

Компьютерная графика

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

2022

Ошибки OpenGL

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

Ошибки OpenGL

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

Ошибки OpenGL

- ▶ Часто драйвер может 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 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 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.

Ошибки OpenGL

- ▶ Часто драйвер может 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 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 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 значения ошибок:

- ▶ `GL_NO_ERROR` – ошибки нет

Ошибки OpenGL

Возможные значения ошибок:

- ▶ GL_NO_ERROR – ошибки нет
- ▶ GL_INVALID_ENUM – недопустимое значение перечисления (например, GL_TEXTURE_2D как первый параметр glBindBuffer)

Ошибки OpenGL

Возможные значения ошибок:

- ▶ GL_NO_ERROR – ошибки нет
- ▶ GL_INVALID_ENUM – недопустимое значение 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 значения ошибок:

- ▶ GL_NO_ERROR – ошибки нет
- ▶ GL_INVALID_ENUM – недопустимое значение 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 значения ошибок:

- ▶ GL_NO_ERROR – ошибки нет
- ▶ GL_INVALID_ENUM – недопустимое значение 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 значения ошибок:

- ▶ `GL_NO_ERROR` – ошибки нет
- ▶ `GL_INVALID_ENUM` – недопустимое значение 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 (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
- ▶ ⇒ 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
- ▶ 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, например
 - ▶ Конкретный производитель выпустил новую функциональность

Расширения OpenGL

- ▶ Предоставляют дополнительную функциональность за рамками возможностей конкретной версии OpenGL, например
 - ▶ 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, например
 - ▶ 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:
 - ▶ 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 тип 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
- ▶ khronos.org/registry/OpenGL/index_gl.php – список всех зарегистрированных расширений

Blending (смешивание)

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

Blending (смешивание)

- ▶ 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)

- ▶ 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

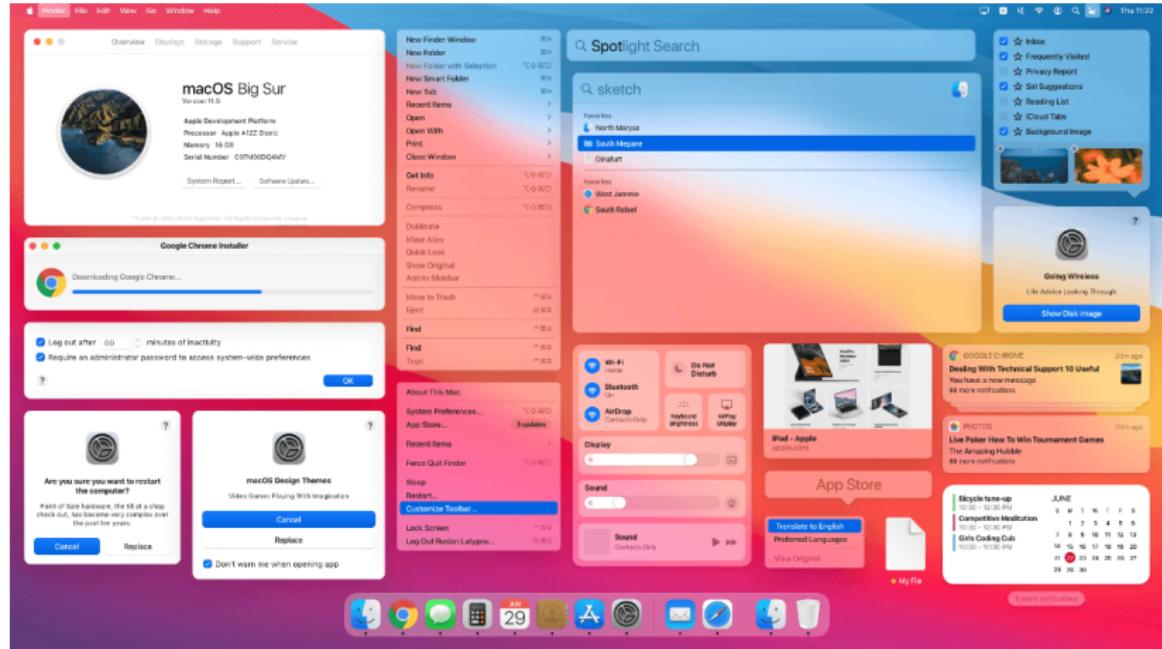


Full transparent window



Partially transparent window

Полупрозрачность



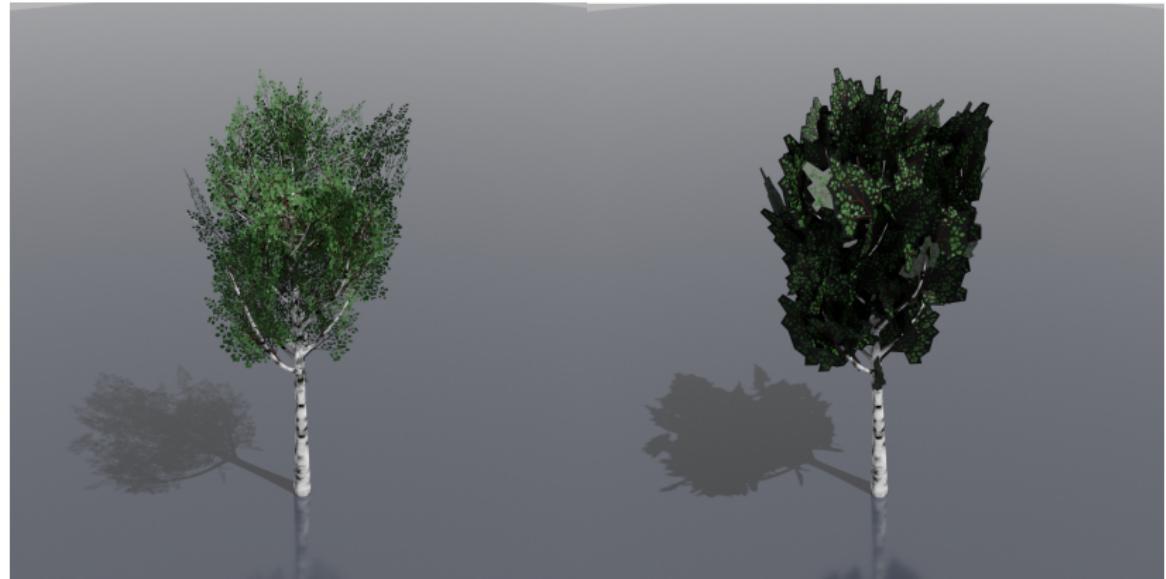
Blending (смешивание)

