

# Компьютерная графика

Лекция 6: Ошибки OpenGL, расширения OpenGL, blending,  
освещение

2021

# 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 мы поговорим 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 мы поговорим 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](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, 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

- ▶ Делятся на
  - ▶ 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](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)

- ▶ 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

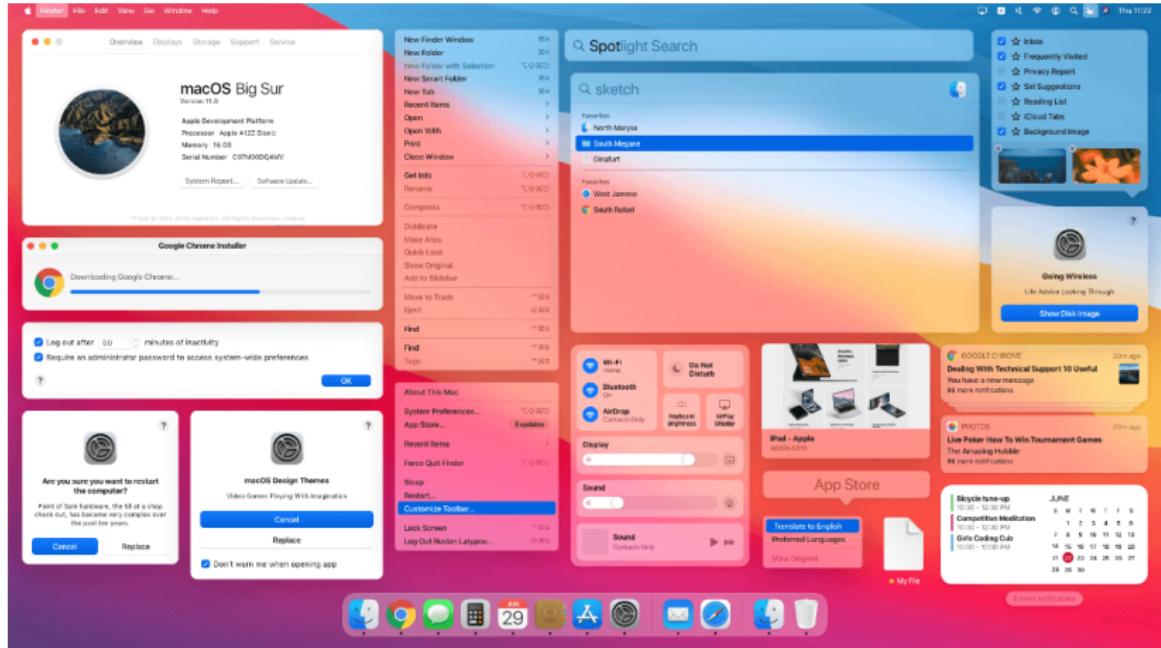


Full transparent window



Partially transparent window

# Полупрозрачность



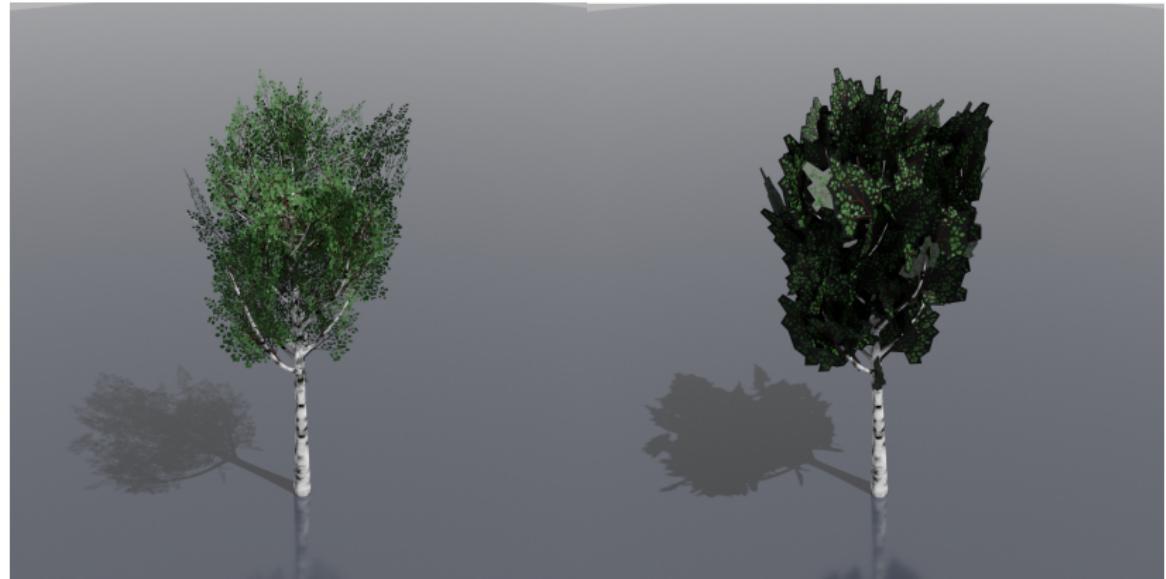
## Blending (смешивание)

- ▶ 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

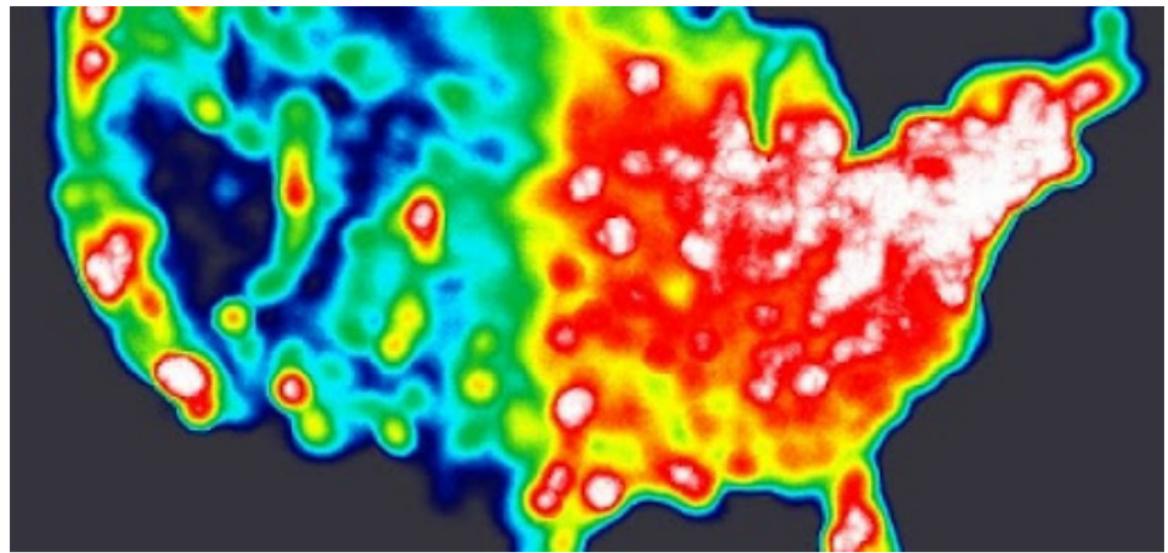
## Blending (смешивание)

- ▶ 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



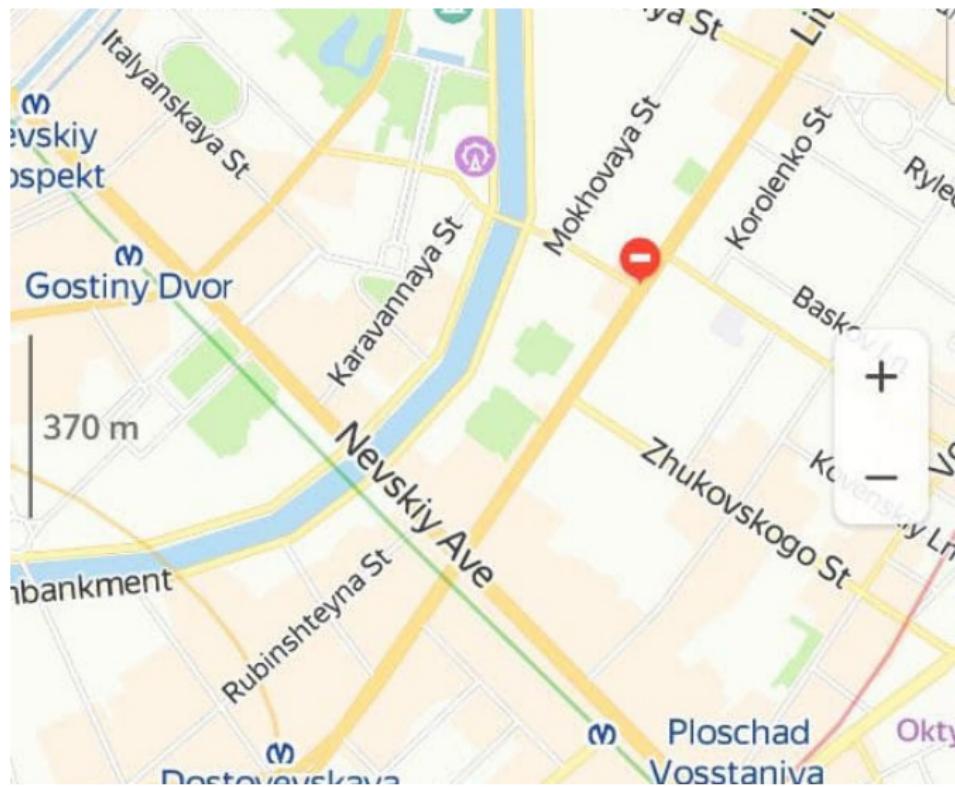
## Blending (смешивание)

