Компьютерная графика

Практика 3: VAO, VBO, кривые Безье

2021

Кривые Безье

- Используются для генерации плавных кривых
- Строются по набору точек $p_0, p_1, \ldots, p_n$
- Кривая Безье с параметром $t \in [0, 1]$ определяется как аффинная комбинация
  $b(t) = \sum_{k=0}^{n} b_{k,n}(t) \cdot p_k$
- Коэффициенты – полиномы Бернштейна:
  $b_{k,n}(t) = \binom{n}{k} x^k (1 - x)^{n-k}$

Кривые Безье

- Используются для генерации плавных кривых
- Строются по набору точек $p_0, p_1, \ldots, p_n$
- Кривая Безье с параметром $t \in [0, 1]$ определяется как аффинная комбинация
  $b(t) = \sum_{k=0}^{n} b_{k,n}(t) \cdot p_k$
- Коэффициенты – полиномы Бернштейна:
  $b_{k,n}(t) = \binom{n}{k} x^k (1 - x)^{n-k}$
- Кривая первого порядка ($n = 1$): отрезок $p_0 p_1$
  $b(t) = (1 - t) \cdot p_0 + t \cdot p_1$

Кривые Безье

- Используются для генерации плавных кривых
- Строются по набору точек $p_0, p_1, \ldots, p_n$
- Кривая Безье с параметром $t \in [0, 1]$ определяется как аффинная комбинация
  $b(t) = \sum_{k=0}^{n} b_{k,n}(t) \cdot p_k$
- Коэффициенты – полиномы Бернштейна:
  $b_{k,n}(t) = \binom{n}{k} x^k (1-x)^{n-k}$
- Кривая первого порядка $(n = 1)$: отрезок $p_0 p_1$
  $b(t) = (1-t) \cdot p_0 + t \cdot p_1$
- Кривая второго порядка $(n = 2)$:
  $b(t) = (1-t)^2 \cdot p_0 + 2t(1-t) \cdot p_1 + t^2 \cdot p_2$

Задание 1

Загружаем данные в VBO

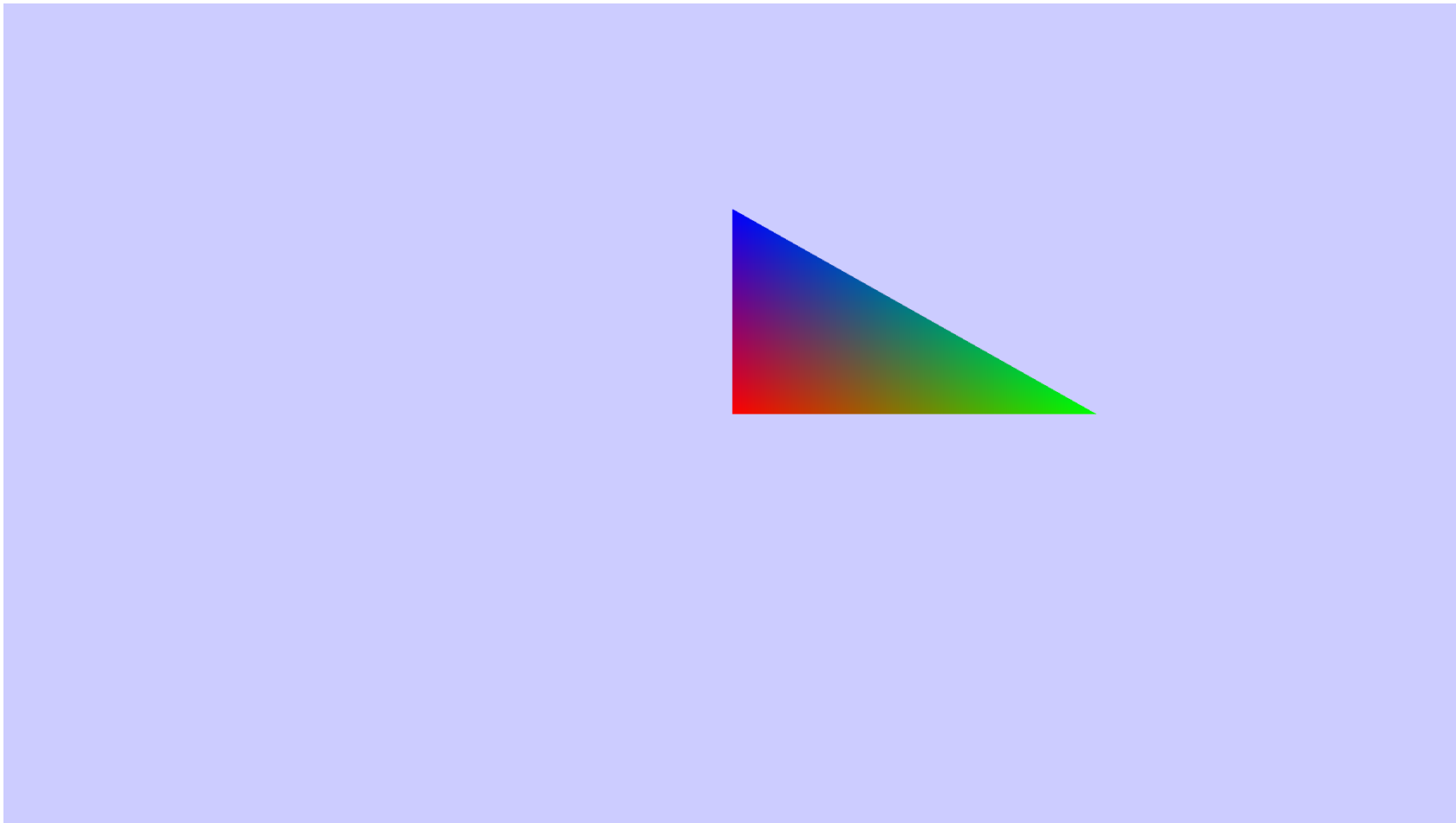
- Создайте и заполните массив из трёх вершин типа `vertex`
- Создайте VBO и загрузите в него данные: `glGenBuffers`, `glBindBuffer`, `glBufferData`
- Проверьте, что данные загрузились, создав временную переменную, и считав в неё координату какой-нибудь вершины (`glGetBufferSubData`) и выведя результат в `std::cout`

