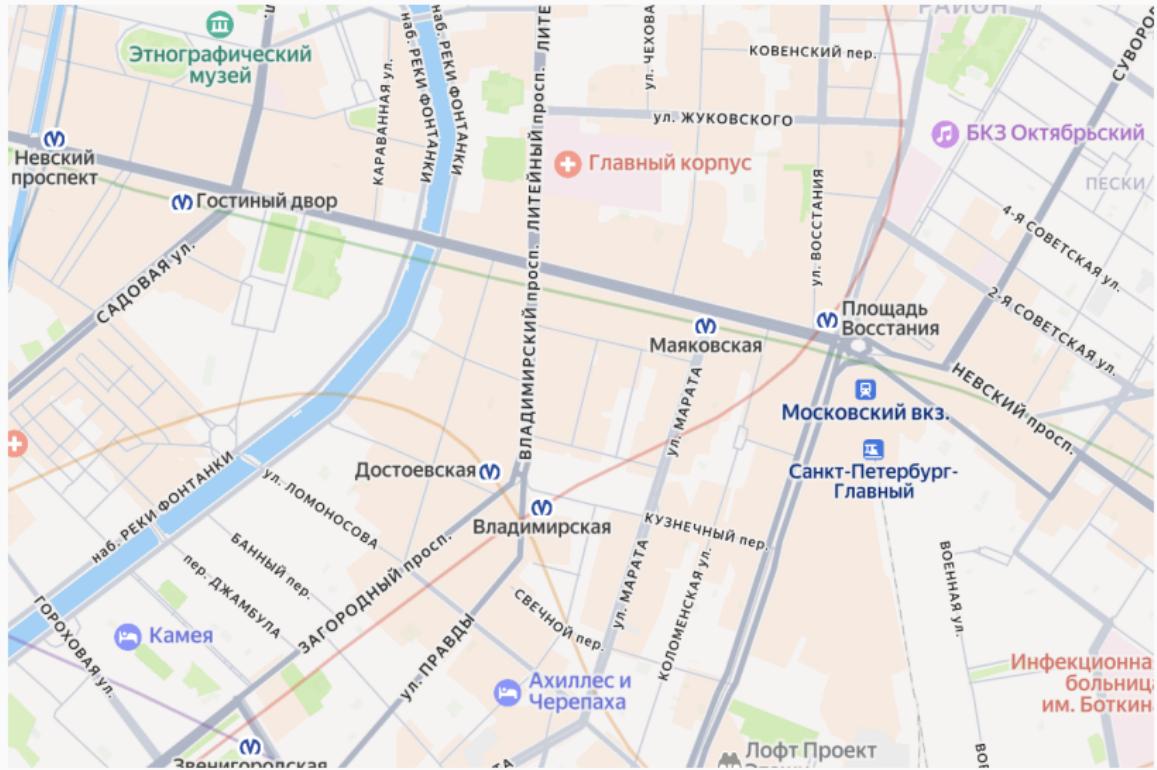


## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность
  - Алгоритмы освещения и теней
  - Heatmaps

## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность
  - Алгоритмы освещения и теней
  - Heatmaps
  - Ручное сглаживание



## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность
  - Алгоритмы освещения и теней
  - Heatmaps
  - Ручное сглаживание

## Blending (смешивание)

- По умолчанию результат фрагментного шейдера (`layout (location = 0) vec4 out_color;`) записывается в пиксель экрана, стирая то, что было в нём до этого
- Иногда мы хотим ‘смешать’ результат и пиксель экрана, и записать результат смешивания
  - Полупрозрачные объекты
  - Растительность
  - Алгоритмы освещения и теней
  - Heatmaps
  - Ручное сглаживание
  - И т.д.

## Blending (смешивание)

- Включается/выключается: `glEnable(GL_BLEND)`

## Blending (смешивание)

- Включается/выключается: `glEnable(GL_BLEND)`
- Уравнение blending'a:

$$dst \leftarrow f(C_{src} \cdot src, C_{dst} \cdot dst)$$

## Blending (смешивание)

- Включается/выключается: `glEnable(GL_BLEND)`
- Уравнение blending'a:

$$dst \leftarrow f(C_{src} \cdot src, C_{dst} \cdot dst)$$

- $src$  – результат фрагментного шейдера

## Blending (смешивание)

- Включается/выключается: `glEnable(GL_BLEND)`
- Уравнение blending'a:

$$dst \leftarrow f(C_{src} \cdot src, C_{dst} \cdot dst)$$

- $src$  – результат фрагментного шейдера
- $dst$  – пиксель на экране

## Blending (смешивание)

- Включается/выключается: `glEnable(GL_BLEND)`
- Уравнение blending'a:

$$dst \leftarrow f(C_{src} \cdot src, C_{dst} \cdot dst)$$

- $src$  – результат фрагментного шейдера
- $dst$  – пиксель на экране
- $f$  – настраиваемая функция

## Blending (смешивание)

- Включается/выключается: `glEnable(GL_BLEND)`
- Уравнение blending'a:

$$dst \leftarrow f(C_{src} \cdot src, C_{dst} \cdot dst)$$

- $src$  – результат фрагментного шейдера
- $dst$  – пиксель на экране
- $f$  – настраиваемая функция
- $C_{src}$  и  $C_{dst}$  – настраиваемые веса

## Blending (смешивание)

- Настройка функции  $f$ : `glBlendEquation(GLenum equation)`:

## Blending (смешивание)

- Настройка функции  $f$ : `glBlendEquation(GLenum equation)`:
  - `GL_FUNC_ADD`:  $f(src, dst) = src + dst$

## Blending (смешивание)

- Настройка функции  $f$ : `glBlendEquation(GLenum equation)`:
  - `GL_FUNC_ADD`:  $f(src, dst) = src + dst$
  - `GL_FUNC_SUBTRACT`:  $f(src, dst) = src - dst$

## Blending (смешивание)

- Настройка функции  $f$ : `glBlendEquation(GLenum equation)`:
  - `GL_FUNC_ADD`:  $f(src, dst) = src + dst$
  - `GL_FUNC_SUBTRACT`:  $f(src, dst) = src - dst$
  - `GL_FUNC_REVERSE_SUBTRACT`:  $f(src, dst) = dst - src$

## Blending (смешивание)

- Настройка функции  $f$ : `glBlendEquation(GLenum equation)`:
  - `GL_FUNC_ADD`:  $f(src, dst) = src + dst$
  - `GL_FUNC_SUBTRACT`:  $f(src, dst) = src - dst$
  - `GL_FUNC_REVERSE_SUBTRACT`:  $f(src, dst) = dst - src$
  - `GL_MIN`:  $f(src, dst) = \min(src, dst)$

## Blending (смешивание)

- Настройка функции  $f$ : `glBlendEquation(GLenum equation)`:
  - `GL_FUNC_ADD`:  $f(src, dst) = src + dst$
  - `GL_FUNC_SUBTRACT`:  $f(src, dst) = src - dst$
  - `GL_FUNC_REVERSE_SUBTRACT`:  $f(src, dst) = dst - src$
  - `GL_MIN`:  $f(src, dst) = \min(src, dst)$
  - `GL_MAX`:  $f(src, dst) = \max(src, dst)$

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :

```
glBlendFunc(GLenum src, GLenum dst):
```

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :  
`glBlendFunc(GLenum src, GLenum dst):`
  - `GL_ZERO`:  $C = 0$

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :

```
glBlendFunc(GLenum src, GLenum dst):
```

- GL\_ZERO:  $C = 0$
- GL\_ONE:  $C = 1$

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :

```
glBlendFunc(GLenum src, GLenum dst):
```

- GL\_ZERO:  $C = 0$
- GL\_ONE:  $C = 1$
- GL\_SRC\_COLOR:  $C = src$

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :

```
glBlendFunc(GLenum src, GLenum dst):
```

- GL\_ZERO:  $C = 0$
- GL\_ONE:  $C = 1$
- GL\_SRC\_COLOR:  $C = src$
- GL\_ONE\_MINUS\_SRC\_COLOR:  $C = 1 - src$

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :

```
glBlendFunc(GLenum src, GLenum dst):
```

- GL\_ZERO:  $C = 0$
- GL\_ONE:  $C = 1$
- GL\_SRC\_COLOR:  $C = src$
- GL\_ONE\_MINUS\_SRC\_COLOR:  $C = 1 - src$
- GL\_SRC\_ALPHA:  $C = src_A$

## Blending (смешивание)

- Настройка весов  $C_{src}$  и  $C_{dst}$ :

```
glBlendFunc(GLenum src, GLenum dst):
```

- GL\_ZERO:  $C = 0$
- GL\_ONE:  $C = 1$
- GL\_SRC\_COLOR:  $C = src$
- GL\_ONE\_MINUS\_SRC\_COLOR:  $C = 1 - src$
- GL\_SRC\_ALPHA:  $C = src_A$
- GL\_ONE\_MINUS\_SRC\_ALPHA:  $C = 1 - src_A$
- И т.д.

## Blending (смешивание)

- Обычно  $src = (R, G, B, A)$  означает, что цвет имеет полуупрозрачность (точнее – *непрозрачность*, или *opacity*) равную  $A$ 
  - $A = 0$  – полностью прозрачный цвет
  - $A = 1$  – полностью непрозрачный цвет

## Blending (смешивание)

- Обычно  $src = (R, G, B, A)$  означает, что цвет имеет полуупрозрачность (точнее – непрозрачность, или *opacity*) равную  $A$ 
  - $A = 0$  – полностью прозрачный цвет
  - $A = 1$  – полностью непрозрачный цвет
- Для этого нужна такая формула смешивания:

$$dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst = \text{lerp}(dst, src, src_A)$$

## Blending (смешивание)

- Обычно  $src = (R, G, B, A)$  означает, что цвет имеет полуупрозрачность (точнее – непрозрачность, или *opacity*) равную  $A$ 
  - $A = 0$  – полностью прозрачный цвет
  - $A = 1$  – полностью непропускающий цвет
- Для этого нужна такая формула смешивания:

$$dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst = \text{lerp}(dst, src, src_A)$$

