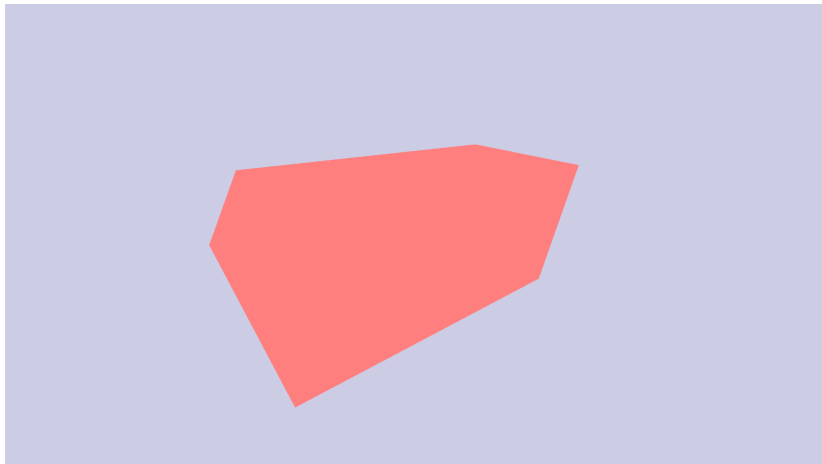


# Компьютерная графика

## Практика 12: Volume rendering

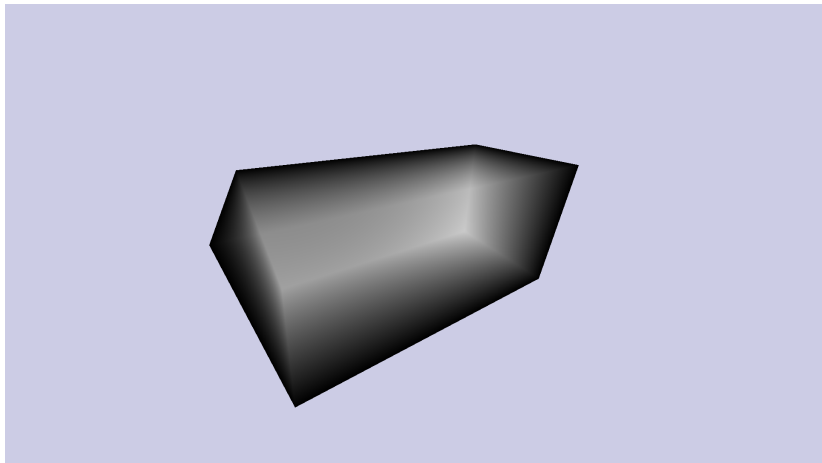
2022



# Задание 1

Находим пересечение с AABV объекта (во фрагментном шейдере)

- ▶ Вычисляем **нормированный** вектор направления из камеры в текущую точку поверхности – это вектор направления луча
- ▶ Вычисляем пересечение этого луча с AABV: интервал  $[t_{min}, t_{max}]$  для которых  $p + t \cdot d$  содержится в AABV (в коде уже есть функция `intersect_bbox`, возвращает `vec2(tmin, tmax)`)
- ▶ Делаем `tmin = max(tmin, 0.0)`, чтобы не включать часть пересечения сзади камеры
- ▶ В качестве цвета пикселя выводим `vec3(tmax - tmin)` (это значение часто будет больше единицы, так что можно разделить, например, на 4.0)



## Задание 2

Вычисляем optical depth куба (во фрагментном шейдере)

- ▶ Заводим константу для коэффициента поглощения:  
`absorption = 1.0`
- ▶ Вычисляем optical depth:  
`optical_depth = (tmax - tmin) * absorption`
- ▶ Вычисляем непрозрачность пикселя:  
`opacity = 1.0 - exp(-optical_depth)`
- ▶ Записываем значение opacity в альфа-канал результирующего цвета (RGB-каналы заполните вашим любимым цветом)
- ▶ Можно поиграться со значением absorption чтобы понять, как оно влияет на результат