- ▶ 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)

- ▶ 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
 - ▶ Раствительность

Деревья



Blending (смешивание)

- ▶ 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
 - ▶ Раствительность

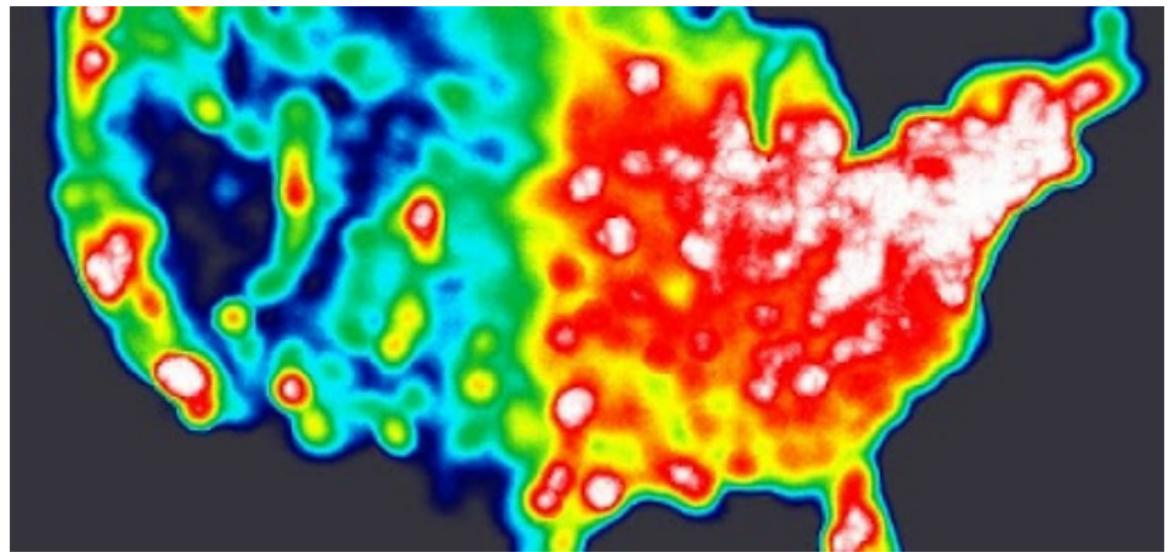
Blending (смешивание)

- ▶ 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)

- ▶ 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



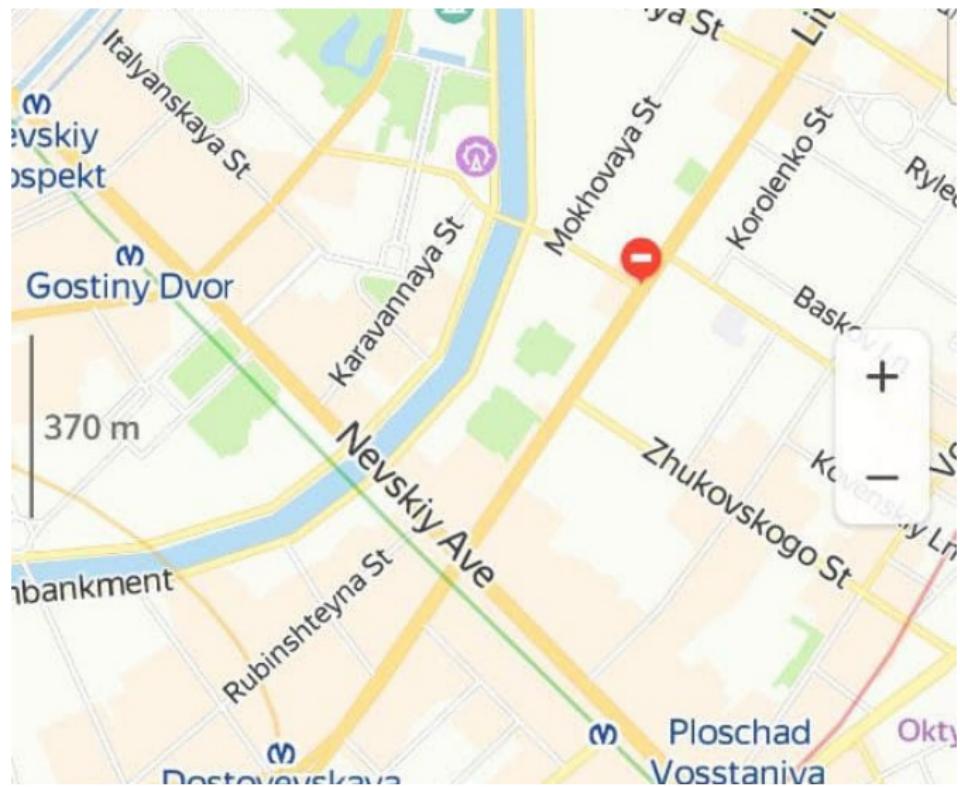
Blending (смешивание)

- ▶ 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)

- ▶ 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

Yandex maps



Blending (смешивание)

- ▶ 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)

- ▶ 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
- ▶ 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`: $f(src, dst) = src + dst$
- ▶ `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: $C = 0$
- ▶ 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 полупрозрачность A
 - ▶ $A = 0$ – полностью прозрачный цвет
 - ▶ $A = 1$ – полностью непрозрачный цвет

Blending (смешивание)

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

$$dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$$

Blending (смешивание)