- Этому соответствует настройка

```
glBlendEquation(GL_FUNC_ADD);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

## Blending (смешивание)

- Обычно  $src = (R, G, B, A)$  означает, что цвет имеет полупрозрачность (точнее – непрозрачность, или *opacity*) равную  $A$ 
  - $A = 0$  – полностью прозрачный цвет
  - $A = 1$  – полностью непрозрачный цвет
- Для этого нужна такая формула смешивания:

$$dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst = \text{lerp}(dst, src, src_A)$$

- Этому соответствует настройка

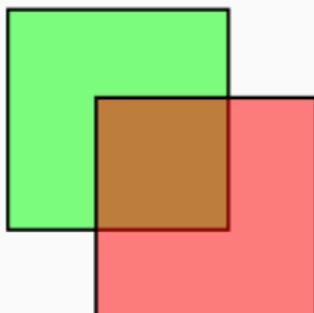
```
glBlendEquation(GL_FUNC_ADD);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- Это *один из самых типичных* способов использовать blending (но не самый правильный)

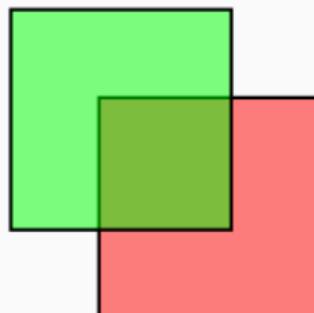
## Blending (смешивание)

- Такое смешивание некоммутативно, т.е. результат зависит от порядка рисования
- Бывают конкретные ситуации, когда коммутативность не нарушается (наложение двух объектов с  $A = 0.5$  или наложение нескольких объектов одинакового цвета)

Red on top

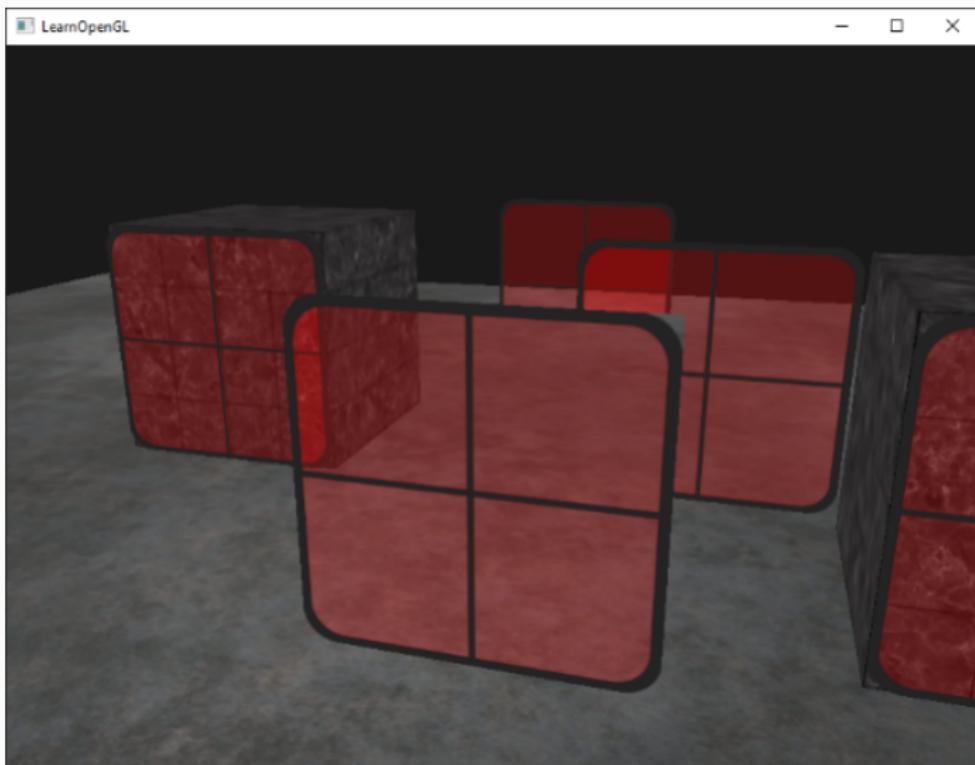


Green on top



## Blending (смешивание)

- Смешивание некорректно работает с тестом глубины



## Blending (смешивание)

- В общем случае нужно сортировать полупрозрачные объекты и рисовать в порядке уменьшения расстояния до камеры

## Blending (смешивание)

- В общем случае нужно сортировать полупрозрачные объекты и рисовать в порядке уменьшения расстояния до камеры
  - Что именно считать расстоянием до камеры (расстояние от центра объекта / от ближайшей вершины / от самой далёкой вершины / среднее между минимальным и максимальным расстоянием) – большой вопрос

## Blending (смешивание)

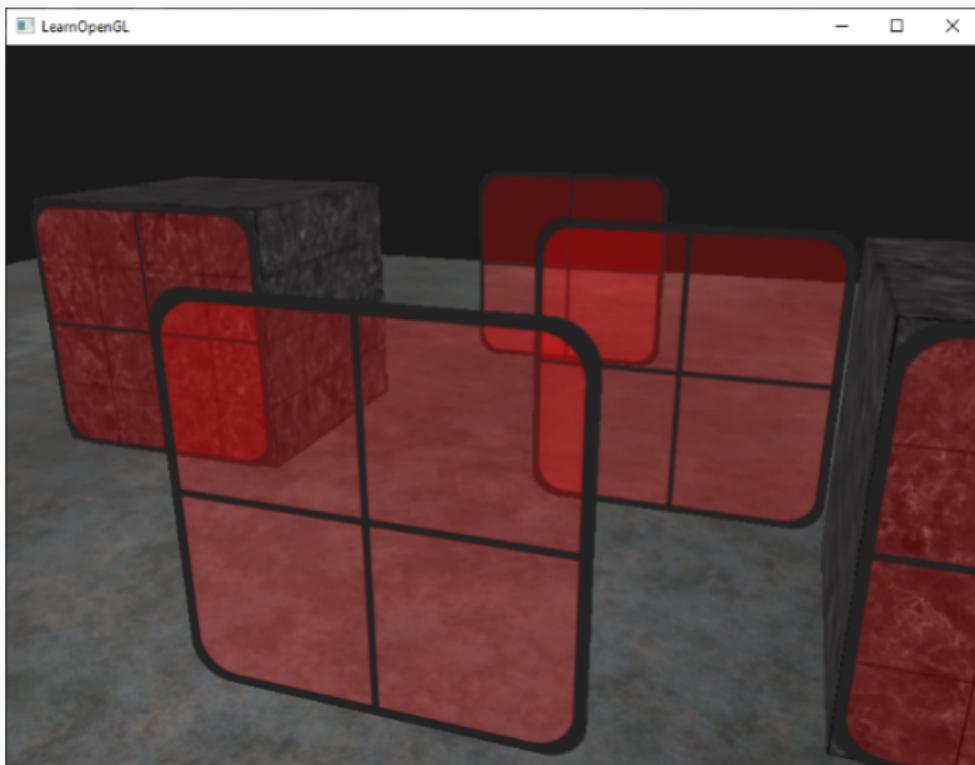
- В общем случае нужно сортировать полупрозрачные объекты и рисовать в порядке уменьшения расстояния до камеры
  - Что именно считать расстоянием до камеры (расстояние от центра объекта / от ближайшей вершины / от самой далёкой вершины / среднее между минимальным и максимальным расстоянием) – большой вопрос
  - Для любого варианта можно найти примеры, ломающие его (цикл между объектами, накрывающими друг друга)

## Blending (смешивание)

- В общем случае нужно сортировать полупрозрачные объекты и рисовать в порядке уменьшения расстояния до камеры
  - Что именно считать расстоянием до камеры (расстояние от центра объекта / от ближайшей вершины / от самой далёкой вершины / среднее между минимальным и максимальным расстоянием) – большой вопрос
  - Для любого варианта можно найти примеры, ломающие его (цикл между объектами, накрывающими друг друга)
- Order-independent transparency (OIT) – активная тема исследований

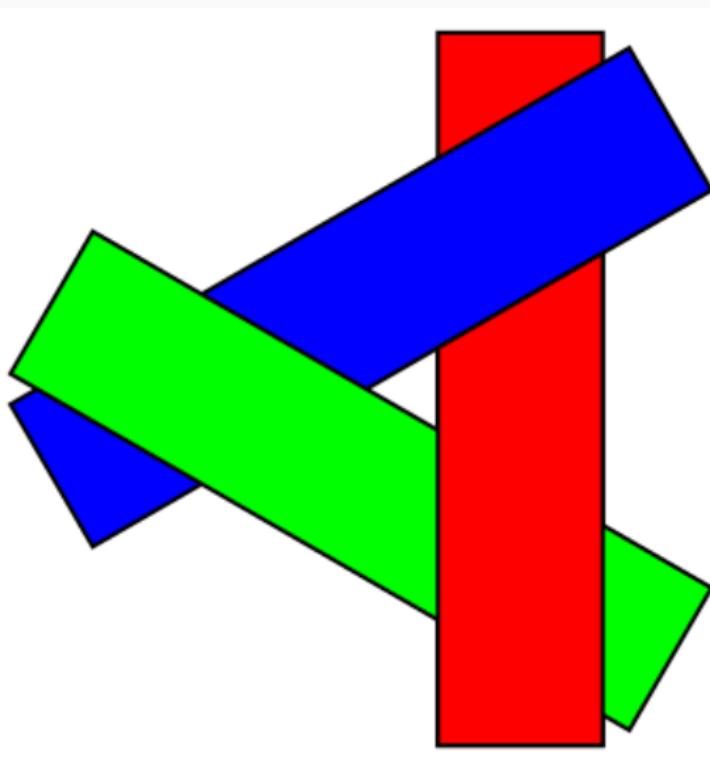
## Blending (смешивание)

- Сортировка по расстоянию до камеры



## Blending (смешивание)

- Случай, когда сортировка не поможет



## Аддитивное смешивание

- Один часто используемый вариант смешивания – аддитивное смешивание
- $dst \leftarrow src + dst$  или  $dst \leftarrow src_A \cdot src + dst$

## Аддитивное смешивание

- Один часто используемый вариант смешивания – аддитивное смешивание
- $dst \leftarrow src + dst$  или  $dst \leftarrow src_A \cdot src + dst$
- `glBlendEquation(GL_FUNC_ADD)`
- `glBlendFunc(GL_ONE, GL_ONE)` или  
`glBlendFunc(GL_SRC_ALPHA, GL_ONE)`