- ▶ 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)
- ▶  $src$  - результат фрагментного шейдера
- ▶  $dst$  - пиксель на экране
- ▶ Уравнение blending'a:

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

## Blending (смешивание)

- ▶ Включается/выключается: glEnable(GL\_BLEND)
- ▶  $src$  - результат фрагментного шейдера
- ▶  $dst$  - пиксель на экране
- ▶ Уравнение blending'a:

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

- ▶  $f$  - 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

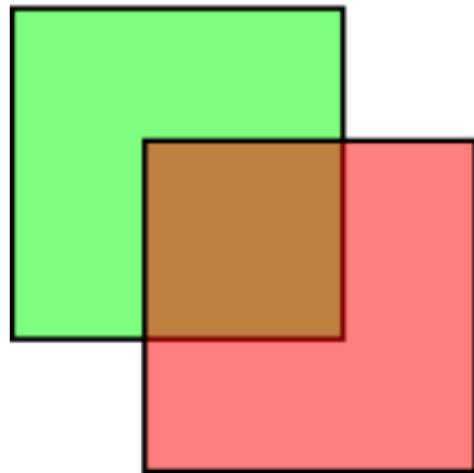
```
glBlendEquation(GL_FUNC_ADD);  
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- ▶ Это самый типичный способ использовать blending

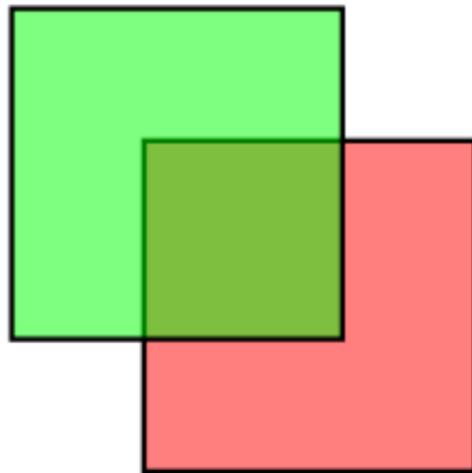
## Blending (смешивание)

- ▶ Такое смешивание некоммутативно, т.е. результат зависит от порядка рисования

Red on top

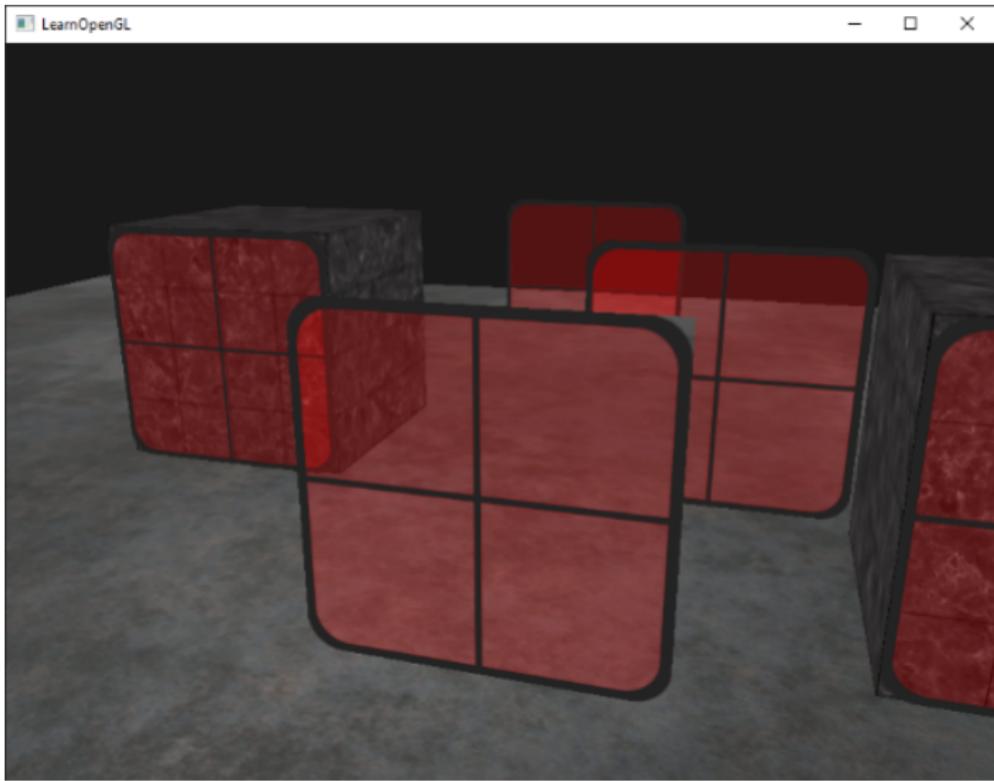


Green on top



## Blending (смешивание)

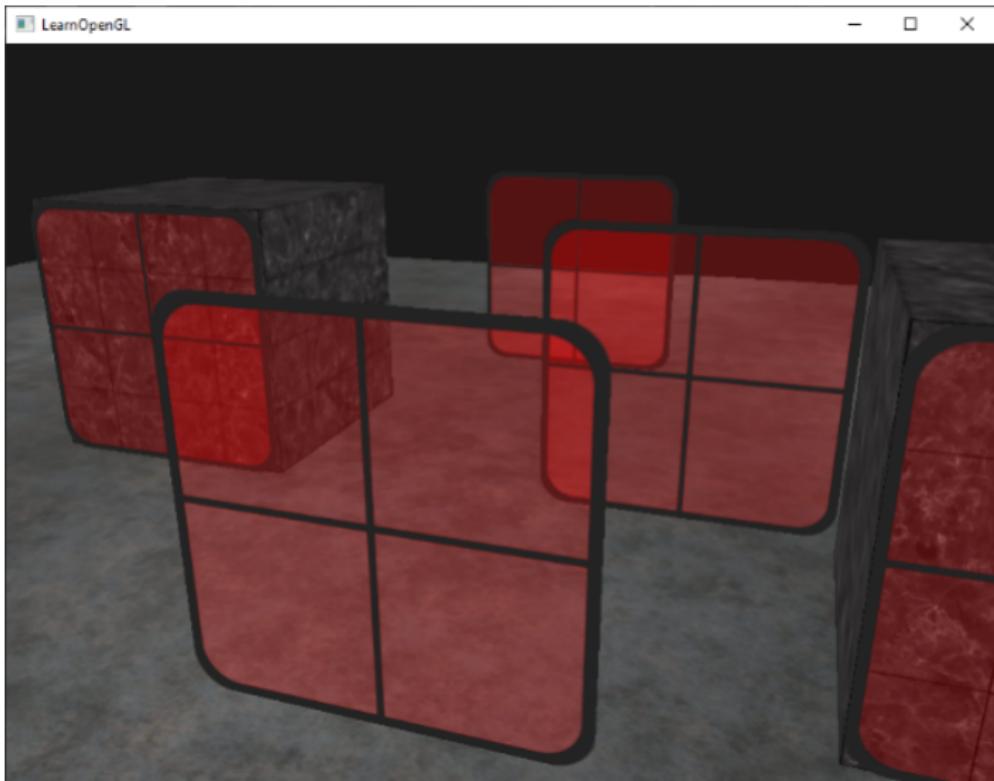
- ▶ Смешивание некорректно работает с тестом глубины



## Blending (смешивание)

- ▶ В общем случае нужно сортировать полупрозрачные объекты и рисовать в порядке уменьшения расстояния до камеры
- ▶ Order-independent transparency (OIT) - активная тема исследований

# Blending (смешивание)



# Blending (смешивание)

- ▶ [khronos.org/opengl/wiki/Blending](http://khronos.org/opengl/wiki/Blending)
- ▶ [docs.gl/gl3/glBlendFunc](http://docs.gl/gl3/glBlendFunc)
- ▶ [docs.gl/gl3/glBlendEquation](http://docs.gl/gl3/glBlendEquation)
- ▶ [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)

# Освещение

Зачем нужно освещение?

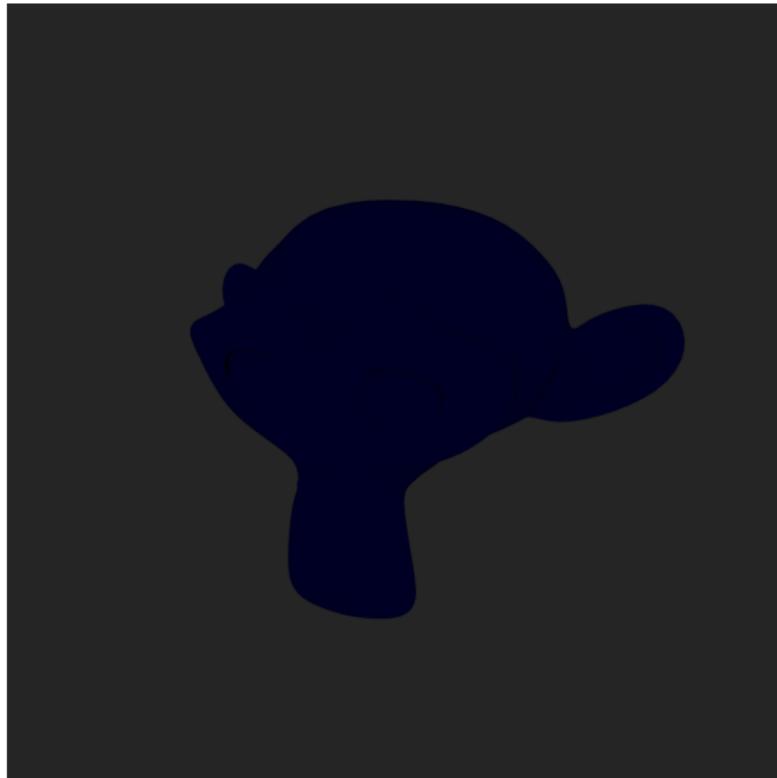
- ▶ Реалистичный рендеринг: всё, что мы видим, это свет

# Освещение

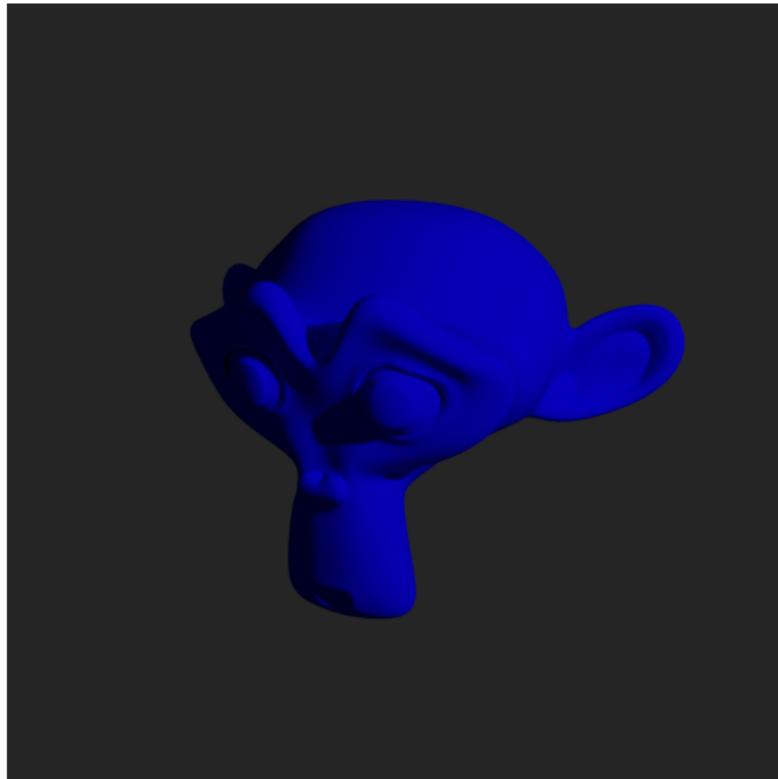
Зачем нужно освещение?

- ▶ Реалистичный рендеринг: всё, что мы видим, это свет
- ▶ Нереалистичный рендеринг:
  - ▶ Помогает понять форму объектов
  - ▶ Помогает понять соотношение между объектами в пространстве
  - ▶ Создаёт атмосферу

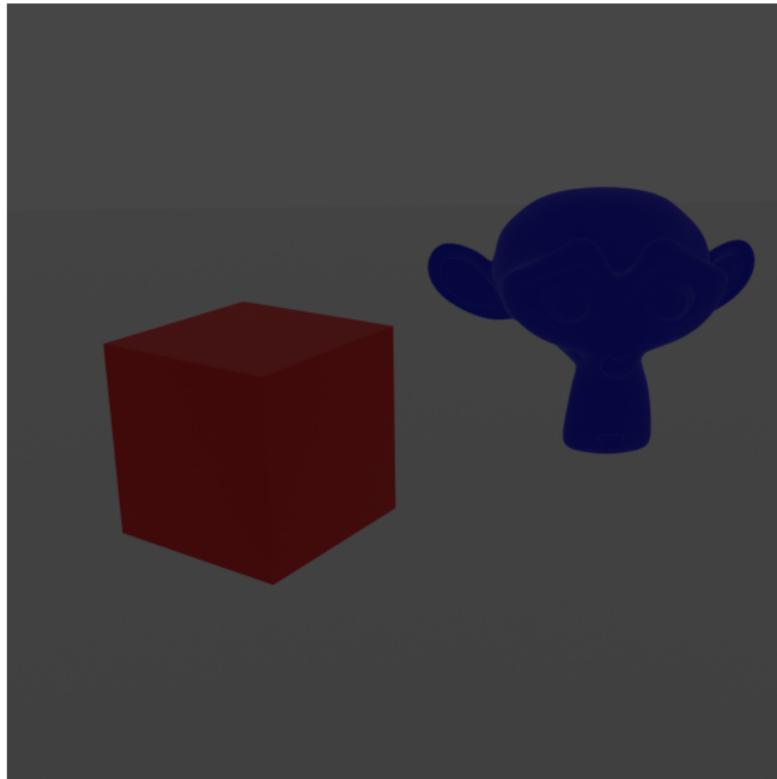
## Освещение



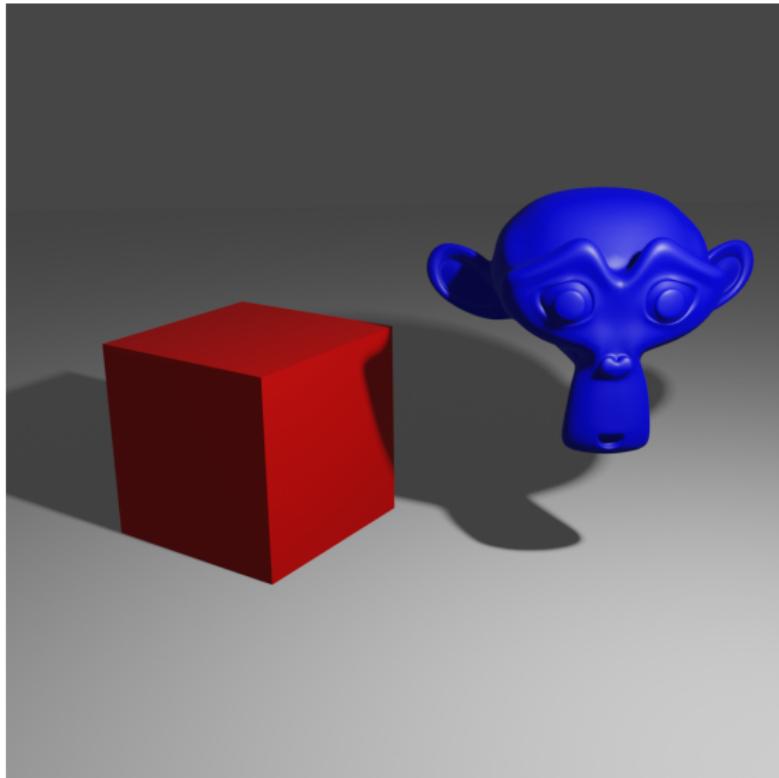
## Освещение



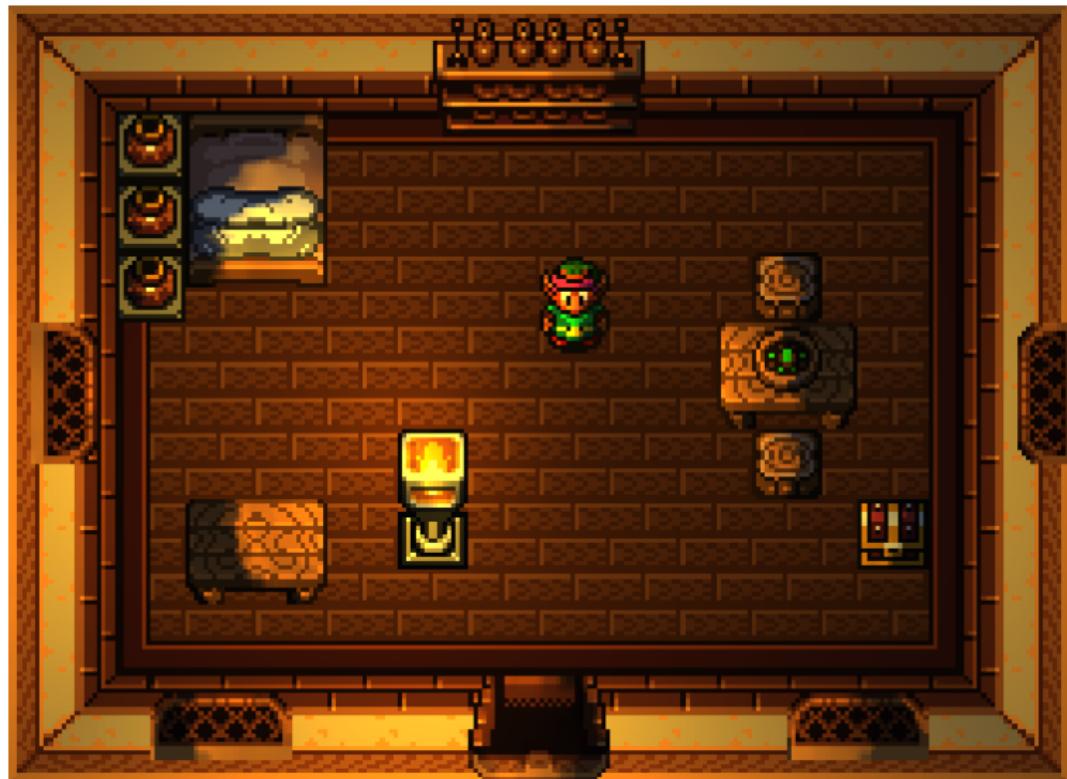
## Освещение



# Освещение



## Освещение



# Свет

- ▶ Свет - электромагнитная волна
- ▶ Пара векторных полей (электрическое  $E$  и магнитное  $B$ ), описываемых уравнениями Максвелла

$$\nabla \cdot E = 4\pi\rho$$

$$\nabla \cdot B = 0$$

$$\nabla \times E = -\frac{1}{c} \frac{\partial B}{\partial t}$$