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

$$dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$$

- ▶ 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 A
 - ▶ $A = 0$ – полностью прозрачный цвет
 - ▶ $A = 1$ – полностью непрозрачный цвет
- ▶ Для этого нужна такая формула смешивания:

$$dst \leftarrow src_A \cdot src + (1 - src_A) \cdot dst$$

- ▶ Этому соответствует настройка

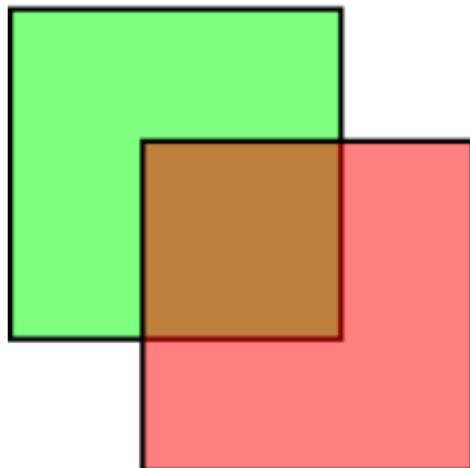
```
glBlendEquation(GL_FUNC_ADD);  
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- ▶ Это самый типичный способ использовать blending

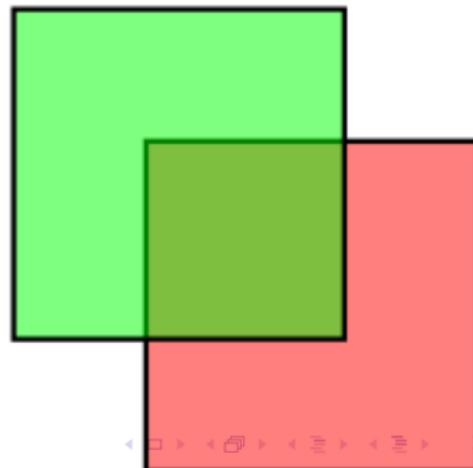
Blending (смешивание)

- ▶ 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

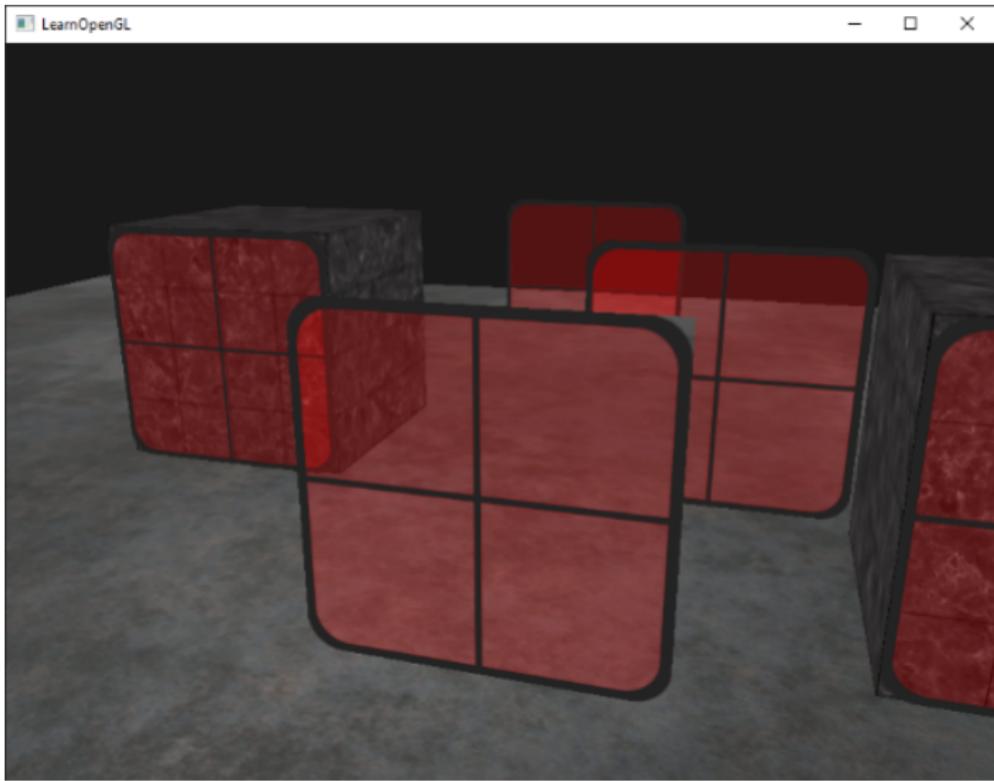


Green on top



Blending (смешивание)

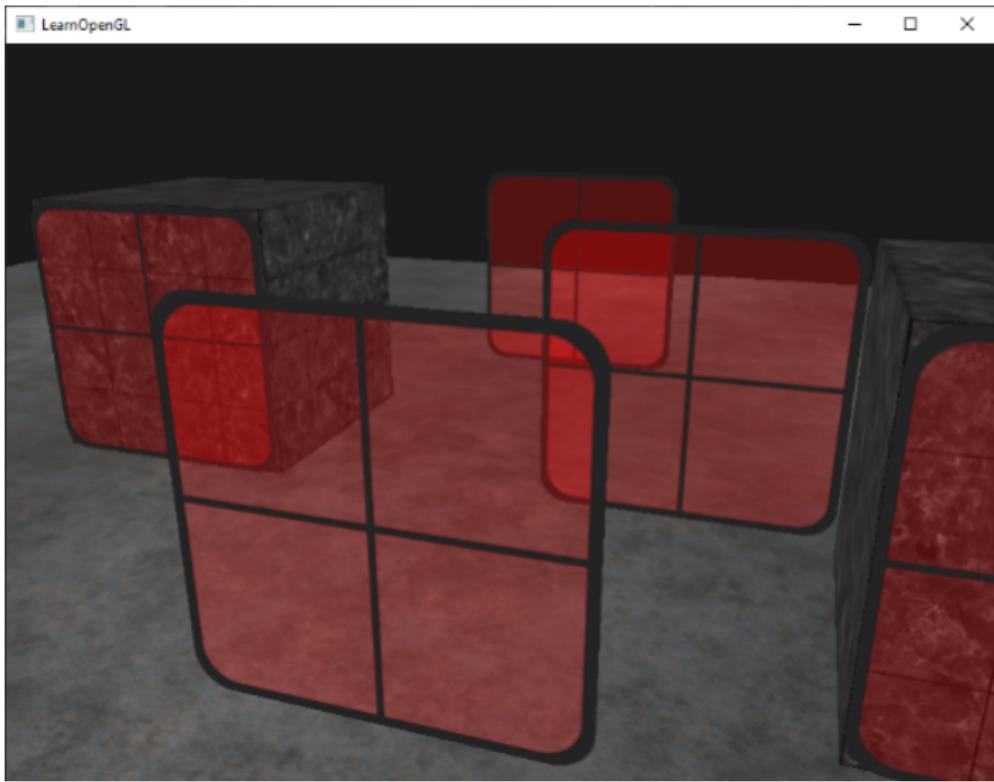
- ▶ Смешивание некорректно работает с тестом глубины



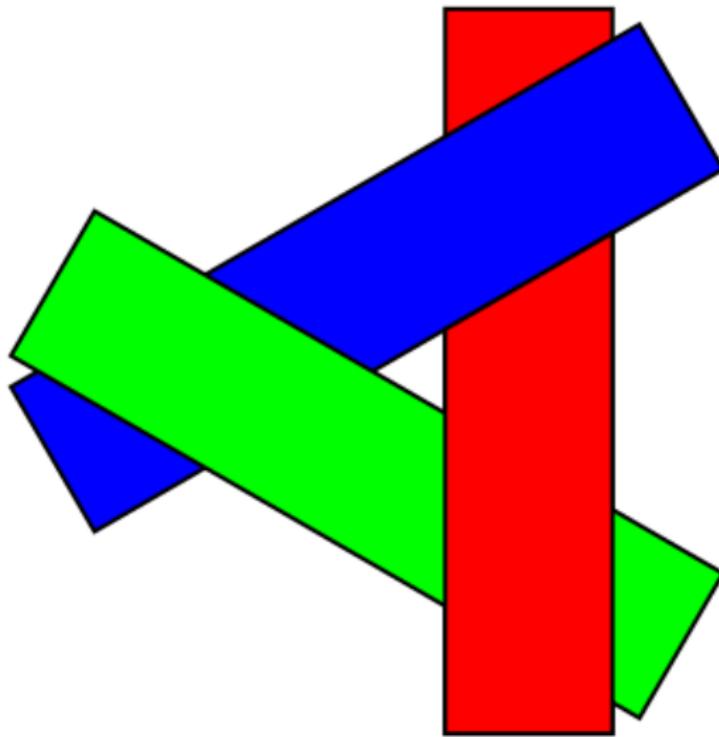
Blending (смешивание)

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



Blending (смешивание)



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)`

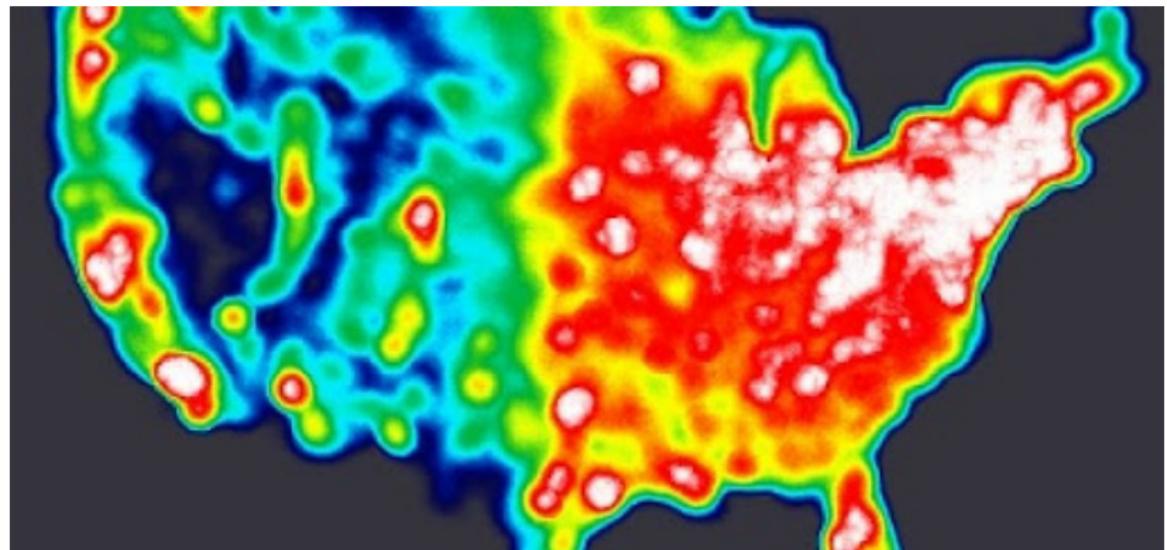
Аддитивное смешивание

- ▶ 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!

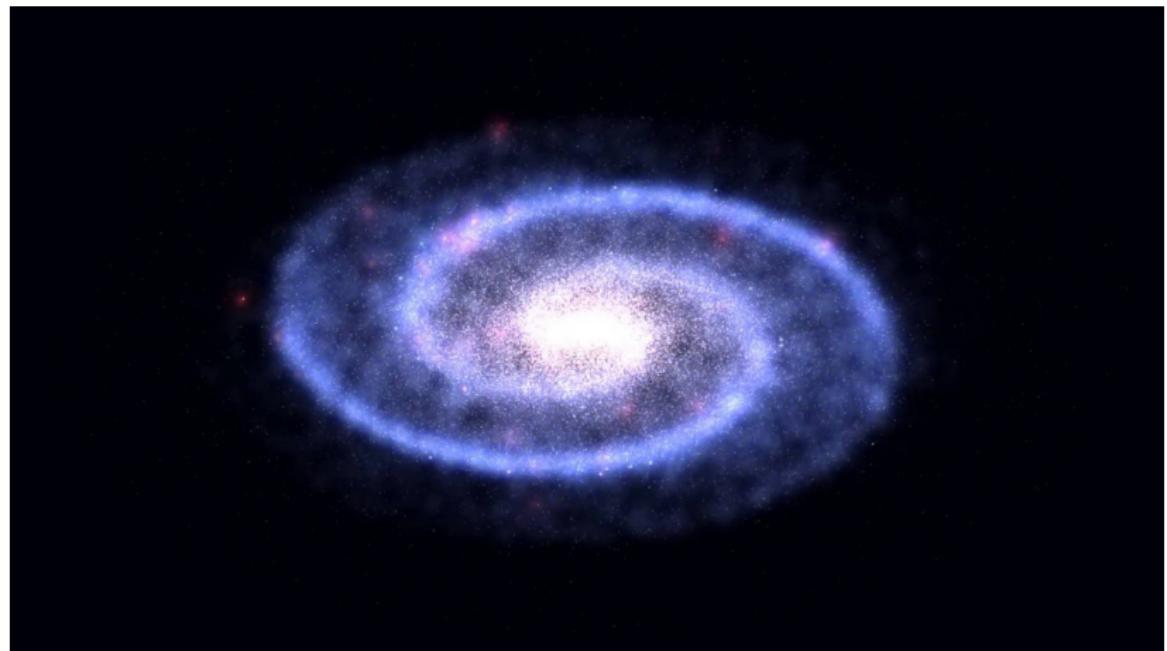
Аддитивное смешивание

- ▶ 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!
- ▶ Heatmaps, облака точек

Аддитивное смещивание



Аддитивное смешивание



Blending (смешивание)

- ▶ khronos.org/opengl/wiki/Blending
- ▶ opengl-tutorial.org/intermediate-tutorials/tutorial-10-transparency
- ▶ learnopengl.com/Advanced-OpenGL/Blending

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`, `GL_GREATER`, `GL_EQUAL`, ...
 - ▶ 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`, `GL_GREATER`, `GL_EQUAL`, ...
 - ▶ 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)`

- ▶ `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

- ▶ 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

- ▶ Возможные значения `sfail`, `dpfail` 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