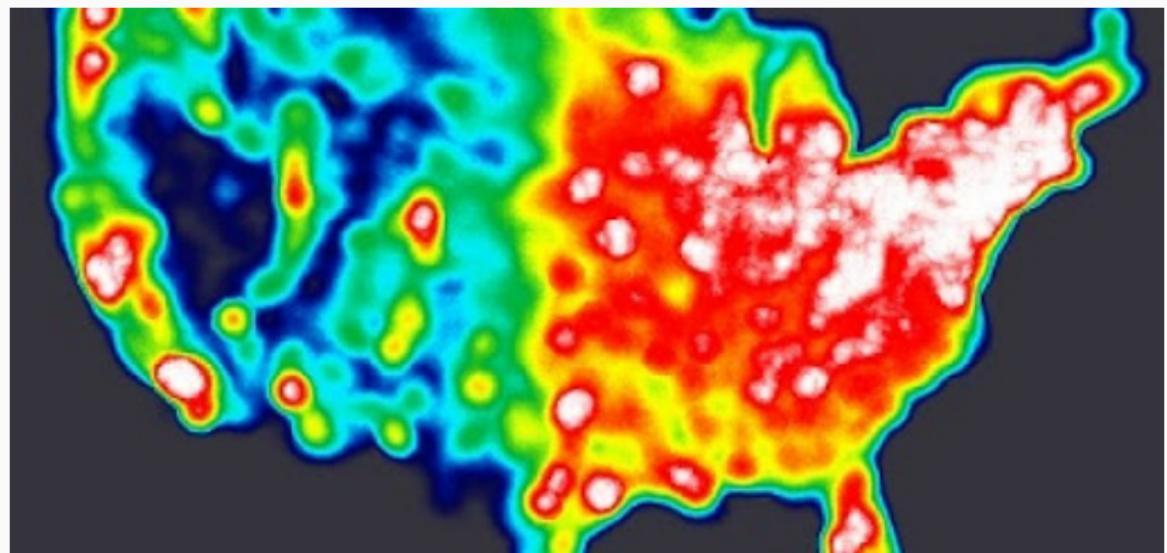
## Аддитивное смешивание

- Один часто используемый вариант смешивания – аддитивное смешивание
- $dst \leftarrow src + dst$  или  $dst \leftarrow src_A \cdot src + dst$
- `glBlendEquation(GL_FUNC_ADD)`
- `glBlendFunc(GL_ONE, GL_ONE)` или  
`glBlendFunc(GL_SRC_ALPHA, GL_ONE)`
- Такое смешивание коммутативно  $\implies$  не важно, в каком порядке рисовать объекты

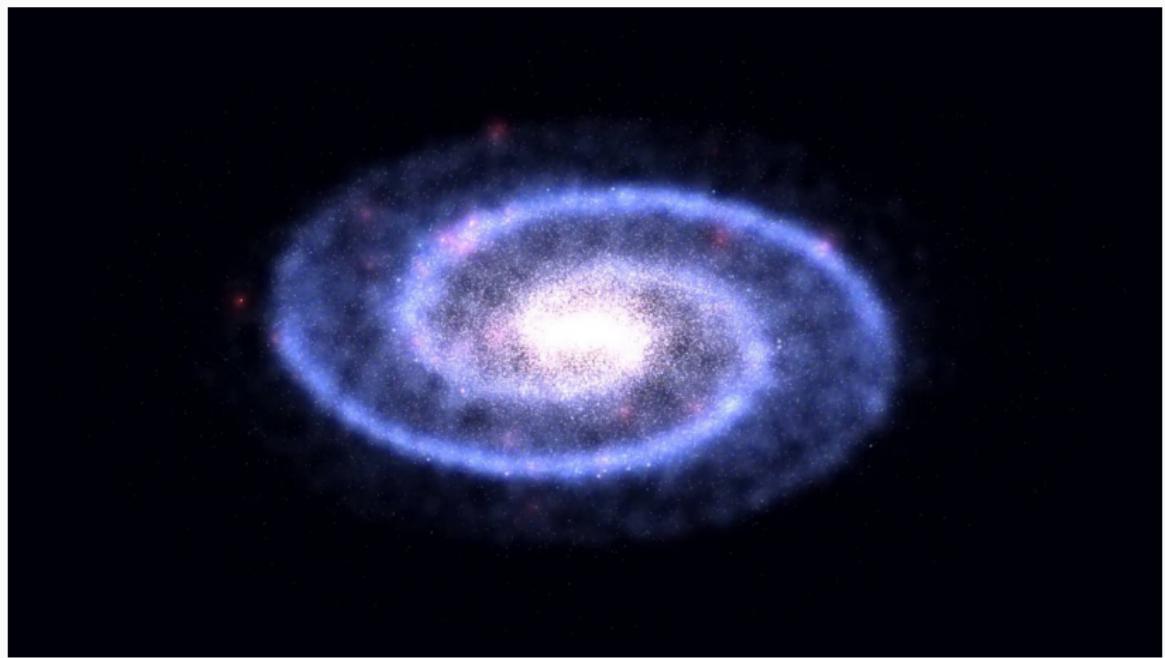
## Аддитивное смешивание

- Один часто используемый вариант смешивания – аддитивное смешивание
- $dst \leftarrow src + dst$  или  $dst \leftarrow src_A \cdot src + dst$
- `glBlendEquation(GL_FUNC_ADD)`
- `glBlendFunc(GL_ONE, GL_ONE)` или  
`glBlendFunc(GL_SRC_ALPHA, GL_ONE)`
- Такое смешивание коммутативно  $\implies$  не важно, в каком порядке рисовать объекты
- Heatmaps, облака точек

## Аддитивное смешивание



## Аддитивное смешивание



## Blending (смешивание)

- Формула  $dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$  работает только если destination – непрозрачный цвет

## Blending (смешивание)

- Формула  $dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$  работает только если destination – непрозрачный цвет
- Если мы хотим наложить несколько полупрозрачных объектов, и получить суммарную полупрозрачную картинку, нужна более сложная формула

## Blending (смешивание)

- Формула  $dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$  работает только если destination – непрозрачный цвет
- Если мы хотим наложить несколько полупрозрачных объектов, и получить суммарную полупрозрачную картинку, нужна более сложная формула
- Пусть есть цвет  $C_0$ , на который накладывается цвет  $C_1$  с opacity  $a_1$ , а потом  $C_2$  с opacity  $a_2$ : получится цвет

## Blending (смешивание)

- Формула  $dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$  работает только если destination – непрозрачный цвет
- Если мы хотим наложить несколько полупрозрачных объектов, и получить суммарную полупрозрачную картинку, нужна более сложная формула
- Пусть есть цвет  $C_0$ , на который накладывается цвет  $C_1$  с opacity  $a_1$ , а потом  $C_2$  с opacity  $a_2$ : получится цвет

$$C_2a_2 + C_1a_1(1 - a_2) + C_0(1 - a_1)(1 - a_2)$$

## Blending (смешивание)

- Формула  $dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$  работает только если destination – непрозрачный цвет
- Если мы хотим наложить несколько полупрозрачных объектов, и получить суммарную полупрозрачную картинку, нужна более сложная формула
- Пусть есть цвет  $C_0$ , на который накладывается цвет  $C_1$  с opacity  $a_1$ , а потом  $C_2$  с opacity  $a_2$ : получится цвет

$$C_2a_2 + C_1a_1(1 - a_2) + C_0(1 - a_1)(1 - a_2)$$

- Мы хотим привести это к виду  $C'a' + C_0(1 - a')$

## Blending (смешивание)

- Мы хотим привести это к виду  $C'a' + C_0(1 - a')$

## Blending (смешивание)

- Мы хотим привести это к виду  $C'a' + C_0(1 - a')$
- Отсюда

$$a' = 1 - (1 - a_1)(1 - a_2) = a_1 + a_2 - a_1a_2 = a_2 + a_1(1 - a_2)$$

$$C' = \frac{C_2a_2 + C_1a_1(1 - a_2)}{a_2 + a_1(1 - a_2)}$$

## Blending (смешивание)

- Мы хотим привести это к виду  $C'a' + C_0(1 - a')$
- Отсюда

$$a' = 1 - (1 - a_1)(1 - a_2) = a_1 + a_2 - a_1a_2 = a_2 + a_1(1 - a_2)$$

$$C' = \frac{C_2a_2 + C_1a_1(1 - a_2)}{a_2 + a_1(1 - a_2)}$$

- Это т.н. *over operator* смешивания (см. Porter, Duff'84)

## Blending (смешивание)

- Мы хотим привести это к виду  $C'a' + C_0(1 - a')$
- Отсюда

$$a' = 1 - (1 - a_1)(1 - a_2) = a_1 + a_2 - a_1a_2 = a_2 + a_1(1 - a_2)$$

$$C' = \frac{C_2a_2 + C_1a_1(1 - a_2)}{a_2 + a_1(1 - a_2)}$$

- Это т.н. *over operator* смешивания (см. Porter, Duff'84)
- Такую функцию **не выразить** в OpenGL: блендинг не умеет разделить результирующий цвет на его альфа-канал

## Blending (смешивание)

- Мы хотим привести это к виду  $C'a' + C_0(1 - a')$
- Отсюда

$$a' = 1 - (1 - a_1)(1 - a_2) = a_1 + a_2 - a_1a_2 = a_2 + a_1(1 - a_2)$$

$$C' = \frac{C_2a_2 + C_1a_1(1 - a_2)}{a_2 + a_1(1 - a_2)}$$

- Это т.н. *over operator* смешивания (см. Porter, Duff'84)
- Такую функцию **не выразить** в OpenGL: блендинг не умеет разделить результирующий цвет на его альфа-канал
- ⇒ Решение: *премультиплицированный* альфа-канал

## Premultiplied alpha

- Идея: давайте хранить цвета не в виде  $(R, G, B, A)$ , а в виде  $(R \cdot A, G \cdot A, B \cdot A, A)$

## Premultiplied alpha

- Идея: давайте хранить цвета не в виде  $(R, G, B, A)$ , а в виде  $(R \cdot A, G \cdot A, B \cdot A, A)$
- Тогда формула blending'a заменится на

$$a' = a_2 + a_1(1 - a_2)$$

$$C' \cdot a' = (C_2 \cdot a_2) + (C_1 \cdot a_1)(1 - a_2)$$

# Premultiplied alpha

- Идея: давайте хранить цвета не в виде  $(R, G, B, A)$ , а в виде  $(R \cdot A, G \cdot A, B \cdot A, A)$
- Тогда формула blending'a заменится на

$$a' = a_2 + a_1(1 - a_2)$$

$$C' \cdot a' = (C_2 \cdot a_2) + (C_1 \cdot a_1)(1 - a_2)$$

- $\implies$  Если и входные, и выходные цвета премультиплицированные, то правильный blending можно выразить встроенными операциями:

```
glBlendEquation(GL_FUNC_ADD);
glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
```

## Premultiplied alpha: mipmaps

- При обычной генерации mipmaps для полупрозрачных изображений есть проблема: очень прозрачные или полностью прозрачные пиксели дают такой же вклад в результат, как и непрозрачные