## Задание 3

Загружаем 3D текстуру

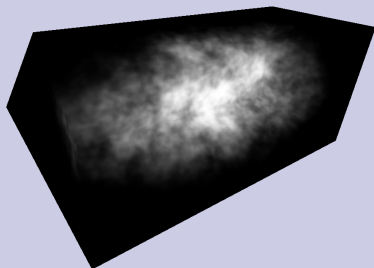
- ▶ Создаём текстуру типа `GL_TEXTURE_3D`, min/mag фильтры – `GL_LINEAR`
- ▶ Устанавливаем параметры `WRAP_R`, `WRAP_S`, `WRAP_T` в `GL_CLAMP_TO_EDGE`
- ▶ Считываем данные из файла `cloud_data_path` (128x64x64, одноканальная, 1 байт на пиксель):
  - ▶ Заводим `std::vector<char> pixels(...)` нужного размера
  - ▶ Открываем файл `std::ifstream input(path, std::ios::binary)`
  - ▶ Читаем данные `input.read(pixels.data(), pixels.size())`
- ▶ Загружаем в текстуру с помощью `glTexImage3D` (internal format – `GL_R8`, format – `GL_RED`, type – `GL_UNSIGNED_BYTE`)
- ▶ Добавляем текстуру в шейдер (`uniform sampler3D`), выводим в качестве цвета значение из текстуры в точке
$$p = position + direction * (tmin + tmax) / 2.0$$
  - ▶ Нужно перевести пространственные координаты в текстурные:
$$(p - box\_min) / (bbox\_max - bbox\_min)$$
  - ▶ Удобно завести функцию, возвращающую значение из текстуры по точке в пространстве
- ▶ В качестве альфа-канала возьмём 1, иначе ничего не увидим

## Задание 3

N.B. можно взять другую текстуру: `bunny.data` (есть в репозитории с заданием)

- ▶ Размер – 64x64x64
- ▶ `bbox_min = vec3(-1, -1, -1)`
- ▶ `bbox_max = vec3( 1, 1, 1)`





## Задание 4

Вычисляем optical depth с помощью front-to-back алгоритма (во фрагментном шейдере)

- ▶ Инициализируем `optical_depth = 0`
- ▶ Делаем цикл, например, в 64 шага; один шаг цикла соответствует  $1/64$  части отрезка  
 $dt = (t_{max} - t_{min}) / 64$ 
  - ▶ Вместо 64 можно взять любое другое число; чем больше, тем красивее и медленнее
  - ▶ Каждой итерации  $i$  цикла соответствует значение  
 $t = t_{min} + (i + 0.5) * dt$
  - ▶ Каждому значению  $t$  соответствует точка луча  
 $p = position + t * direction$
  - ▶ Берём плотность из текстуры в текущей точке  $p$
  - ▶ Обновляем optical depth:  
 $optical\_depth += absorption * density * dt$
- ▶ Вычисляем opacity как в задании 2



## Задание 5

Вычисляем рассеяние (во фрагментном шейдере), считаем что фазовая функция не зависит от угла рассеяния (тогда  $f(p, \theta) = \frac{1}{4\pi}$ )

- ▶ Коэффициент поглощения можно сделать поменьше (или даже нулём)
- ▶ Заводим коэффициенты рассеяния `scattering = 4.0` и вымирания `extinction = absorption + scattering`
- ▶ Заводим интенсивность света `light_color = vec3(16.0)`
- ▶ Инициализируем рассеянный свет `color = vec3(0.0)`
- ▶ В цикле аккумулируем и `optical depth`, и рассеянный свет
- ▶ `optical_depth += extinction * density * dt`
- ▶ Для рассеяния нужно посчитать `light_optical_depth` аналогичным вложенным циклом (число итераций может быть другое, например 16) вдоль луча из текущей точки в направлении света `light_direction`
  - ▶ Придётся вызвать `intersect_bbox` на каждую итерацию внешнего цикла
  - ▶ Придётся читать из текстуры на каждую итерацию внутреннего цикла
- ▶ Обновляем рассеянный свет как
$$\text{color} += \text{light\_color} * \exp(-\text{light\_optical\_depth}) * \exp(-\text{optical\_depth}) * dt * \text{density} * \text{scattering} / 4.0 / \text{PI}$$
- ▶ В качестве результата шейдера выводим `vec4(color, alpha)`



## Задание 6\*

Разные коэффициенты рассеяния для разных цветов

- ▶ Обычный блендинг не умеет делать альфа-канал для каждого цвета по отдельности, так что заменим цвет фона (`glClearColor`) на чёрный
- ▶ Коэффициенты `absorption`, `scattering`, `extinction`, а также величины `optical_depth` и `light_optical_depth` должны стать `vec3`
- ▶ В координаты `scattering` нужно записать три разных числа (подберите что-нибудь сами в районе 1..10)
- ▶ Поиграйтесь со значением `scattering`, чтобы посмотреть, как оно влияет на результат