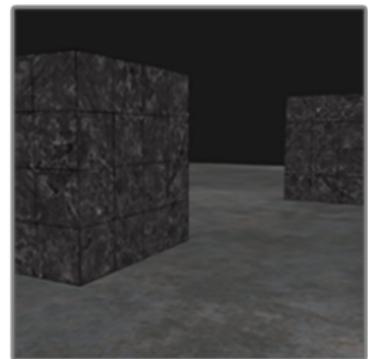
Stencil тест

- ▶ 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 тест



Color buffer

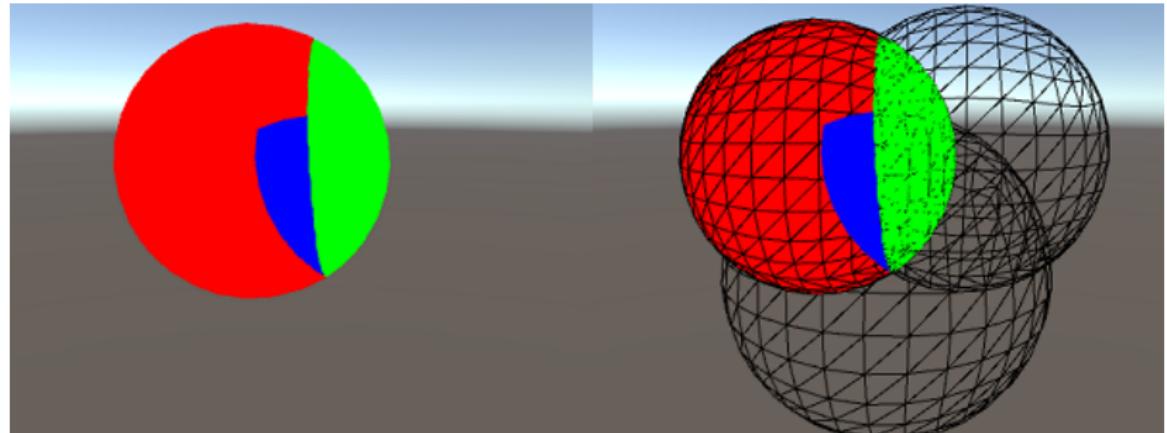
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
01111111110000000000
01111111110000000000
01100001100000000000
01100001100000000000
01111111110000000000
01111111110000000000
00000000000000000000

Stencil buffer



After stencil test

Stencil тест



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	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	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

Stencil буфер: применение

- ▶ Некоторые алгоритмы рисования теней (shadow volumes)

Shadow volumes (stencil shadows)



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 ⇒ 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 ⇒ можно избежать проблем с точностью буфера глубины
 - ▶ UI, который нужно нарисовать в какой-то ограниченной области экрана (например, scroll)

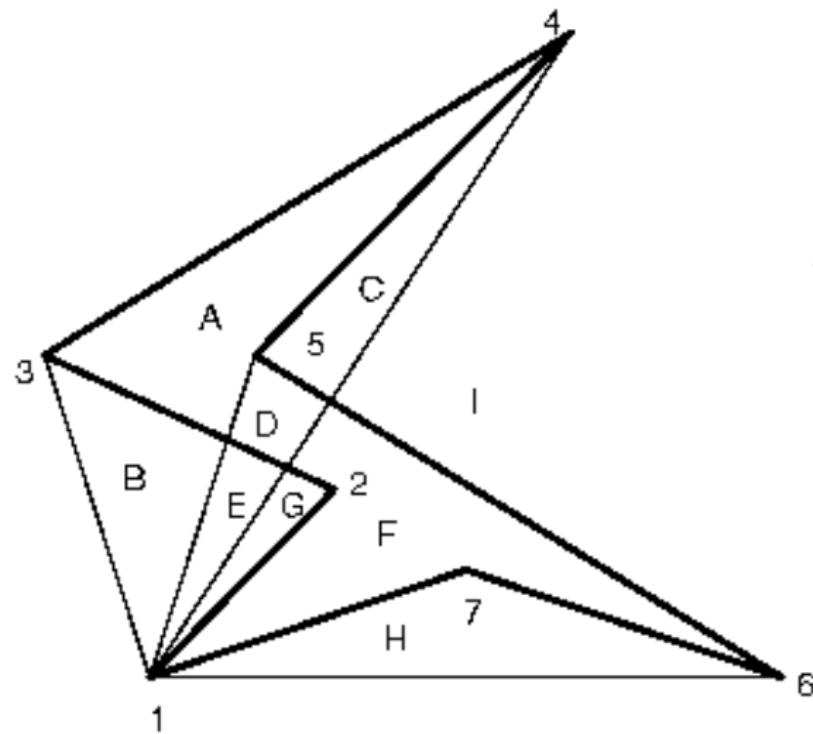
ScrollView

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)
 - ▶ 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
- ▶ learnopengl.com/Advanced-OpenGL/Stencil-testing
- ▶ open.gl/depthstencils
- ▶ 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 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 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
- ▶ Можно вызвать
`glBindFramebuffer(GL_FRAMEBUFFER, fbo)` – это эквивалентно

`glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo);`
`glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo);`
- ▶ Советую не использовать `GL_FRAMEBUFFER`

Default framebuffer

- ▶ Особый объект, имеет ID = 0

Default framebuffer

- ▶ Особый объект, имеет ID = 0
- ▶ Создаётся при создании OpenGL-контекста

Default framebuffer

- ▶ Особый объект, имеет ID = 0
- ▶ Создаётся при создании OpenGL-контекста
- ▶ Настроен, чтобы рисовать на экран, к которому привязан контекст

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