## Premultiplied alpha: mipmaps

- При обычной генерации mipmaps для полупрозрачных изображений есть проблема: очень прозрачные или полностью прозрачные пиксели дают такой же вклад в результат, как и непрозрачные
- Пример: три чёрных прозрачных пикселя  $(R, G, B, A) = (0, 0, 0, 0)$  усредняются с одним красным непрозрачным  $(R, G, B, A) = (1, 0, 0, 1)$ , и получается тёмно-красный пиксель с opacity 25%:  $(0.25, 0, 0, 0.25)$

## Premultiplied alpha: mipmaps

- При обычной генерации mipmaps для полупрозрачных изображений есть проблема: очень прозрачные или полностью прозрачные пиксели дают такой же вклад в результат, как и непрозрачные
- Пример: три чёрных прозрачных пикселя  $(R, G, B, A) = (0, 0, 0, 0)$  усредняются с одним красным непрозрачным  $(R, G, B, A) = (1, 0, 0, 1)$ , и получается тёмно-красный пиксель с opacity 25%:  $(0.25, 0, 0, 0.25)$
- Значение opacity верное, но значение цвета – нет: пиксель должен был остаться красным

## Premultiplied alpha: mipmaps

- При обычной генерации mipmaps для полупрозрачных изображений есть проблема: очень прозрачные или полностью прозрачные пиксели дают такой же вклад в результат, как и непрозрачные
- Пример: три чёрных прозрачных пикселя  $(R, G, B, A) = (0, 0, 0, 0)$  усредняются с одним красным непрозрачным  $(R, G, B, A) = (1, 0, 0, 1)$ , и получается тёмно-красный пиксель с opacity 25%:  $(0.25, 0, 0, 0.25)$
- Значение opacity верное, но значение цвета – нет: пиксель должен был остаться красным
- Если хранить пиксели текстуры в премультиплицированном виде, проблема отпадает: цвета автоматически усредняются пропорционально значениям альфа-канала:

$$a' = \frac{1}{4}(a_0 + a_1 + a_2 + a_3)$$

$$C' \cdot a' = \frac{1}{4}(C_0 \cdot a_0 + C_1 \cdot a_1 + C_2 \cdot a_2 + C_3 \cdot a_3)$$

## Premultiplied alpha

- У такого способа хранения текстур есть минус: значения цветовых каналов ограничены значением альфа-канала, и теряется их точность

## Premultiplied alpha

- У такого способа хранения текстур есть минус: значения цветовых каналов ограничены значением альфа-канала, и теряется их точность
- Обычно это не проблема: если пиксель сильно прозрачный, то слабая точность его цвета всё равно слабо заметна

## Premultiplied alpha

- У такого способа хранения текстур есть минус: значения цветовых каналов ограничены значением альфа-канала, и теряется их точность
- Обычно это не проблема: если пиксель сильно прозрачный, то слабая точность его цвета всё равно слабо заметна
- В общем случае, если вы делаете хоть что-то нетривиальное с полупрозрачностью, лучше **сразу использовать** премультиплицированные цвета

## Blending (смешивание)

- [khronos.org/opengl/wiki/Blending](http://khronos.org/opengl/wiki/Blending)
- [opengl-tutorial.org/intermediate-tutorials/tutorial-10-transparency](http://opengl-tutorial.org/intermediate-tutorials/tutorial-10-transparency)
- [learnopengl.com/Advanced-OpenGL/Blending](http://learnopengl.com/Advanced-OpenGL/Blending)
- Porter, Duff - Compositing Digital Images

## Stencil buffer (буфер трафарета)

- Особый буфер, хранящий произвольные данные (почти всегда unsigned 8-bit на каждый пиксель), с особыми побитовыми операциями над ними

## Stencil buffer (буфер трафарета)

- Особый буфер, хранящий произвольные данные (почти всегда unsigned 8-bit на каждый пиксель), с особыми побитовыми операциями над ними
- Чем-то аналогичен буферу глубины: тоже хранит какие-то данные, такого же размера (как и цветовой буфер), тоже позволяет рисовать или не рисовать пиксель по какому-то условию (stencil test – происходит перед depth test'ом)

## Stencil buffer (буфер трафарета)

- Особый буфер, хранящий произвольные данные (почти всегда unsigned 8-bit на каждый пиксель), с особыми побитовыми операциями над ними
- Чем-то аналогичен буферу глубины: тоже хранит какие-то данные, такого же размера (как и цветовой буфер), тоже позволяет рисовать или не рисовать пиксель по какому-то условию (stencil test – происходит перед depth test'ом)
- У дефолтного фреймбуфера часто есть свой stencil буфер (зависит от настроек контекста OpenGL)

## Stencil buffer (буфер трафарета)

- Особый буфер, хранящий произвольные данные (почти всегда unsigned 8-bit на каждый пиксель), с особыми побитовыми операциями над ними
- Чем-то аналогичен буферу глубины: тоже хранит какие-то данные, такого же размера (как и цветовой буфер), тоже позволяет рисовать или не рисовать пиксель по какому-то условию (stencil test – происходит перед depth test'ом)
- У дефолтного фреймбуфера часто есть свой stencil буфер (зависит от настроек контекста OpenGL)
- Можно (и нужно, если вы его используете) очищать как `glClear(GL_STENCIL_BUFFER_BIT)`

## Stencil buffer (буфер трафарета)

- Особый буфер, хранящий произвольные данные (почти всегда unsigned 8-bit на каждый пиксель), с особыми побитовыми операциями над ними
- Чем-то аналогичен буферу глубины: тоже хранит какие-то данные, такого же размера (как и цветовой буфер), тоже позволяет рисовать или не рисовать пиксель по какому-то условию (stencil test – происходит перед depth test'ом)
- У дефолтного фреймбуфера часто есть свой stencil буфер (зависит от настроек контекста OpenGL)
- Можно (и нужно, если вы его используете) очищать как `glClear(GL_STENCIL_BUFFER_BIT)`
- Настроить значение, которым очищается буфер:  
`glClearStencil`

## Stencil тест

- Включить/выключить stencil тест:  
`glEnable/glDisable(GL_STENCIL_TEST)`

# Stencil тест

- Включить/выключить stencil тест:  
`glEnable/glDisable(GL_STENCIL_TEST)`
- Настроить stencil тест: `glStencilFunc(func, ref, mask)`
  - func – одна из констант GL\_ALWAYS, GL\_LESS, ...
  - ref – референсное значение для теста
  - mask – побитовая маска для теста
- Stencil тест: `func(ref & mask, stencil & mask)`

# Stencil тест

- Включить/выключить stencil тест:  
`glEnable/glDisable(GL_STENCIL_TEST)`
- Настроить stencil тест: `glStencilFunc(func, ref, mask)`
  - func – одна из констант `GL_ALWAYS`, `GL_LESS`, ...
  - ref – референсное значение для теста
  - mask – побитовая маска для теста
- Stencil тест: `func(ref & mask, stencil & mask)`
- Так же, как с depth тестом: если stencil тест не прошёл, пиксель не будет нарисован

## Stencil тест

- Как записать значение в stencil буфер?

## Stencil тест

- Как записать значение в stencil буфер?  
`glStencilOp(sfail, dpfail, dppass)`

## Stencil тест

- Как записать значение в stencil буфер?  
`glStencilOp(sfail, dpfail, dppass)`
  - `sfail` – что делать, если пиксель не прошёл stencil тест
  - `dpfail` – что делать, если пиксель прошёл stencil тест, но не прошёл depth тест
  - `dppass` – что делать, если пиксель прошёл оба теста

# Stencil тест

- Возможные значение `sfail`, `dppfail` И `dppass`:
  - `GL_KEEP` – не менять записанное значение
  - `GL_ZERO` – записать 0
  - `GL_INVERT` – побитово обратить
  - `GL_REPLACE` – записать `ref` из функции `glStencilFunc`
  - `GL_INCR` – увеличить на 1, если значение меньше максимального
  - `GL_DECR` – уменьшить на 1, если значение больше минимального (0)
  - `GL_INCR_WRAP` – увеличить на 1 с целочисленным переполнением
  - `GL_DECR_WRAP` – уменьшить на 1 с целочисленным переполнением

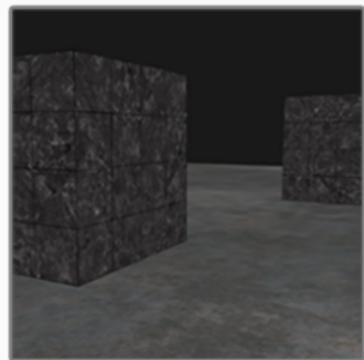
## Stencil тест

- Дополнительно можно включать/выключать запись отдельных битов stencil буфера: `glStencilMask`

## Stencil тест

- Дополнительно можно включать/выключать запись отдельных битов stencil буфера: `glStencilMask`
- Все параметры stencil теста можно настраивать отдельно для front и back граней функциями `glStencilFuncSeparate`, `glStencilOpSeparate`, `glStencilMaskSeparate`

## Stencil test



## Color buffer

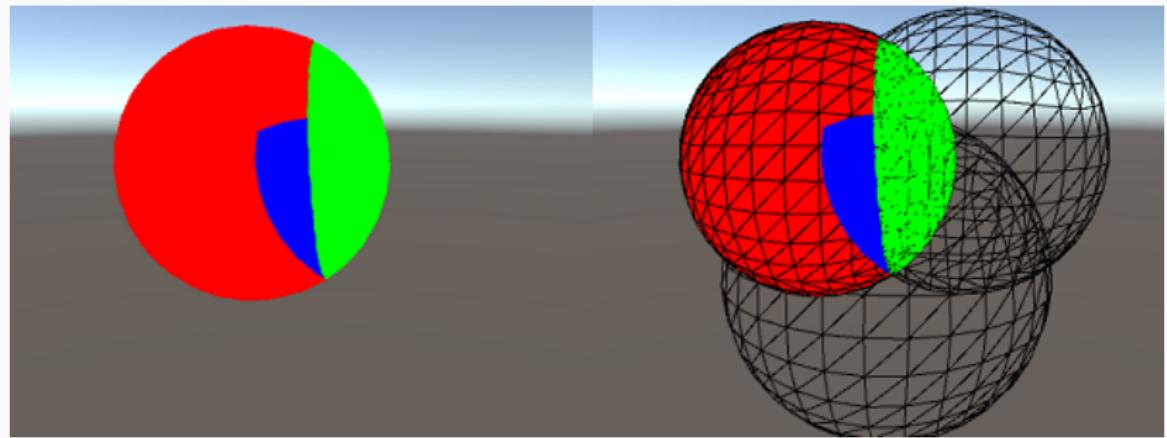
```
00000000000000000000  
00000000000000000000  
00000000000000000000  
00000000000000000000  
00000000000000000000  
00000000000000000000  
00000000000000000000  
01111111100000000000  
01111111100000000000  
01100001100000000000  
01100001100000000000  
01111111100000000000  
01111111100000000000  
00000000000000000000
```