$$\nabla \times B = \frac{1}{c} \left( 4\pi J + \frac{\partial E}{\partial t} \right)$$

# Свет

- ▶ Свет - электромагнитная волна
- ▶ Пара векторных полей (электрическое  $E$  и магнитное  $B$ ), описываемых уравнениями Максвелла

$$\nabla \cdot E = 4\pi\rho$$

$$\nabla \cdot B = 0$$

$$\nabla \times E = -\frac{1}{c} \frac{\partial B}{\partial t}$$

$$\nabla \times B = \frac{1}{c} \left( 4\pi J + \frac{\partial E}{\partial t} \right)$$

- ▶ Раскладывается в сумму волн фиксированной длины волны  $\lambda$  (монохроматические волны)
- ▶ Интерференция, дифракция, поляризация, взаимодействие с веществом

# Свет

- ▶ В реднеринге обычно достаточно геометрической оптики - предела при  $\lambda \rightarrow 0$

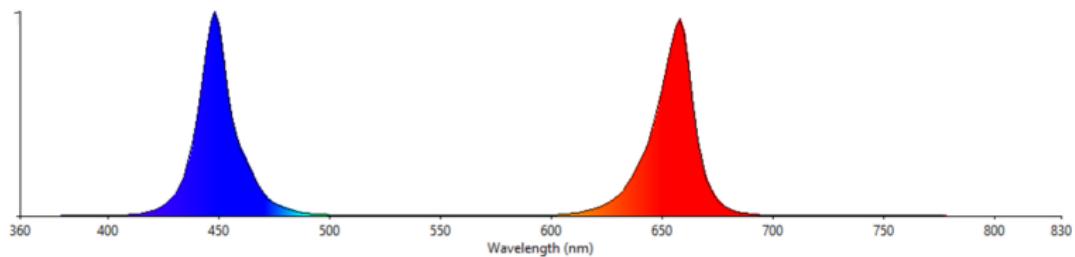
# Свет

- ▶ В реднерионге обычно достаточно геометрической оптики - предела при  $\lambda \rightarrow 0$
- ▶ Свет распространяется прямыми лучами (исключение - граница раздела сред или неоднородные среды)
- ▶ Свет распространяется бесконечно быстро ( $c \rightarrow \infty$ )
- ▶ Луч света разбивается в сумму монохроматических лучей, каждая имеет свою амплитуду (количество света)

# Свет

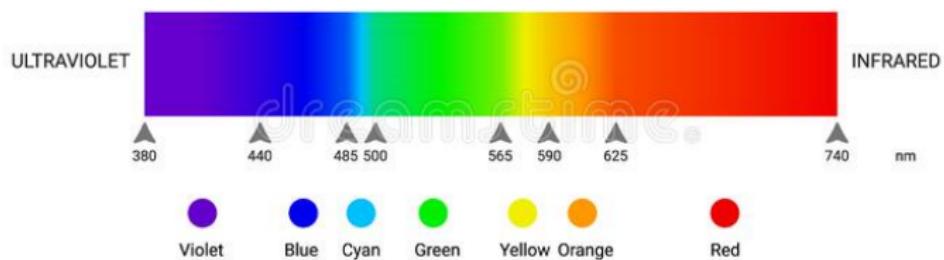
- ▶ В реднерионге обычно достаточно геометрической оптики - предела при  $\lambda \rightarrow 0$
- ▶ Свет распространяется прямыми лучами (исключение - граница раздела сред или неоднородные среды)
- ▶ Свет распространяется бесконечно быстро ( $c \rightarrow \infty$ )
- ▶ Луч света разбивается в сумму монохроматических лучей, каждая имеет свою амплитуду (количество света)
- ▶ Луч света - распределение амплитуд по длинам волн

# Свет



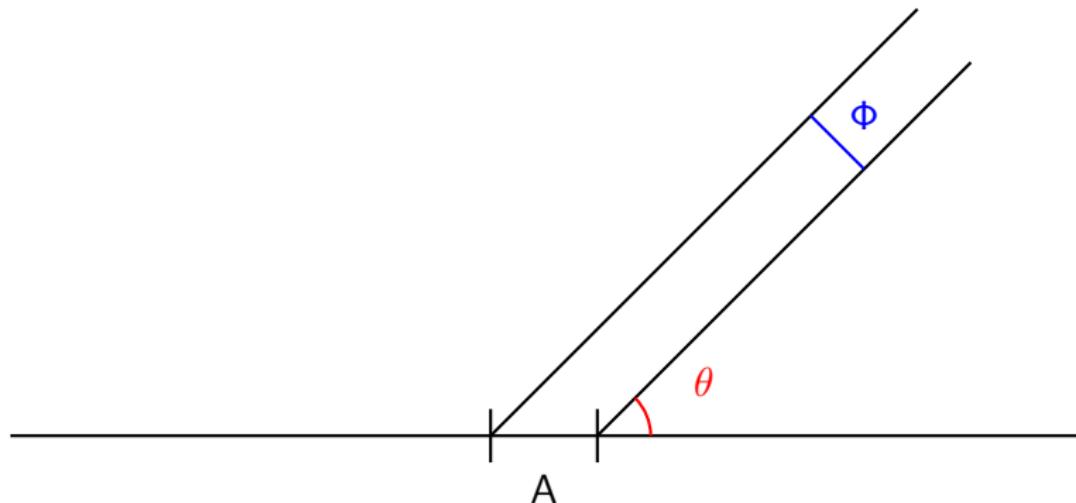
# Видимый спектр

## Visible spectrum



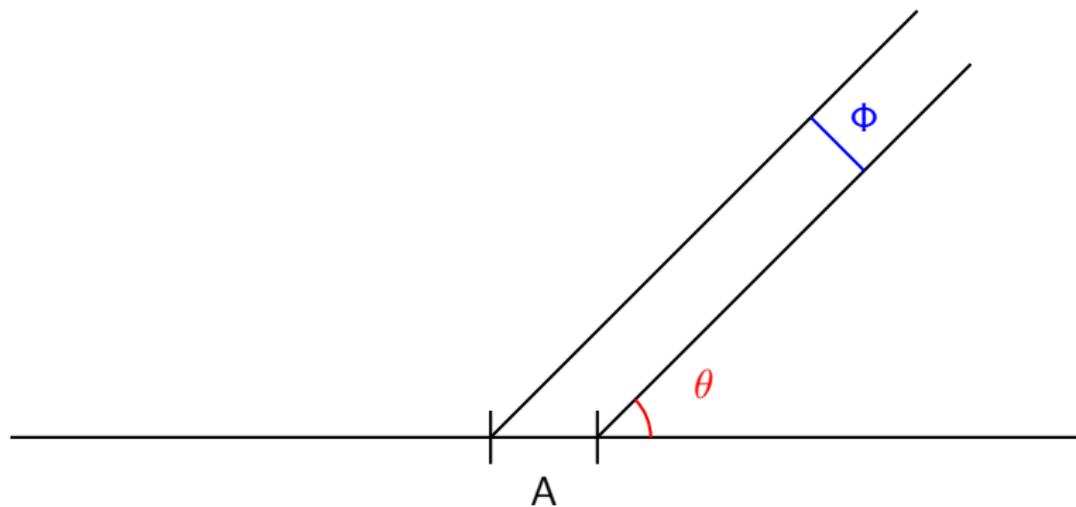
## Косинус угла падения

- ▶ На поверхность площадью  $A$  падает световой поток с площадью поперечного сечения  $\Phi$
- ▶ Тогда  $\Phi = A \cdot \cos \theta$
- ▶



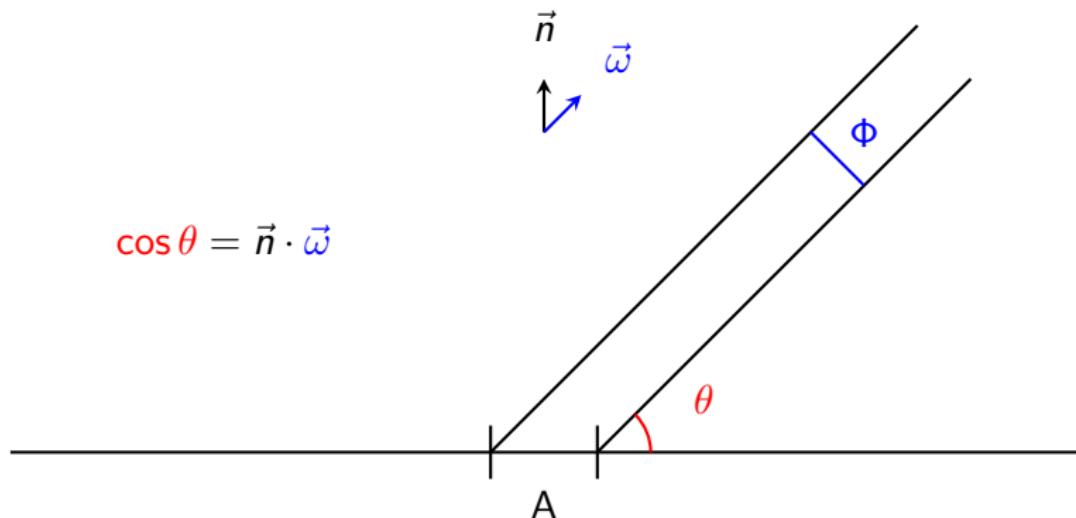
## Косинус угла падения

- ▶ На поверхность площадью  $A$  падает световой поток с площадью поперечного сечения  $\Phi$
- ▶ Тогда  $\Phi = A \cdot \cos \theta$
- ▶  $\Rightarrow$  На единицу площади приходится  $\cos \theta$  от общей плотности потока



## Косинус угла падения

- ▶ На поверхность площадью  $A$  падает световой поток с площадью поперечного сечения  $\Phi$
- ▶ Тогда  $\Phi = A \cdot \cos \theta$
- ▶  $\Rightarrow$  На единицу площади приходится  $\cos \theta$  от общей плотности потока



## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?

## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:

## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет

## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет
  - ▶ Зеркало: отражает свет в строго определённом направлении

## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет
  - ▶ Зеркало: отражает свет в строго определённом направлении
  - ▶ Старое, плохо отполированное зеркало / лакированная поверхность: отражает свет примерно в определённом направлении

## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет
  - ▶ Зеркало: отражает свет в строго определённом направлении
  - ▶ Старое, плохо отполированное зеркало / лакированная поверхность: отражает свет примерно в определённом направлении
  - ▶ Диффузная (ламбертова) поверхность: отражает свет во все стороны равномерно

## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет
  - ▶ Зеркало: отражает свет в строго определённом направлении
  - ▶ Старое, плохо отполированное зеркало / лакированная поверхность: отражает свет примерно в определённом направлении
  - ▶ Диффузная (ламбертова) поверхность: отражает свет во все стороны равномерно
  - ▶ Стекло: преломляет свет

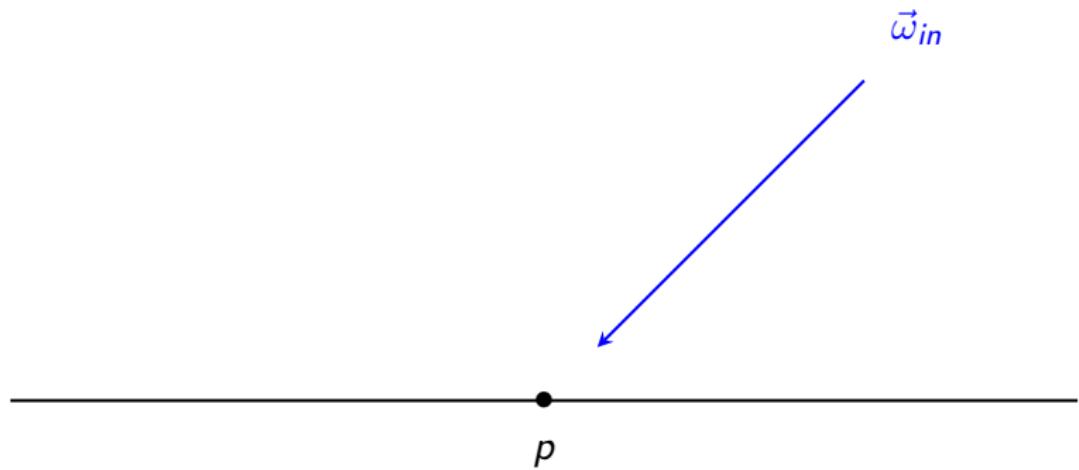
## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет
  - ▶ Зеркало: отражает свет в строго определённом направлении
  - ▶ Старое, плохо отполированное зеркало / лакированная поверхность: отражает свет примерно в определённом направлении
  - ▶ Диффузная (ламбертова) поверхность: отражает свет во все стороны равномерно
  - ▶ Стекло: преломляет свет
  - ▶ Вода: частично отражает, частично преломляет (закон Френеля)

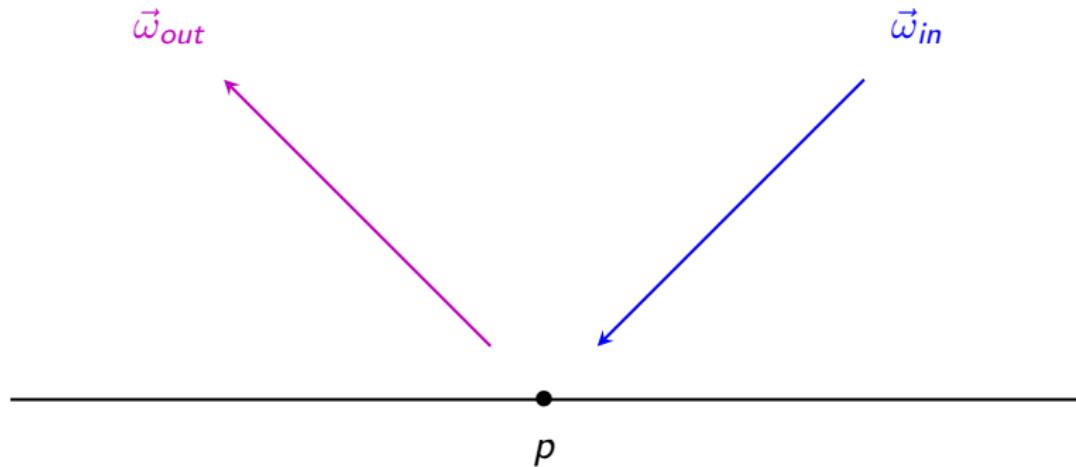
## Столкновение света с поверхностью

- ▶ Что происходит, когда луч света сталкивается с некой поверхностью?
- ▶ Зависит от её *материала*:
  - ▶ Абсолютно чёрное тело: поглощает свет
  - ▶ Зеркало: отражает свет в строго определённом направлении
  - ▶ Старое, плохо отполированное зеркало / лакированная поверхность: отражает свет примерно в определённом направлении
  - ▶ Диффузная (ламбертова) поверхность: отражает свет во все стороны равномерно
  - ▶ Стекло: преломляет свет
  - ▶ Вода: частично отражает, частично преломляет (закон Френеля)
- ▶ ⇒ Свет, выходящий в каком-то направлении, может зависеть от света, пришедшего из *любого* направления

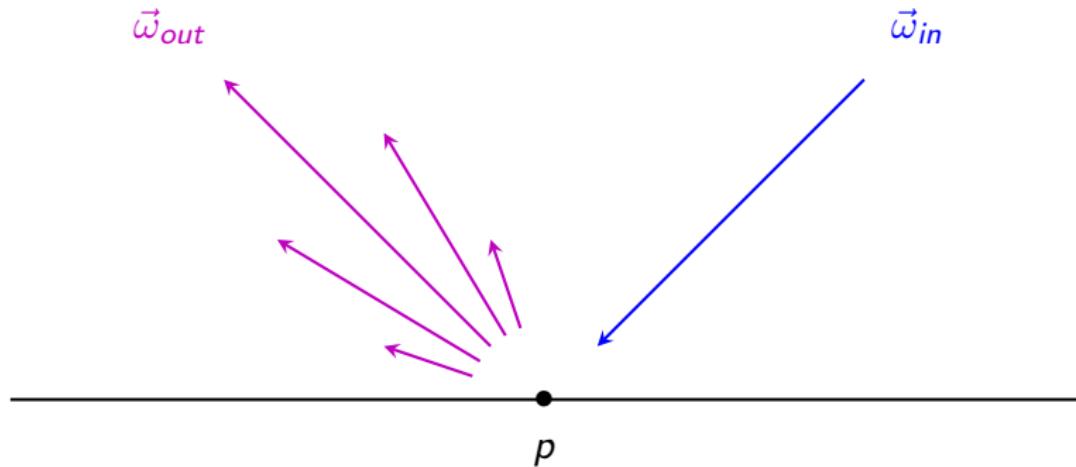
# Абсолютно чёрное тело



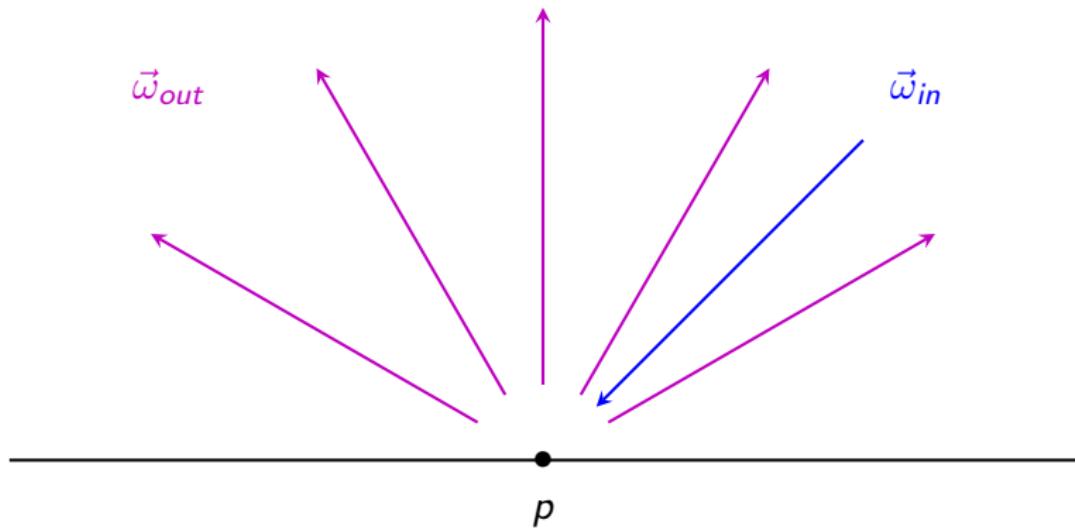