- ▶ 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 *рисовать* с его помощью в текстуру
- ▶ Все операции рисования (`glDraw*`) рисуют в текущий `GL_DRAW_FRAMEBUFFER`
- ▶ *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`)

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

Framebuffer attachments

- ▶ Связать текстуру с фреймбуфером:

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

Framebuffer attachments

- ▶ Связать текстуру с фреймбуфером:

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

- ▶ target – текстура будет привязана к текущему фреймбуферу с таким таргетом (GL_READ_FRAMEBUFFER или GL_DRAW_FRAMEBUFFER)

Framebuffer attachments

- ▶ Связать текстуру с фреймбуфером:

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

- ▶ target – текстура будет привязана к текущему фреймбуферу с таким таргетом (GL_READ_FRAMEBUFFER или GL_DRAW_FRAMEBUFFER)
- ▶ attachment – GL_COLOR_ATTACHMENT0, ...

Framebuffer attachments

- ▶ Связать текстуру с фреймбуфером:
`glFramebufferTexture(target, attachment, texture, level)`
 - ▶ target – текстура будет привязана к текущему фреймбуферу с таким таргетом (`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 – текстура будет привязана к текущему фреймбуферу с таким таргетом (`GL_READ_FRAMEBUFFER` или `GL_DRAW_FRAMEBUFFER`)
 - ▶ attachment – `GL_COLOR_ATTACHMENT0`, ...
 - ▶ texture – ID текстуры (делать саму текстуру текущей не нужно!)
 - ▶ level – міртар-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 `кубетар`-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
- ▶ Фреймбуфер считается complete, если
 - ▶ Размеры всех его attachment'ов совпадают

Framebuffer completeness

- ▶ У фреймбуферов есть особое свойство – completeness: правильно ли настроен фреймбуфер
- ▶ Фреймбуфер считается complete, если
 - ▶ Размеры всех его attachment'ов совпадают
 - ▶ Все attachment'ы имеют правильный формат

Framebuffer completeness

- ▶ У фреймбуферов есть особое свойство – completeness: правильно ли настроен фреймбуфер