## Stencil buffer



### After stencil test

# Stencil test



# Stencil тест

Always pass, value is unused  
`glStencilFunc(GL_ALWAYS, 0, 0xFF);`  
Unused(never fails), increment on pass, increment on z-fail  
`glStencilOp(GL_KEEP, GL_INCR, GL_INCR);`  
Draw 3 quads on top of each other

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	1	1	2	2	2	1	1	0	0
0	1	2	3	3	3	1	1	0	0
0	0	1	2	2	2	1	1	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Pass only when stencil value is >=2  
`glStencilFunc(GL_GEQUAL, 2, 0xFF);`  
Don't modify (or do?)  
`glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);`  
Draw a fullscreen quad (only bright pixels are shaded)

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	0	0	0
0	1	1	2	2	2	2	1	1	0
0	1	2	3	3	3	3	1	1	0
0	0	1	2	2	2	2	1	1	0
0	0	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0

## Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)

## Shadow volumes (stencil shadows)



## Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)

## Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)
- Любая ситуация, в которой нужно ограничить рисование определённых пикселей:

# Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)
- Любая ситуация, в которой нужно ограничить рисование определённых пикселей:
  - Симулятор самолёта: сначала рисуется внутренность самолёта, затем – окружающий мир, только там, где не был нарисован самолёт  $\Rightarrow$  можно избежать проблем с точностью буфера глубины

# Microsoft flight simulator



## Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)
- Любая ситуация, в которой нужно ограничить рисование определённых пикселей:
  - Симулятор самолёта: сначала рисуется внутренность самолёта, затем – окружающий мир, только там, где не был нарисован самолёт  $\Rightarrow$  можно избежать проблем с точностью буфера глубины

## Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)
- Любая ситуация, в которой нужно ограничить рисование определённых пикселей:
  - Симулятор самолёта: сначала рисуется внутренность самолёта, затем – окружающий мир, только там, где не был нарисован самолёт  $\Rightarrow$  можно избежать проблем с точностью буфера глубины
  - UI, который нужно нарисовать в какой-то ограниченной области экрана (например, scroll)

# ScrollView

A Scroll Rect is usually used to scroll a large image or panel of



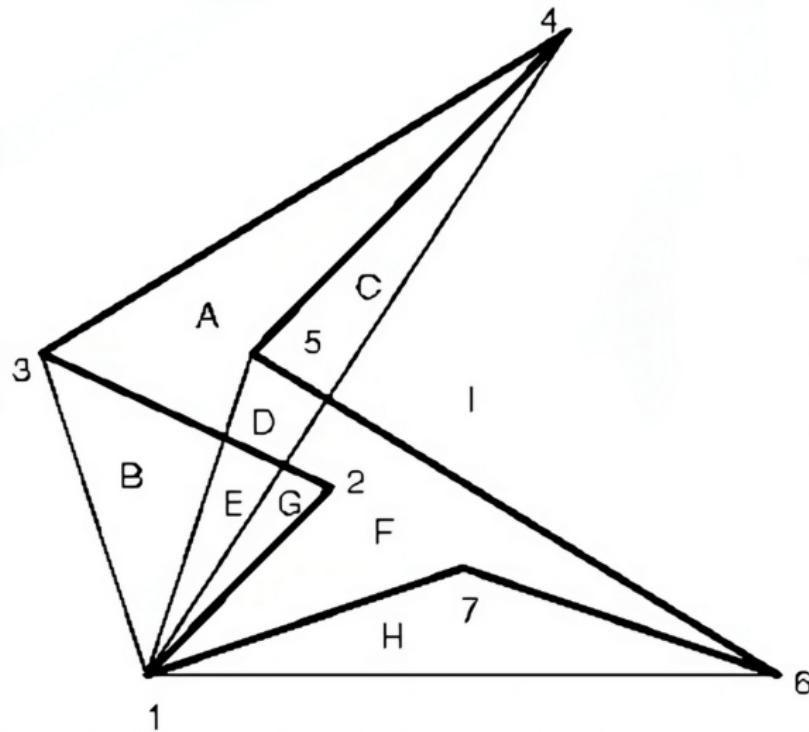
## Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)
- Любая ситуация, в которой нужно ограничить рисование определённых пикселей:
  - Симулятор самолёта: сначала рисуется внутренность самолёта, затем – окружающий мир, только там, где не был нарисован самолёт  $\Rightarrow$  можно избежать проблем с точностью буфера глубины
  - UI, который нужно нарисовать в какой-то ограниченной области экрана (например, scroll)

# Stencil буфер: применение

- Некоторые алгоритмы рисования теней (shadow volumes)
- Любая ситуация, в которой нужно ограничить рисование определённых пикселей:
  - Симулятор самолёта: сначала рисуется внутренность самолёта, затем – окружающий мир, только там, где не был нарисован самолёт  $\Rightarrow$  можно избежать проблем с точностью буфера глубины
  - UI, который нужно нарисовать в какой-то ограниченной области экрана (например, scroll)
  - Рисование невыпуклых полигонов (odd-even rule)

# Невыпуклый полигон



- \*A: 134
- B: 123 134
- C: 134 145
- \*D: 134 145 156
- E: 123 134 145 156
- \*F: 156
- G: 123 156
- H: 156 167
- I: (none)

## Stencil буфер: ссылки

- [khronos.org/opengl/wiki/Stencil\\_Test](http://khronos.org/opengl/wiki/Stencil_Test)
- [learnopengl.com/Advanced-OpenGL/Stencil-testing](http://learnopengl.com/Advanced-OpenGL/Stencil-testing)
- [open.gl/depthstencils](http://open.gl/depthstencils)
- [en.wikibooks.org/wiki/OpenGL\\_Programming/Stencil\\_buffer](http://en.wikibooks.org/wiki/OpenGL_Programming/Stencil_buffer)

## Framebuffer (FBO, кадровый буфер)

- OpenGL-объект, хранящий настройки того, куда осуществляется рисование

## Framebuffer (FBO, кадровый буфер)

- OpenGL-объект, хранящий настройки того, куда осуществляется рисование
- Не владеет памятью, только ссылается на неё

## Framebuffer (FBO, кадровый буфер)

- OpenGL-объект, хранящий настройки того, куда осуществляется рисование
- **Не владеет** памятью, только ссылается на неё
- Может иметь свой depth buffer, stencil buffer, и несколько color buffer'ов

## Framebuffer (FBO, кадровый буфер)

- OpenGL-объект, хранящий настройки того, куда осуществляется рисование
- **Не владеет** памятью, только ссылается на неё
- Может иметь свой depth buffer, stencil buffer, и несколько color buffer'ов
- Общепринятая аббревиатура: FBO

# Framebuffer

- Создание/удаление: `glGenFramebuffers`/`glDeleteFramebuffers`

# Framebuffer

- Создание/удаление: `glGenFramebuffers`/`glDeleteFramebuffers`
- Сделать текущим: `glBindFramebuffer`

# Framebuffer

- Создание/удаление: `glGenFramebuffers`/`glDeleteFramebuffers`
- Сделать текущим: `glBindFramebuffer`
- Два значения target:
  - `GL_DRAW_FRAMEBUFFER` – в этот фреймбуфер рисуют все операции рисования (`glDrawArrays`, etc), этот фреймбуфер очищает `glClear`

# Framebuffer

- Создание/удаление: `glGenFramebuffers`/`glDeleteFramebuffers`
- Сделать текущим: `glBindFramebuffer`
- Два значения target:
  - `GL_DRAW_FRAMEBUFFER` – в этот фреймбуфер рисуют все операции рисования (`glDrawArrays`, etc), этот фреймбуфер очищает `glClear`
  - `GL_READ_FRAMEBUFFER` – из этого фреймбуфера читают операции чтения

# Framebuffer

- Создание/удаление: `glGenFramebuffers`/`glDeleteFramebuffers`
- Сделать текущим: `glBindFramebuffer`
- Два значения target:
  - `GL_DRAW_FRAMEBUFFER` – в этот фреймбуфер рисуют все операции рисования (`glDrawArrays`, etc), этот фреймбуфер очищает `glClear`
  - `GL_READ_FRAMEBUFFER` – из этого фреймбуфера читают операции чтения
- Значение target `GL_FRAMEBUFFER` иногда ведёт себя как `GL_DRAW_FRAMEBUFFER`, а иногда – как `DRAW+READ`
- Советую *не использовать* `GL_FRAMEBUFFER`

## Default framebuffer

- Default framebuffer – особый объект, имеет ID = 0

## Default framebuffer

- Default framebuffer – особый объект, имеет ID = 0
- Создаётся при создании OpenGL-контекста

## Default framebuffer

- Default framebuffer – особый объект, имеет ID = 0
- Создаётся при создании OpenGL-контекста
- Настроен, чтобы рисовать на экран, к которому привязан контекст

## Default framebuffer

- Default framebuffer – особый объект, имеет ID = 0
- Создаётся при создании OpenGL-контекста
- Настроен, чтобы рисовать на экран, к которому привязан контекст
- Для него форматы цвета, буфера глубины и stencil буфера настраиваются при создании контекста

## Default framebuffer

- Default framebuffer – особый объект, имеет ID = 0
- Создаётся при создании OpenGL-контекста
- Настроен, чтобы рисовать на экран, к которому привязан контекст
- Для него форматы цвета, буфера глубины и stencil буфера настраиваются при создании контекста
- Нельзя настроить по-другому или удалить

## Запись во фреймбуфер

- Напрямую писать во фреймбуфер нельзя!