# Зеркало



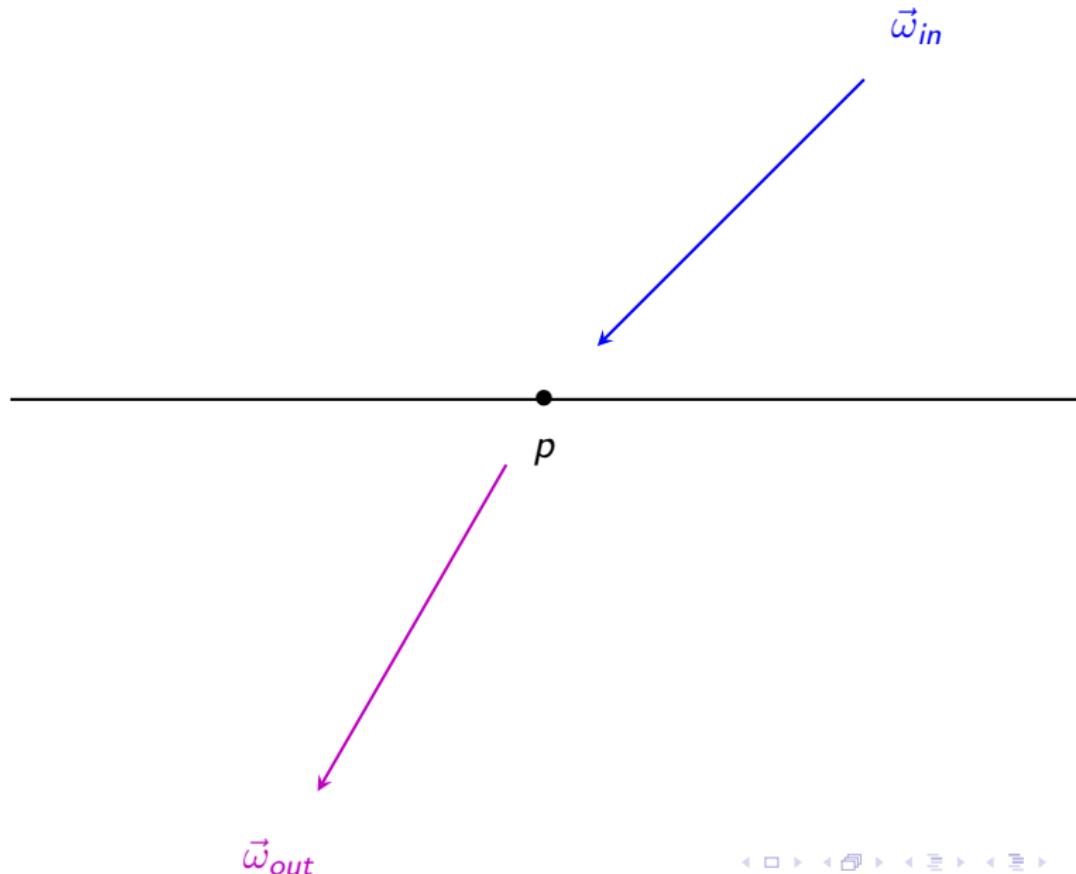
## Старое зеркало / лакированная поверхность



# Диффузная поверхность



# Стекло



## Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

(1)

## Уравнение рендеринга

$$I_{\text{out}}(p, \vec{\omega}_{\text{out}}, \lambda) = I_e(p, \vec{\omega}_{\text{out}}) + \int_{\mathbb{S}^2} I_{\text{in}}(p, \vec{\omega}_{\text{in}}, \lambda) \cdot f(p, \vec{\omega}_{\text{in}}, \vec{\omega}_{\text{out}}, \lambda) \cdot (\vec{\omega}_{\text{in}} \cdot \vec{n}) d\vec{\omega}_{\text{in}}$$

(1)

- ▶ Уравнение для конкретной точки  $p$

## Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

(1)

- ▶ Уравнение для конкретной точки  $p$
- ▶  $I_{out}$  - количество света, выходящего из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$

## Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

(1)

- ▶ Уравнение для конкретной точки  $p$
- ▶  $I_{out}$  - количество света, выходящего из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$
- ▶  $I_e$  - количество света, испускаемого поверхностью из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$

## Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in}$$

(1)

- ▶ Уравнение для конкретной точки  $p$
- ▶  $I_{out}$  - количество света, выходящего из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$
- ▶  $I_e$  - количество света, испускаемого поверхностью из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$
- ▶  $I_{in}$  - количество света, приходящего в точку  $p$  из направления  $\vec{\omega}_{in}$  с длиной волны  $\lambda$

## Уравнение рендеринга

$$I_{out}(p, \vec{\omega}_{out}, \lambda) = I_e(p, \vec{\omega}_{out}) + \int_{\mathbb{S}^2} I_{in}(p, \vec{\omega}_{in}, \lambda) \cdot f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) \cdot (\vec{\omega}_{in} \cdot \vec{n}) d\vec{\omega}_{in} \quad (1)$$

- ▶ Уравнение для конкретной точки  $p$
- ▶  $I_{out}$  - количество света, выходящего из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$
- ▶  $I_e$  - количество света, испускаемого поверхностью из точки  $p$  в направлении  $\vec{\omega}_{out}$  с длиной волны  $\lambda$
- ▶  $I_{in}$  - количество света, приходящего в точку  $p$  из направления  $\vec{\omega}_{in}$  с длиной волны  $\lambda$
- ▶  $f$  - функция, определяющая, сколько света с длиной волны  $\lambda$ , пришедшего из направления  $\vec{\omega}_{in}$ , отразится в направлении  $\vec{\omega}_{out}$