- ▶ Фреймбуфер считается complete, если
 - ▶ Размеры всех его attachment'ов совпадают
 - ▶ Все attachment'ы имеют 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
- ▶ Фреймбуфер считается complete, если
 - ▶ Размеры всех его attachment'ов совпадают
 - ▶ Все attachment'ы имеют правильный формат
- ▶ Проверить completeness – `glCheckFramebufferStatus`: вернёт `GL_FRAMEBUFFER_COMPLETE` или некий код ошибки
- ▶ Рисовать в incomplete фреймбуфер нельзя!

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]`
- ▶ N.B.: для рисования в default framebuffer обычно нужен `buffers[] = {GL_FRONT_LEFT}`

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]`
- ▶ N.B.: для рисования в default framebuffer обычно нужен `buffers[] = {GL_FRONT_LEFT}`
- ▶ Есть устаревшая функция `glDrawBuffer` – ей лучше не пользоваться

Viewport

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

Viewport

- ▶ glViewport настраивает преобразование из координат $[-1, 1]^2$ в пиксельные координаты
- ▶ Никак не связан с фреймбуфером

Viewport

- ▶ glViewport настраивает преобразование из координат $[-1, 1]^2$ в пиксельные координаты
- ▶ Никак не связан с фреймбуфером
- ▶ Но обычно мы хотим, чтобы viewport совпадал с размером текущего фреймбуфера

Viewport

- ▶ glViewport настраивает преобразование из координат $[-1, 1]^2$ в пиксельные координаты
- ▶ Никак не связан с фреймбуфером
- ▶ Но обычно мы хотим, чтобы viewport совпадал с размером текущего фреймбуфера
- ▶ ⇒ Перед рисованием с помощью фреймбуфера надо подумать о viewport'е

Viewport

- ▶ `glViewport` настраивает преобразование из координат $[-1, 1]^2$ в пиксельные координаты
- ▶ Никак не связан с фреймбуфером
- ▶ Но обычно мы хотим, чтобы `viewport` совпадал с размером текущего фреймбуфера
- ▶ ⇒ Перед рисованием с помощью фреймбуфера надо подумать о `viewport`'е
- ▶ N.B. `glClear` игнорирует `viewport`

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);

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

Рисование в текстуру: рисование