## Запись во фреймбуфер

- Напрямую писать во фреймбуфер нельзя!
- Можно скопировать пиксели из одного фреймбуфера в другой

## Запись во фреймбуфер

- Напрямую писать во фреймбуфер нельзя!
- Можно скопировать пиксели из одного фреймбуфера в другой
- Можно рисовать во фреймбуфер

# Запись во фреймбуфер

- Напрямую писать во фреймбуфер нельзя!
- Можно скопировать пиксели из одного фреймбуфера в другой
- Можно рисовать во фреймбуфер
- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`

## Запись во фреймбуфер

- Напрямую писать во фреймбуфер нельзя!
- Можно скопировать пиксели из одного фреймбуфера в другой
- Можно рисовать во фреймбуфер
- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`
- Приём рисования в текстуру называется *render to texture*

# Чтение из фреймбуфера

- `glReadPixels` – копирует на CPU пиксели из определённого участка текущего `GL_READ_FRAMEBUFFER` (можно прочитать цвет, глубину или stencil)

# Чтение из фреймбуфера

- `glReadPixels` – копирует на CPU пиксели из определённого участка текущего `GL_READ_FRAMEBUFFER` (можно прочитать цвет, глубину или stencil)
- `glBlitFramebuffer` – копирует пиксели из определённого участка текущего `GL_READ_FRAMEBUFFER` в определённый участок текущего `GL_DRAW_FRAMEBUFFER` (можно скопировать цвет, глубину или stencil)

# Чтение из фреймбуфера

- `glReadPixels` – копирует на CPU пиксели из определённого участка текущего `GL_READ_FRAMEBUFFER` (можно прочитать цвет, глубину или stencil)
- `glBlitFramebuffer` – копирует пиксели из определённого участка текущего `GL_READ_FRAMEBUFFER` в определённый участок текущего `GL_DRAW_FRAMEBUFFER` (можно скопировать цвет, глубину или stencil)
- `glReadBuffer` – настроить, из какого *attachment*'а текущего `GL_READ_FRAMEBUFFER` мы хотим читать

## Framebuffer attachments

- Текстура, на которую ссылается фреймбуфер, называется *attachment*

## Framebuffer attachments

- Текстура, на которую ссылается фреймбуфер, называется *attachment*
- У каждого фреймбуфера есть набор *attachment points*

## Framebuffer attachments

- Текстура, на которую ссылается фреймбуфер, называется *attachment*
- У каждого фреймбуфера есть набор *attachment points*
- `GL_COLOR_ATTACHMENT0`, ... `GL_COLOR_ATTACHMENT7` – цветовые буферы

## Framebuffer attachments

- Текстура, на которую ссылается фреймбуфер, называется *attachment*
- У каждого фреймбуфера есть набор *attachment points*
- `GL_COLOR_ATTACHMENT0`, ... `GL_COLOR_ATTACHMENT7` – цветовые буферы
  - Максимальное количество: `glGet(GL_MAX_COLOR_ATTACHMENTS)`

# Framebuffer attachments

- Текстура, на которую ссылается фреймбуфер, называется *attachment*
- У каждого фреймбуфера есть набор *attachment points*
- `GL_COLOR_ATTACHMENT0`, ... `GL_COLOR_ATTACHMENT7` – цветовые буферы
  - Максимальное количество: `glGet(GL_MAX_COLOR_ATTACHMENTS)`
- `GL_DEPTH_ATTACHMENT` – буфер глубины
- `GL_STENCIL_ATTACHMENT` – stencil буфер
- `GL_DEPTH_STENCIL_ATTACHMENT` – буфер глубины + stencil буфер в одной текстуре

## Framebuffer attachments

- Связать текстуру с фреймбуфером:

```
glFramebufferTexture(target, attachment, texture, level)
```

# Framebuffer attachments

- Связать текстуру с фреймбуфером:

```
glFramebufferTexture(target, attachment, texture, level)
```

- target – таргет фреймбуфера (`GL_READ_FRAMEBUFFER` или `GL_DRAW_FRAMEBUFFER`)

# Framebuffer attachments

- Связать текстуру с фреймбуфером:

```
glFramebufferTexture(target, attachment, texture, level)
```

- target – таргет фреймбуфера (`GL_READ_FRAMEBUFFER` или `GL_DRAW_FRAMEBUFFER`)
- attachment – `GL_COLOR_ATTACHMENT0`, ...

# Framebuffer attachments

- Связать текстуру с фреймбуфером:

```
glFramebufferTexture(target, attachment, texture, level)
```

- target – таргет фреймбуфера (`GL_READ_FRAMEBUFFER` или `GL_DRAW_FRAMEBUFFER`)
- attachment – `GL_COLOR_ATTACHMENT0`, ...
- texture – ID текстуры (делать саму текстуру текущей *не нужно!*)

# Framebuffer attachments

- Связать текстуру с фреймбуфером:

```
glFramebufferTexture(target, attachment, texture, level)
```

- target – таргет фреймбуфера (`GL_READ_FRAMEBUFFER` или `GL_DRAW_FRAMEBUFFER`)
- attachment – `GL_COLOR_ATTACHMENT0`, ...
- texture – ID текстуры (делать саму текстуру текущей *не нужно!*)
- level – mipmap-уровень текстуры, в который будет осуществляться рисование (обычно 0)

## Framebuffer attachments

- Можно привязать несколько текстур к одному фреймбуферу (в т.ч. иметь несколько цветовых буферов – соответственно, несколько `out`-переменных во фрагментном шейдере)

## Framebuffer attachments

- Можно привязать несколько текстур к одному фреймбуферу (в т.ч. иметь несколько цветовых буферов – соответственно, несколько `out`-переменных во фрагментном шейдере)
- Можно привязать одну текстуру к нескольким фреймбуферам

## Framebuffer attachments

- Можно привязать несколько текстур к одному фреймбуферу (в т.ч. иметь несколько цветовых буферов – соответственно, несколько `out`-переменных во фрагментном шейдере)
- Можно привязать одну текстуру к нескольким фреймбуферам
- Можно привязать несколько разных тіртар-уровней одной текстуры к одному или нескольким фреймбуферам

## Framebuffer attachments

- Можно привязать несколько текстур к одному фреймбуферу (в т.ч. иметь несколько цветовых буферов – соответственно, несколько `out`-переменных во фрагментном шейдере)
- Можно привязать одну текстуру к нескольким фреймбуферам
- Можно привязать несколько разных тіртар-уровней одной текстуры к одному или нескольким фреймбуферам
- Можно привязать одномерные и двумерные текстуры, грани cubemap-текстуры (`glFramebufferTexture2D`), слои трёхмерных или 2D-array текстур (`glFramebufferTexture3D` или `glFramebufferTextureLayer`)

## Framebuffer completeness

- У фреймбуферов есть особое свойство – *completeness*: правильно ли настроен фреймбуфер

## Framebuffer completeness

- У фреймбуферов есть особое свойство – *completeness*: правильно ли настроен фреймбуфер
- В частности, чтобы фреймбуфер был *complete*, нужно, чтобы все attachment'ы имели *правильный формат*

# Framebuffer completeness

- У фреймбуферов есть особое свойство – *completeness*: правильно ли настроен фреймбуфер
- В частности, чтобы фреймбуфер был *complete*, нужно, чтобы все attachment'ы имели *правильный формат*
- Проверить completeness – `glCheckFramebufferStatus`: вернёт `GL_FRAMEBUFFER_COMPLETE` или некий код ошибки

# Framebuffer completeness

- У фреймбуферов есть особое свойство – *completeness*: правильно ли настроен фреймбуфер
- В частности, чтобы фреймбуфер был *complete*, нужно, чтобы все attachment'ы имели *правильный формат*
- Проверить completeness – `glCheckFramebufferStatus`: вернёт `GL_FRAMEBUFFER_COMPLETE` или некий код ошибки
- Рисовать в incomplete фреймбуфер – ошибка

## Renderable formats

- Что такое *правильный формат?*

## Renderable formats

- Что такое *правильный формат?*
- Для цветового буфера – *color-renderable* формат, т.е. любой цветовой формат: GL\_RED, GL\_RGB8, GL\_RGBA8, GL\_RG32F, ...

## Renderable formats

- Что такое *правильный формат?*
- Для цветового буфера – *color-renderable* формат, т.е. любой цветовой формат: GL\_RED, GL\_RGB8, GL\_RGBA8, GL\_RG32F, ...
- Для буфера глубины – *depth-renderable* формат:  
GL\_DEPTH\_COMPONENT16, GL\_DEPTH\_COMPONENT24,  
GL\_DEPTH\_COMPONENT32F, GL\_DEPTH24\_STENCIL8,  
GL\_DEPTH32F\_STENCIL8

## Renderable formats

- Что такое *правильный формат*?
- Для цветового буфера – *color-renderable* формат, т.е. любой цветовой формат: GL\_RED, GL\_RGB8, GL\_RGBA8, GL\_RG32F, ...
- Для буфера глубины – *depth-renderable* формат:  
GL\_DEPTH\_COMPONENT16, GL\_DEPTH\_COMPONENT24,  
GL\_DEPTH\_COMPONENT32F, GL\_DEPTH24\_STENCIL8,  
GL\_DEPTH32F\_STENCIL8
- Для stencil буфера – *stencil-renderable* формат:  
GL\_DEPTH24\_STENCIL8, GL\_DEPTH32F\_STENCIL8

## Renderable formats

- Что такое *правильный формат*?
- Для цветового буфера – *color-renderable* формат, т.е. любой цветовой формат: GL\_RED, GL\_RGB8, GL\_RGBA8, GL\_RG32F, ...
- Для буфера глубины – *depth-renderable* формат:  
GL\_DEPTH\_COMPONENT16, GL\_DEPTH\_COMPONENT24,  
GL\_DEPTH\_COMPONENT32F, GL\_DEPTH24\_STENCIL8,  
GL\_DEPTH32F\_STENCIL8
- Для stencil буфера – *stencil-renderable* формат:  
GL\_DEPTH24\_STENCIL8, GL\_DEPTH32F\_STENCIL8
- Некоторые форматы могут *не поддерживаться* конкретными драйверами/GPU

## Renderable formats

- Что такое *правильный формат*?
- Для цветового буфера – *color-renderable* формат, т.е. любой цветовой формат: GL\_RED, GL\_RGB8, GL\_RGBA8, GL\_RG32F, ...
- Для буфера глубины – *depth-renderable* формат:  
GL\_DEPTH\_COMPONENT16, GL\_DEPTH\_COMPONENT24,  
GL\_DEPTH\_COMPONENT32F, GL\_DEPTH24\_STENCIL8,  
GL\_DEPTH32F\_STENCIL8
- Для stencil буфера – *stencil-renderable* формат:  
GL\_DEPTH24\_STENCIL8, GL\_DEPTH32F\_STENCIL8
- Некоторые форматы могут *не поддерживаться* конкретными драйверами/GPU
- Есть небольшой список гарантированно поддерживаемых форматов (см. документацию)

## Рисование во фреймбуфер

- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`

# Рисование во фреймбуфер

- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`
- Как связать attachments фреймбуфера с `out`-переменными фрагментного шейдера?