# Уравнение рендеринга

- ▶ Абсолютно чёрное тело:  $f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) = 0$

# Уравнение рендеринга

- ▶ Абсолютно чёрное тело:  $f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) = 0$
- ▶ Старое зеркало:  $f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) = (R_{\vec{n}}(\vec{\omega}_{in}) \cdot \vec{\omega}_{out})^7$ 
  - ▶  $R$  - отраженный вектор:  $R_{\vec{n}}(\vec{\omega}) = 2\vec{n} \cdot (\vec{n} \cdot \vec{\omega}) - \vec{\omega}$

# Уравнение рендеринга

- ▶ Абсолютно чёрное тело:  $f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) = 0$
- ▶ Старое зеркало:  $f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) = (R_{\vec{n}}(\vec{\omega}_{in}) \cdot \vec{\omega}_{out})^7$ 
  - ▶  $R$  - отраженный вектор:  $R_{\vec{n}}(\vec{\omega}) = 2\vec{n} \cdot (\vec{n} \cdot \vec{\omega}) - \vec{\omega}$
- ▶ Диффузная поверхность:  $f(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) = 1$

# Цвет

- ▶ Зависимость  $f$  от  $\lambda$  обеспечивает цвет объектов

# Цвет

- ▶ Зависимость  $f$  от  $\lambda$  обеспечивает цвет объектов
- ▶ Мы видим свет, отражённый объектом
- ▶ Объект синего цвета не отражает красный цвет (кажется чёрным при освещении красным светом)

# Материал = BSDF

- ▶  $f$  определяет *материал* объекта

# Материал = BSDF

- ▶  $f$  определяет *материал* объекта
- ▶ Если  $f$  только отражает свет, её называют BRDF:  
Bidirectional Reflectance Distribution Function

# Материал = BSDF

- ▶  $f$  определяет *материал* объекта
- ▶ Если  $f$  только отражает свет, её называют BRDF:  
Bidirectional Reflectance Distribution Function
- ▶ Если  $f$  только преломляет свет, её называют BRTF:  
Bidirectional Transmittance Distribution Function

# Материал = BSDF

- ▶  $f$  определяет *материал* объекта
- ▶ Если  $f$  только отражает свет, её называют BRDF:  
Bidirectional Reflectance Distribution Function
- ▶ Если  $f$  только преломляет свет, её называют BRTF:  
Bidirectional Transmittance Distribution Function
- ▶ В общем случае её называют BSTF: Bidirectional Scattering Distribution Function

# BSDF

- ▶ Обычно BSTF предполагается нормированной:

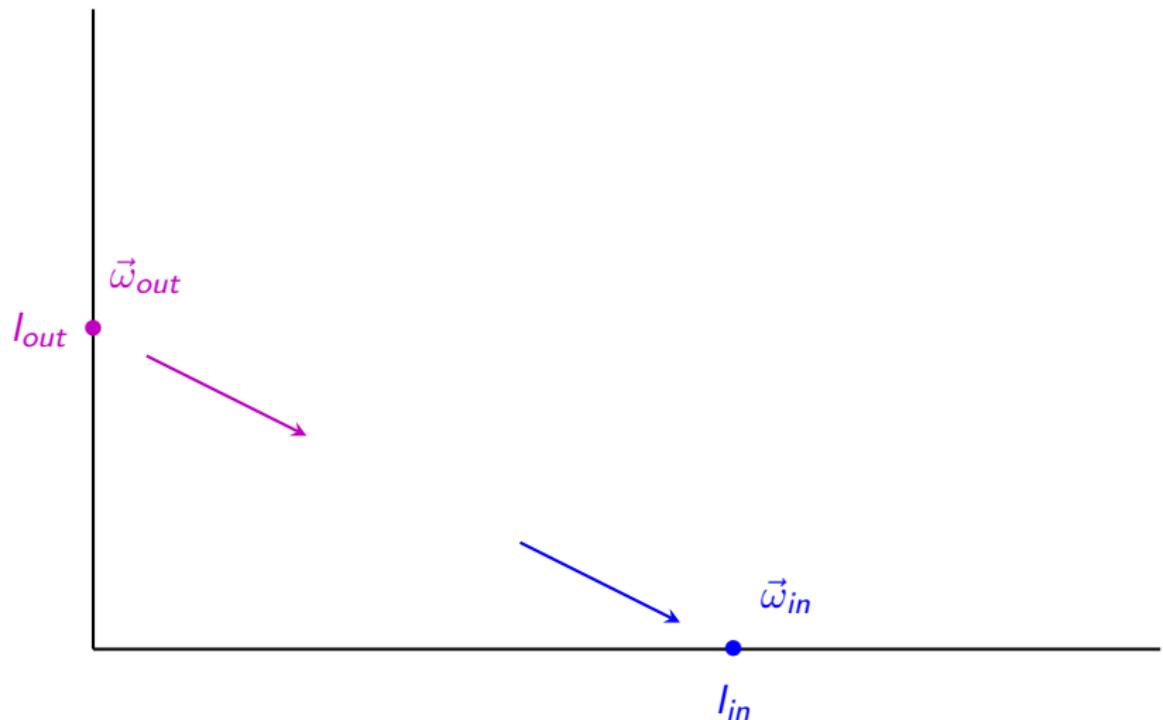
$$\int_{\mathbb{S}^2} \textcolor{green}{f}(p, \vec{\omega}_{in}, \vec{\omega}_{out}, \lambda) d\vec{\omega}_{in} = 1 \quad (2)$$

## Уравнение рендеринга

- ▶ Откуда взять  $I_{in}$ ?

## Уравнение рендеринга

- ▶ Откуда взять  $I_{in}$ ? Ответ: это чей-то  $I_{out}$



# Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены

## Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены
- ▶ Геометрия сцены связывает уравнения для разных точек

## Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены
- ▶ Геометрия сцены связывает уравнения для разных точек
- ▶ Обычно вместо всех возможных значений  $\lambda$  берут дискретный набор (**красный, зелёный, синий**)

## Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены
- ▶ Геометрия сцены связывает уравнения для разных точек
- ▶ Обычно вместо всех возможных значений  $\lambda$  берут дискретный набор (**красный, зелёный, синий**)
- ▶ Задача называется GI: Global Illumination

## Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены
- ▶ Геометрия сцены связывает уравнения для разных точек
- ▶ Обычно вместо всех возможных значений  $\lambda$  берут дискретный набор (**красный, зелёный, синий**)
- ▶ Задача называется GI: Global Illumination
- ▶ Решать очень сложно, есть бесчисленное количество методов

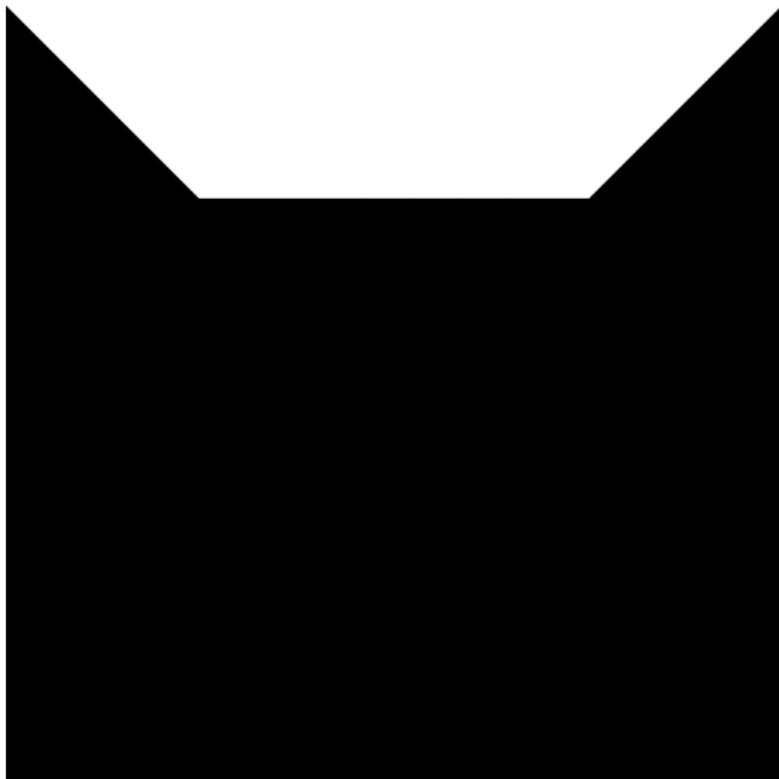
## Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены
- ▶ Геометрия сцены связывает уравнения для разных точек
- ▶ Обычно вместо всех возможных значений  $\lambda$  берут дискретный набор (**красный, зелёный, синий**)
- ▶ Задача называется GI: Global Illumination
- ▶ Решать очень сложно, есть бесчисленное количество методов
- ▶ Реалистичный offline рендеринг: медленно, но очень хорошая картинка

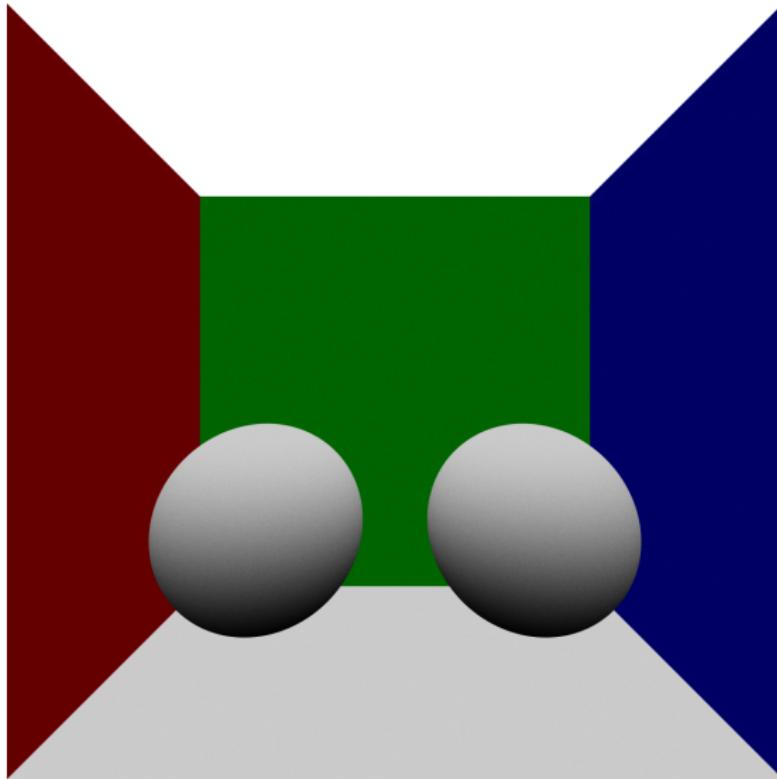
## Уравнение рендеринга

- ▶ Интегральное уравнение для каждой точки каждой поверхности сцены
- ▶ Геометрия сцены связывает уравнения для разных точек
- ▶ Обычно вместо всех возможных значений  $\lambda$  берут дискретный набор (**красный, зелёный, синий**)
- ▶ Задача называется GI: Global Illumination
- ▶ Решать очень сложно, есть бесчисленное количество методов
- ▶ Реалистичный offline рендеринг: медленно, но очень хорошая картинка
- ▶ Real-time рендеринг: нужны грубые аппроксимации

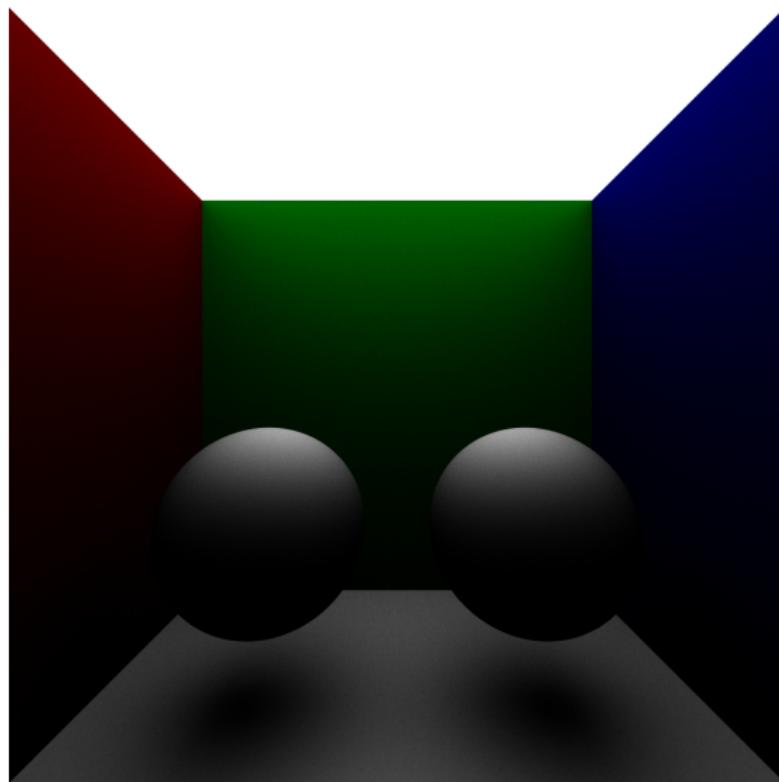
Только источник света



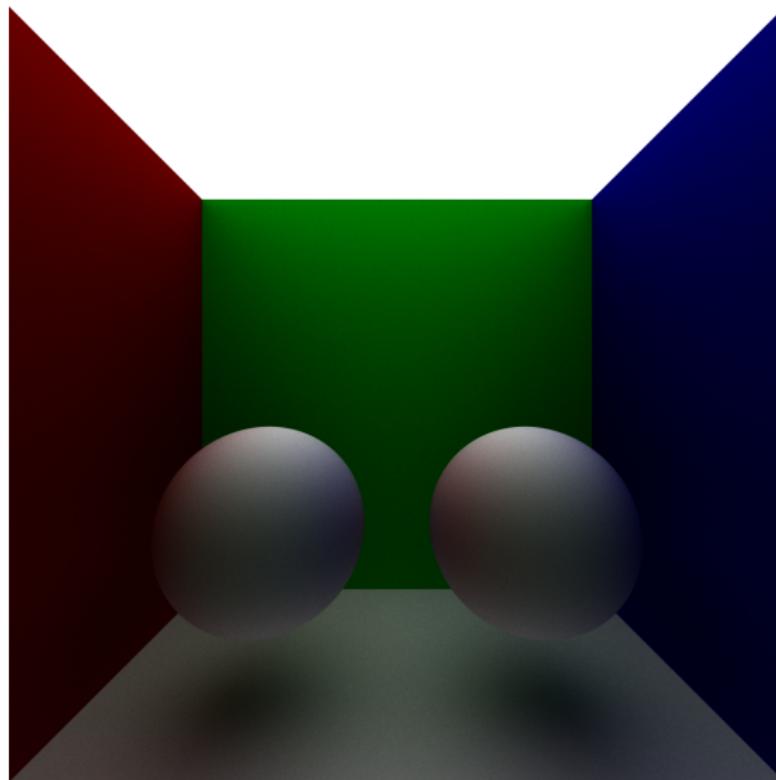
Только прямое освещение



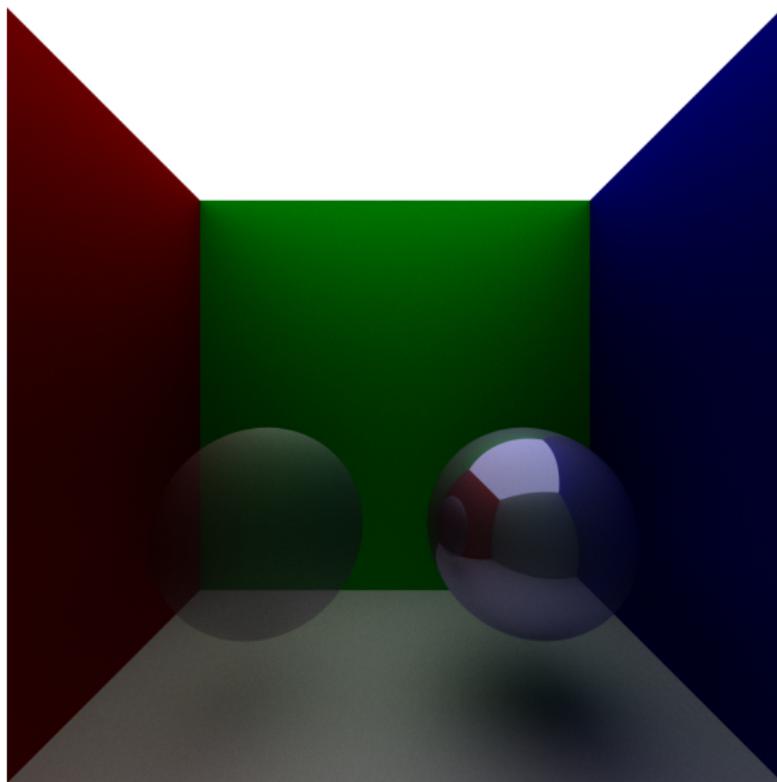
## Прямое освещение и тени



## Прямое и непрямое освещение, тени



## Прозрачность и отражение



## Ambient освещение

- ▶ Свет, приходящий "отовсюду"
- ▶ На улице: небо
- ▶ В помещении: диффузное отражение от стен, пола, потолка, других объектов
- ▶ Определяется своим цветом

## Ambient освещение: фрагментный шейдер