```
// Рисование в FBO
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo);
GLenum draw_buffers[] = {GL_COLOR_ATTACHMENT0};
glDrawBuffers(1, draw_buffers);
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);
GLenum draw_buffers[] = {GL_FRONT_LEFT};
glDrawBuffers(1, draw_buffers);
glClear(GL_COLOR_BUFFER_BIT);
glViewport(0, 0, screen_width, screen_height);
drawSomethingElse();
```

Renderbuffers

- ▶ Объекты OpenGL, хранящие пиксели

Renderbuffers

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

Renderbuffers

- ▶ Объекты OpenGL, хранящие пиксели
- ▶ Нельзя загрузить данные, нет режимов фильтрации, нет mipmaps ⇒ могут быть быстрее текстур, и работать с ними проще
- ▶ Могут быть attachment'ами для фреймбуфера, вместо текстур

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
- ▶ www.khronos.org/opengl/wiki/Renderbuffer_Object
- ▶ opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture
- ▶ 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

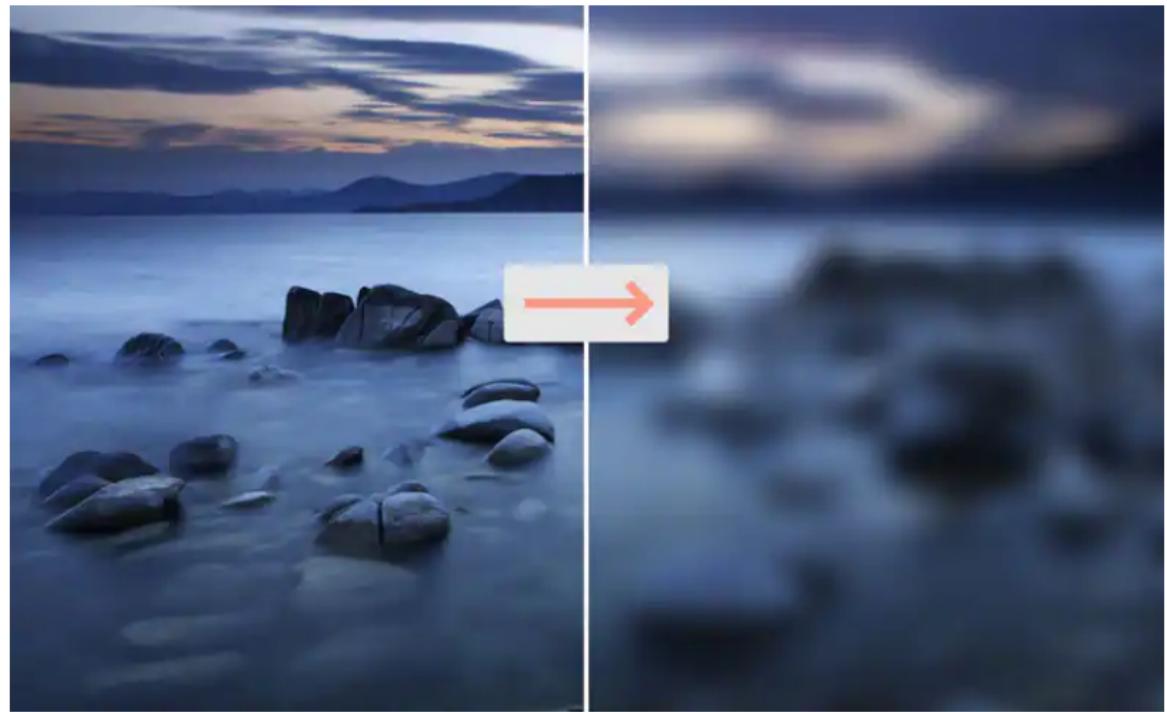
Примеры использования render to texture

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

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)

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)
 - ▶ Toon shading (edge detection, color grading)

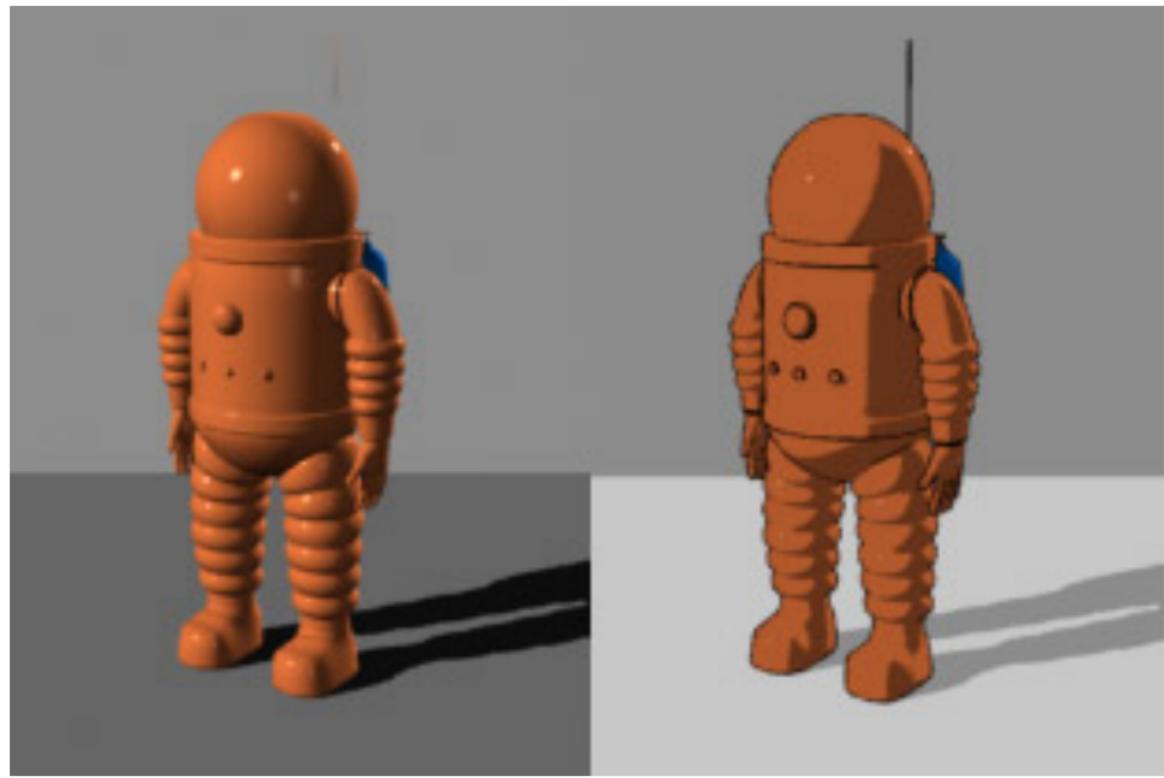
Свечение



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)
 - ▶ Toon shading (edge detection, color grading)
 - ▶ HDR, гамма-коррекция

Toon shading



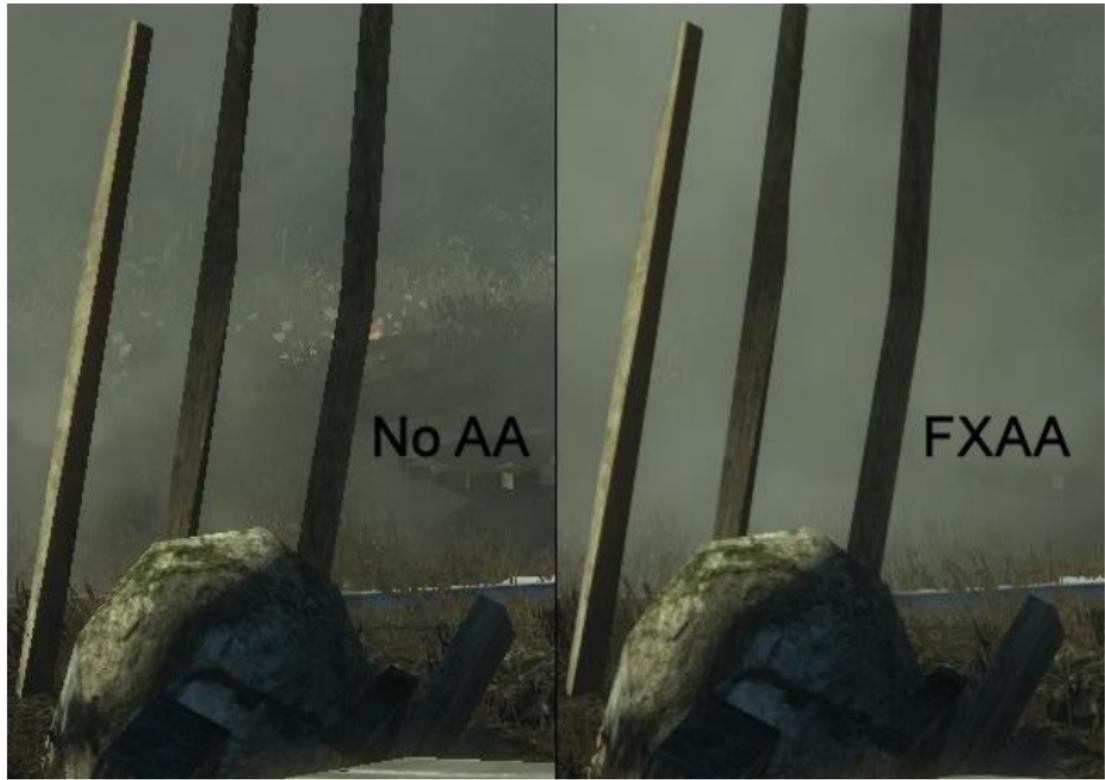
Примеры использования render to texture

- ▶ 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)
 - ▶ 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.

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.
- ▶ 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

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)
- ▶ 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 много другого

Размытие

- ▶ Усреднение значений соседних пикселей

Размытие

- ▶ Усреднение значений соседних пикселей
- ▶ Box blur: все веса одинаковые, получается слегка угловатым

Размытие

- ▶ Усреднение значений соседних пикселей
- ▶ 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
- ▶ 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

Размытие

- ▶ Усреднение значений соседних пикселей
- ▶ 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
- ▶ Обычно gaussian blur делают в два прохода: один размывает по горизонтали, второй – по вертикали

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) {
            sum += texture(source, texcoord +
                           vec2(x,y) / vec2(textureSize(source, 0)));
        }
    }

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

Gaussian blur