# Рисование во фреймбуфер

- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`
- Как связать attachments фреймбуфера с `out`-переменными фрагментного шейдера?
- `glDrawBuffers(n, buffers)`
  - `buffers` – массив констант `GL_COLOR_ATTACHMENTi` или специальных значений для default фреймбуфера
  - `layout (location = k) out ...` будет писать в attachment, указанный в `buffers[k]`

# Рисование во фреймбуфер

- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`
- Как связать attachments фреймбуфера с `out`-переменными фрагментного шейдера?
- `glDrawBuffers(n, buffers)`
  - `buffers` – массив констант `GL_COLOR_ATTACHMENTi` или специальных значений для default фреймбуфера
  - `layout (location = k) out ...` будет писать в attachment, указанный в `buffers[k]`
- Эта настройка запоминается в самом фреймбуфере, её лучше делать один раз при создании фреймбуфера

# Рисование во фреймбуфер

- Все операции рисования `glDraw*` рисуют в текущий `GL_DRAW_FRAMEBUFFER`
- Как связать attachments фреймбуфера с `out`-переменными фрагментного шейдера?
- `glDrawBuffers(n, buffers)`
  - `buffers` – массив констант `GL_COLOR_ATTACHMENTi` или специальных значений для default фреймбуфера
  - `layout (location = k) out ...` будет писать в attachment, указанный в `buffers[k]`
- Эта настройка запоминается в самом фреймбуфере, её лучше делать один раз при создании фреймбуфера
- Есть устаревшая функция `glDrawBuffer` – ей лучше не пользоваться

# Viewport

- `glViewport` глобально настраивает преобразование из координат  $[-1, 1]^2$  в пиксельные координаты

# Viewport

- `glViewport` глобально настраивает преобразование из координат  $[-1, 1]^2$  в пиксельные координаты
- Viewport никак не связан с фреймбуфером

# Viewport

- `glViewport` глобально настраивает преобразование из координат  $[-1, 1]^2$  в пиксельные координаты
- Viewport никак не связан с фреймбуфером
- Но обычно мы хотим, чтобы viewport совпадал с размером текущего фреймбуфера (он может не совпадать с размером экрана)

# Viewport

- `glViewport` глобально настраивает преобразование из координат  $[-1, 1]^2$  в пиксельные координаты
- Viewport никак не связан с фреймбуфером
- Но обычно мы хотим, чтобы viewport совпадал с размером текущего фреймбуфера (он может не совпадать с размером экрана)
- $\Rightarrow$  После каждого изменения текущего фреймбуфера полезно подумать о viewport'e

# Viewport

- *glViewport* глобально настраивает преобразование из координат  $[-1, 1]^2$  в пиксельные координаты
- Viewport никак не связан с фреймбуфером
- Но обычно мы хотим, чтобы viewport совпадал с размером текущего фреймбуфера (он может не совпадать с размером экрана)
- $\Rightarrow$  После каждого изменения текущего фреймбуфера полезно подумать о viewport'e
- N.B. *glClear* игнорирует viewport и очищает весь фреймбуфер

## Рисование в текстуру: инициализация

```
GLuint fbo;
 glGenFramebuffers(1, &fbo);

// создаём текстуру
...

// привязываем текстуру
 glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo);
 glFramebufferTexture(GL_DRAW_FRAMEBUFFER,
    GL_COLOR_ATTACHMENT0,
    fbo_color_texture, 0);
 GLenum draw_buffers[] = {GL_COLOR_ATTACHMENT0};
 glBindDrawBuffers(1, draw_buffers);

// проверяем completeness
 if (glCheckFramebufferStatus(GL_DRAW_FRAMEBUFFER)
     != GL_FRAMEBUFFER_COMPLETE)
     throw std::runtime_error("Framebuffer incomplete");
```

# Рисование в текстуру: рендеринг

```
// Рисование в FBO
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo);
glClear(GL_COLOR_BUFFER_BIT);
glViewport(0, 0, fbo_width, fbo_height);
drawSomething();

...
// Рисование в дефолтный фреймбуфер
// здесь можно использовать текстуру fbo_color_texture
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
glClear(GL_COLOR_BUFFER_BIT);
glViewport(0, 0, screen_width, screen_height);
drawSomethingElseUsing(fbo_color_texture);
```

# Renderbuffers

- Объекты OpenGL, хранящие пиксели

# Renderbuffers

- Объекты OpenGL, хранящие пиксели
- Нельзя загрузить данные, нет режимов фильтрации, нет mipmaps  $\Rightarrow$  могут быть быстрее текстур, и работать с ними проще

# Renderbuffers

- Объекты OpenGL, хранящие пиксели
- Нельзя загрузить данные, нет режимов фильтрации, нет mipmaps  $\Rightarrow$  могут быть быстрее текстур, и работать с ними проще
- Всегда двумерные (как GL\_TEXTURE\_2D)

# Renderbuffers

- Объекты OpenGL, хранящие пиксели
- Нельзя загрузить данные, нет режимов фильтрации, нет mipmap'ов  $\Rightarrow$  могут быть быстрее текстур, и работать с ними проще
- Всегда двумерные (как GL\_TEXTURE\_2D)
- Могут быть attachment'ами для фреймбуфера, вместо текстур

# Renderbuffers

- Объекты OpenGL, хранящие пиксели
- Нельзя загрузить данные, нет режимов фильтрации, нет mipmaps  $\Rightarrow$  могут быть быстрее текстур, и работать с ними проще
- Всегда двумерные (как GL\_TEXTURE\_2D)
- Могут быть attachment'ами для фреймбуфера, вместо текстур
- Удобно использовать напр. для буфера глубины

# Renderbuffers

- Создать/удалить: `glGenRenderbuffers`/`glDeleteRenderbuffers`

# Renderbuffers

- Создать/удалить: `glGenRenderbuffers`/`glDeleteRenderbuffers`
- Сделать текущим: `glBindRenderbuffer`, `target = GL_RENDERBUFFER`

# Renderbuffers

- Создать/удалить: `glGenRenderbuffers`/`glDeleteRenderbuffers`
- Сделать текущим: `glBindRenderbuffer`, `target = GL_RENDERBUFFER`
- Выделить память: `glRenderbufferStorage`

# Renderbuffers

- Создать/удалить: `glGenRenderbuffers`/`glDeleteRenderbuffers`
- Сделать текущим: `glBindRenderbuffer`, `target = GL_RENDERBUFFER`
- Выделить память: `glRenderbufferStorage`
- Привязать renderbuffer к фреймбуферу:  
`glFramebufferRenderbuffer`

## Framebuffers & renderbuffers: ссылки

- [www.khronos.org/opengl/wiki/Framebuffer\\_Object](http://www.khronos.org/opengl/wiki/Framebuffer_Object)
- [www.khronos.org/opengl/wiki/Renderbuffer\\_Object](http://www.khronos.org/opengl/wiki/Renderbuffer_Object)
- [opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture](http://opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture)
- [learnopengl.com/Advanced-OpenGL/Framebuffers](http://learnopengl.com/Advanced-OpenGL/Framebuffers)

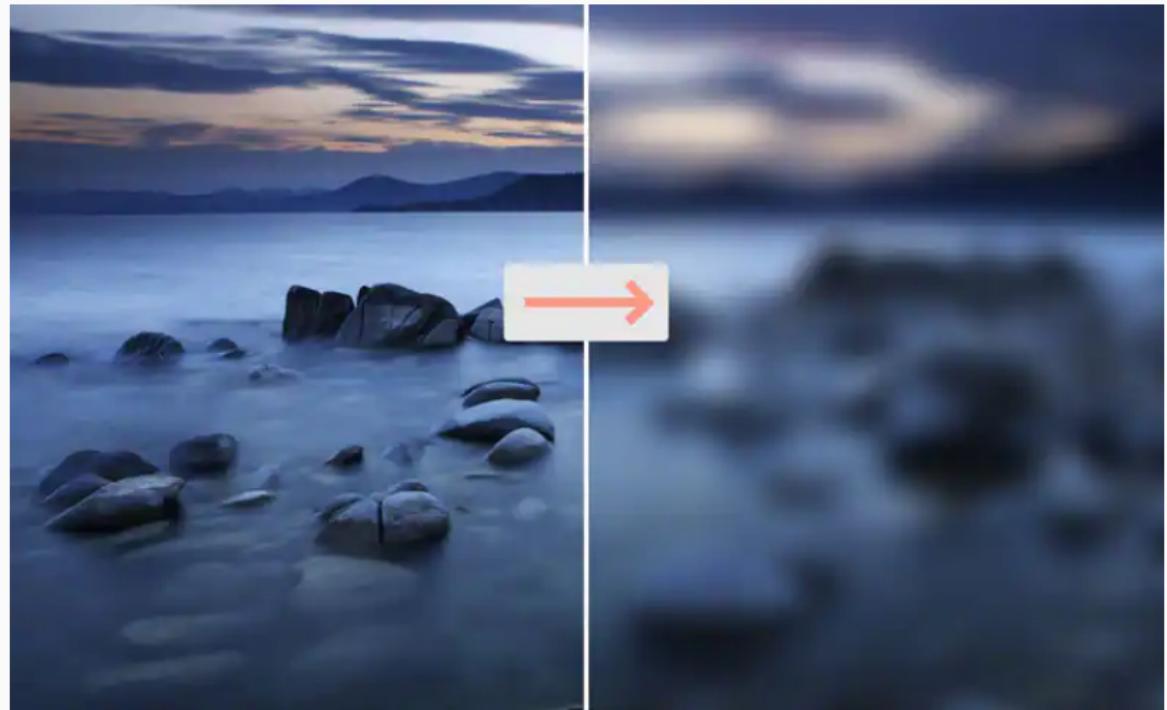
## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие

## Размытие



## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)