```
uniform vec3 ambient_color;  
  
in vec3 color;  
  
layout (location = 0) out vec4 out_color;  
  
void main() {  
    vec3 result_color = color * ambient_color;  
    out_color = vec4(result_color, 1.0);  
}
```

## Источники света

- ▶ Два самых часто используемых типа: направленные и точечные

## Источники света

- ▶ Два самых часто используемых типа: направленные и точечные
- ▶ Направленные:
  - ▶ Задаются направлением и яркостью
  - ▶ Моделируют удалённые источники - солнце, луна

# Источники света

- ▶ Два самых часто используемых типа: направленные и точечные
- ▶ Направленные:
  - ▶ Задаются направлением и яркостью
  - ▶ Моделируют удалённые источники - солнце, луна
- ▶ Точечные:
  - ▶ Задаются расположением и функцией зависимости яркости от расстояния
  - ▶ Моделируют (относительно) близкие источники - костёр, свеча, лампа

## Прямое освещение

- ▶ Нужно найти скалярное произведение между направлением на источник света и нормалью к поверхности

## Прямое освещение

- ▶ Нужно найти скалярное произведение между направлением на источник света и нормалью к поверхности
- ▶ N.B. отрицательный результат означает, что поверхность обращена не по направлению к свету

## Прямое освещение

- ▶ Нужно найти скалярное произведение между направлением на источник света и нормалью к поверхности
- ▶ N.B. отрицательный результат означает, что поверхность обращена не по направлению к свету
- ▶ Нужно найти яркость источника света в данной точке

## Прямое освещение

- ▶ Нужно найти скалярное произведение между направлением на источник света и нормалью к поверхности
- ▶ N.B. отрицательный результат означает, что поверхность обращена не по направлению к свету
- ▶ Нужно найти яркость источника света в данной точке
- ▶ Результат = произведение яркости, цвета объекта и коэффициента освещённости

## Направленный источник света: фрагментный шейдер

```
uniform vec3 light_direction;  
uniform vec3 light_color;  
  
in vec3 normal;  
in vec3 color;  
  
layout (location = 0) out vec4 out_color;  
  
void main() {  
    float cosine = dot(normal, light_direction);  
    float light_factor = max(0.0, cosine);  
    vec3 result_color = light_factor * color * light_color;  
    out_color = vec4(result_color, 1.0);  
}
```

## Точечный источник света

- ▶ Обычно яркость убывает с расстоянием

## Точечный источник света

- ▶ Обычно яркость убывает с расстоянием
- ▶ Часто берут такую функцию затухания (attenuation):

$$f(r) = \frac{1}{C_0 + C_1 r + C_2 r^2} \quad (3)$$

## Точечный источник света: фрагментный шейдер

```
uniform vec3 light_position;
uniform vec3 light_attenuation; // c0, c1, c2
uniform vec3 light_color;

in vec3 position;
in vec3 normal;
in vec3 color;

layout (location = 0) out vec4 out_color;

void main() {
    vec3 light_vector = light_position - position;
    vec3 light_direction = normalize(light_vector);
    float cosine = dot(normal, light_direction);
    float light_factor = max(0.0, cosine);

    float light_distance = length(light_vector);
    float light_intensity = dot(light_attenuation,
        vec3(1.0, light_distance, light_distance * light_distance));

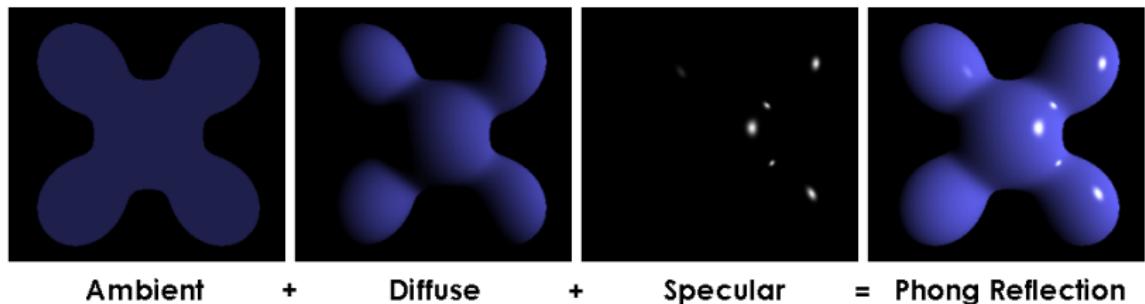
    vec3 result_color = light_factor * light_intensity * color
        * light_color;
    out_color = vec4(result_color, 1.0);
}
```

## Модель Фонга

- ▶ Можно добавить отражённый свет (specular), имитируя гладкую поверхность

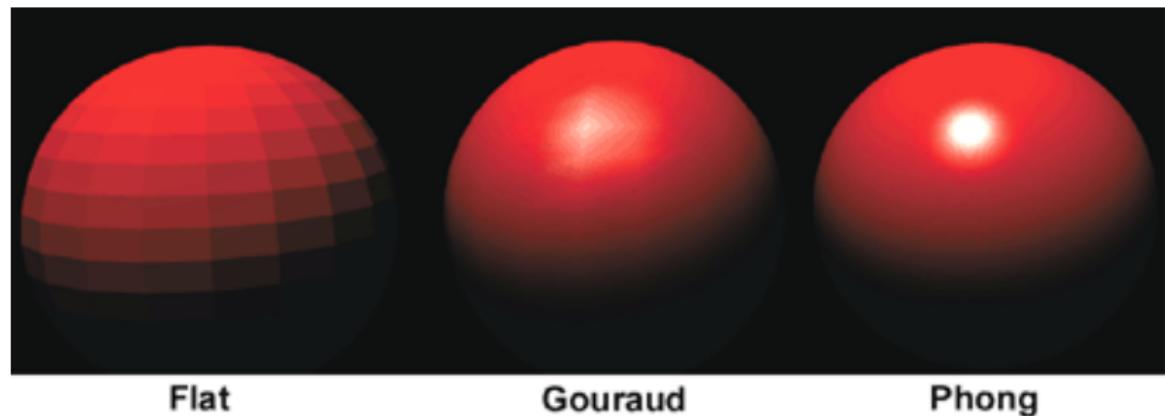
# Модель Фонга

- ▶ Можно добавить отражённый свет (specular), имитируя гладкую поверхность
- ▶ Ambient + direct (diffuse) + specular = модель Фонга



## Модель Гуро

- ▶ То же самое, но в вершинном шейдере, и интерполировать результат
- ▶ Выглядит плохо, в современности не используется



## Ссылки

- ▶ [learnopengl.com/Lighting/Basic-Lighting](http://learnopengl.com/Lighting/Basic-Lighting)
- ▶ [opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading](http://opengl-tutorial.org/beginners-tutorials/tutorial-8-basic-shading)
- ▶ [lighthouse3d.com/tutorials/glsl-tutorial/point-lights](http://lighthouse3d.com/tutorials/glsl-tutorial/point-lights)