```
uniform sampler2D source;

in vec2 texcoord;

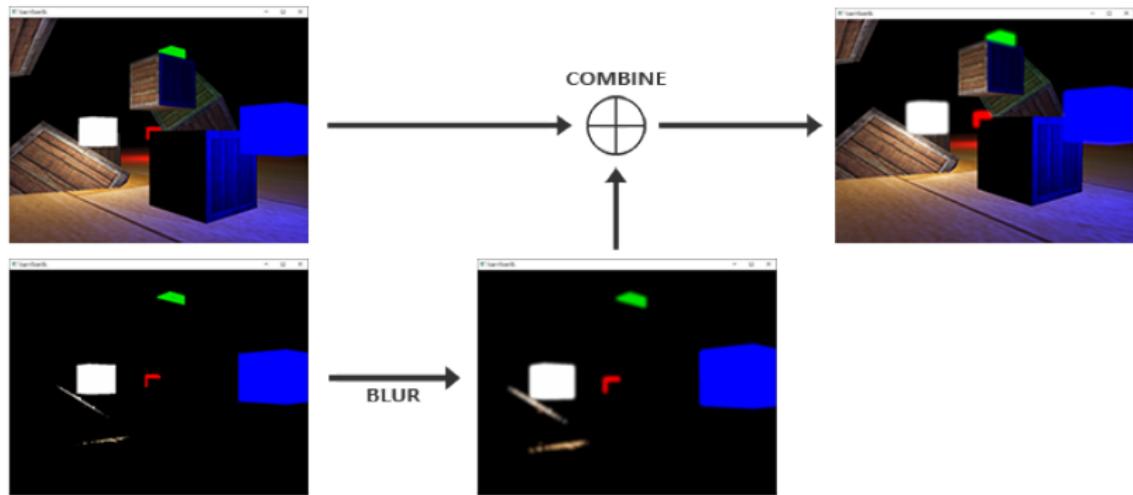
out vec4 out_color;

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) {
            float c = exp(-float(x*x + y*y) / (radius*radius));
            sum += c * texture(source, texcoord +
                vec2(x,y) / vec2(textureSize(source, 0)));
            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
- ▶ Gaussian blur (youtube video tutorial)