# Свечение



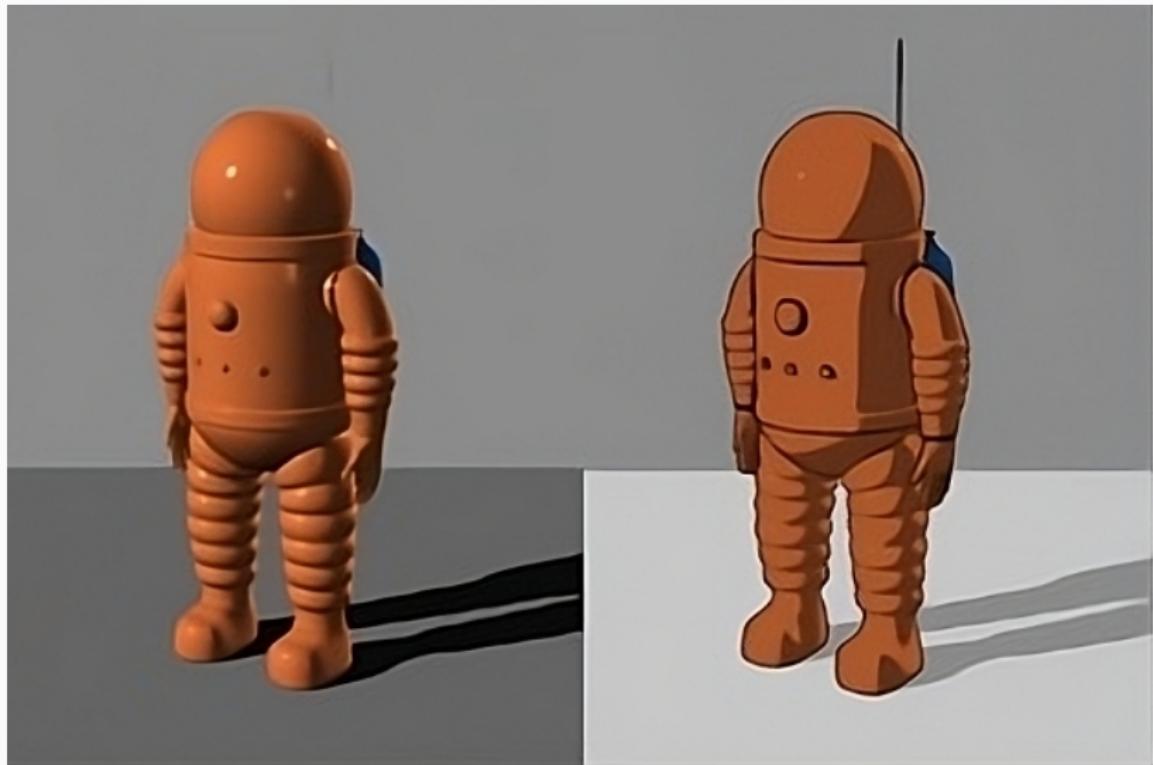
## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)

# Toon shading



## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)

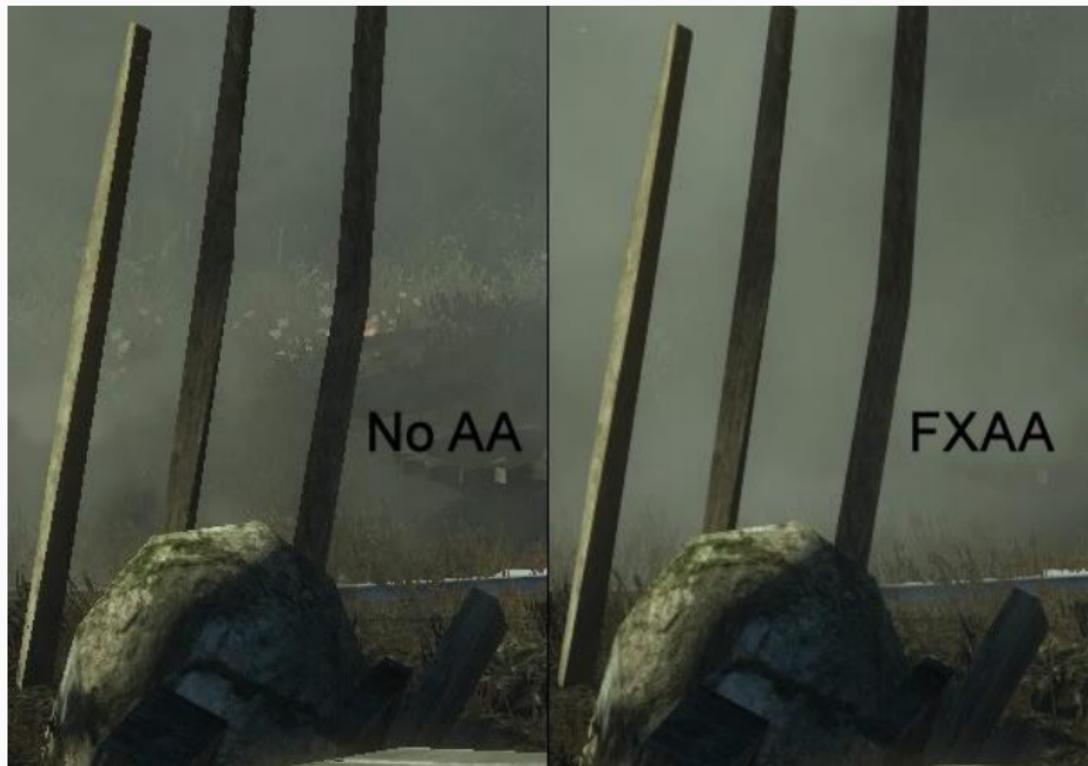
## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)

FXAA



## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)
  - И т.д.

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)
  - И т.д.
- Тени (shadow maps)

# Shadow mapping



## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)
  - И т.д.
- Тени (shadow maps)

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)
  - И т.д.
- Тени (shadow maps)
- Отражения

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)
  - И т.д.
- Тени (shadow maps)
- Отражения
- Deferred shading

## Примеры использования render to texture

- Пост-обработка кадра – реализуется рисованием кадра в текстуру, и затем рисованием полноэкранного прямоугольника на экран с нужным шейдером, читающим эту текстуру
  - Размытие
  - Свечение (bloom)
  - Toon shading (edge detection, color grading)
  - HDR, гамма-коррекция
  - Сглаживание (FXAA)
  - И т.д.
- Тени (shadow maps)
- Отражения
- Deferred shading
- И очень много другого

# Размытие

- Усреднение значений соседних пикселей в некоторой окрестности

# Размытие

- Усреднение значений соседних пикселей в некоторой окрестности
- Box blur: все веса одинаковые, получается слегка угловатым

# Размытие

- Усреднение значений соседних пикселей в некоторой окрестности
- Box blur: все веса одинаковые, получается слегка угловатым
- Gaussian blur: веса пропорциональны гауссиане  $\exp\left(-\frac{x^2+y^2}{r^2}\right)$ , получается равномерное сглаживание

# Размытие

- Усреднение значений соседних пикселей в некоторой окрестности
- Box blur: все веса одинаковые, получается слегка угловатым
- Gaussian blur: веса пропорциональны гауссиане  $\exp\left(-\frac{x^2+y^2}{r^2}\right)$ , получается равномерное сглаживание
- Несколько итераций box blur похожи на один gaussian blur (центральная предельная теорема)

# Размытие

- Усреднение значений соседних пикселей в некоторой окрестности
- Box blur: все веса одинаковые, получается слегка угловатым
- Gaussian blur: веса пропорциональны гауссиане  $\exp\left(-\frac{x^2+y^2}{r^2}\right)$ , получается равномерное сглаживание
- Несколько итераций box blur похожи на один gaussian blur (центральная предельная теорема)
- Обычно gaussian blur делают в два прохода: один размывает по горизонтали, второй – по вертикали (*separable blur*)

# Box blur

```
uniform sampler2D source;

in vec2 texcoord;

out vec4 out_color;

void main()
{
    vec4 sum = vec4(0.0);
    const int N = 5;

    for (int x = -N; x <= N; ++x) {
        for (int y = -N; y <= N; ++y) {
            vec2 offset = vec2(x,y) / vec2(textureSize(source, 0));
            sum += texture(source, texcoord + offset);
        }
    }

    out_color = sum / float((2*N+1)*(2*N+1));
}
```

# Gaussian blur

```
uniform sampler2D source;

in vec2 texcoord;

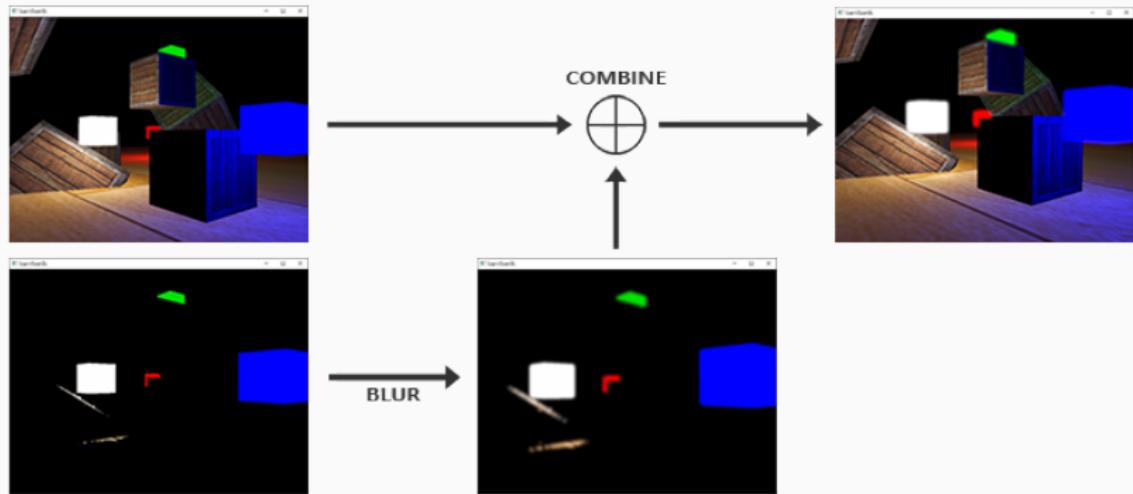
out vec4 out_color;

void main()
{
    vec4 sum = vec4(0.0);
    float sum_w = 0.0;
    const int N = 5;
    float radius = 3.0;

    for (int x = -N; x <= N; ++x) {
        for (int y = -N; y <= N; ++y) {
            vec2 offset = vec2(x,y) / vec2(textureSize(source, 0));
            float c = exp(-float(x*x + y*y) / (radius*radius));
            sum += c * texture(source, texcoord + offset);
            sum_w += c;
        }
    }

    out_color = sum / sum_w;
}
```

# Bloom



## Toon shading (cel shading)

- Edge detection + color grading

## Toon shading (cel shading)

- Edge detection + color grading
- Edge detection: шейдер находит и выделяет резкие перепады цвета или глубины (используя, например, *Sobel filter*)

## Toon shading (cel shading)

- Edge detection + color grading
- Edge detection: шейдер находит и выделяет резкие перепады цвета или глубины (используя, например, *Sobel filter*)
- Color grading: палитра цветов искусственно уменьшается (например, сжимается до 2-3 бит на канал)

## Ссылки

- [learnopengl.com/Advanced-Lighting/Bloom](http://learnopengl.com/Advanced-Lighting/Bloom)
- Gaussian blur (youtube video tutorial)