Задание 2

Рисуем с помощью VAO

- Создайте VAO и настройте атрибуты вершин: `glGenVertexArrays`, `glBindVertexArray`, `glEnableVertexAttribArray`, `glVertexAttribPointer`
- Нарисуйте треугольник с помощью этого VAO: `glDrawArrays`

Задание 2



Задание 3

Переходим к оконным координатам

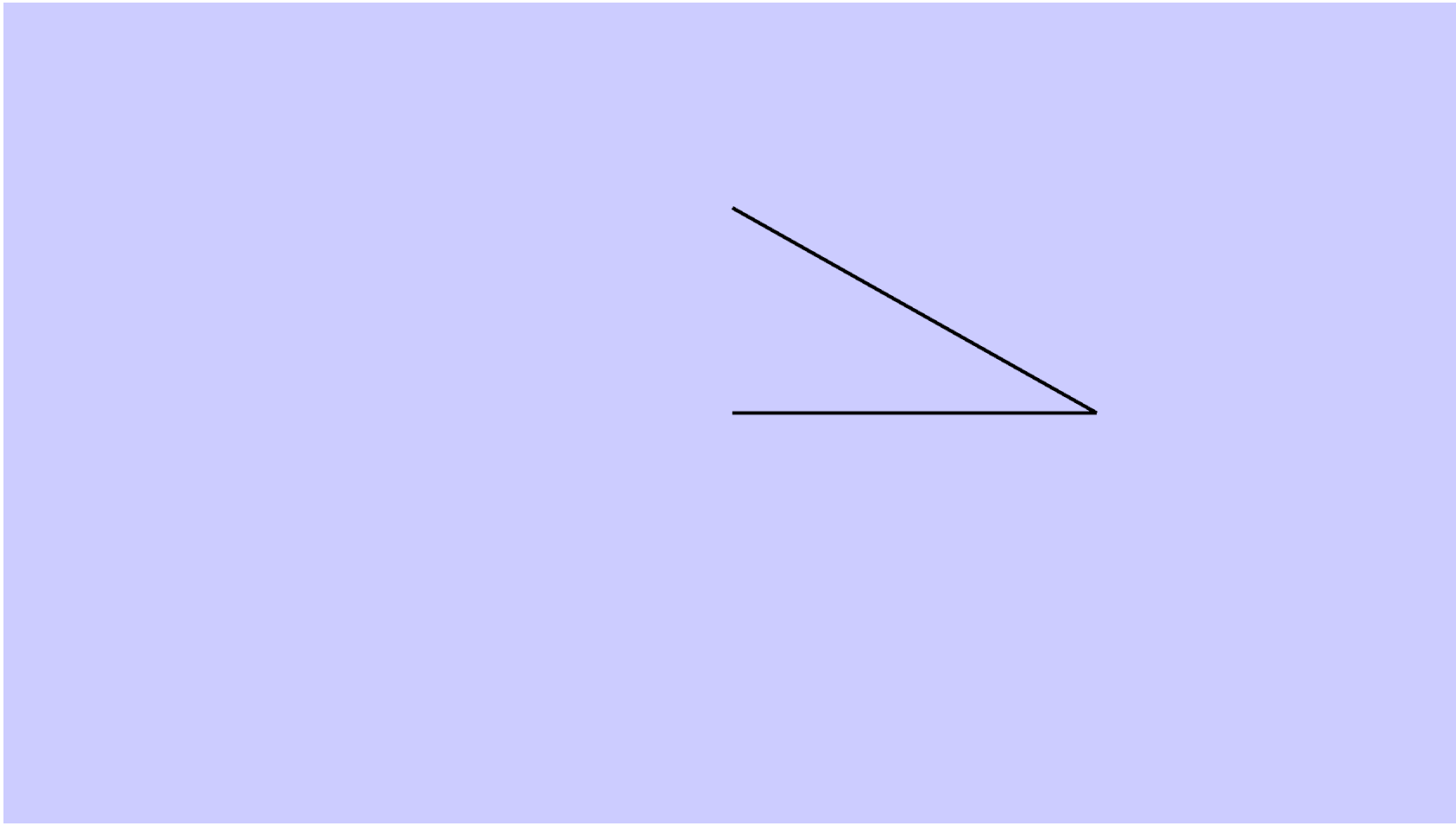
- Заполните view-матрицу преобразованием, которое переводит X из `[0, width]` в `[-1, 1]`, и Y из `[height, 0]` в `[-1, 1]`
- Измените координаты треугольника, чтобы он был заметен на экране

Задание 4

Динамически добавляем/удаляем точки мышкой

- Замените статический массив с вершинами на контейнер `std::vector` (изначально пустой)
- При нажатии левой кнопки мыши (`SDL_BUTTON_LEFT`) добавьте новую вершину в контейнер с координатами мыши (цвета выбирайте как угодно)
- При нажатии правой кнопки мыши (`SDL_BUTTON_RIGHT`) удалите последнюю вершину из контейнера, если он не пустой
- Если контейнер с вершинами изменился, обновите данные в соответствующем VBO
- Рисуем линию из всех точек: `GL_LINE_STRIP`
- Делаем линию потолще: `glLineWidth(5.f)`

Задание 4

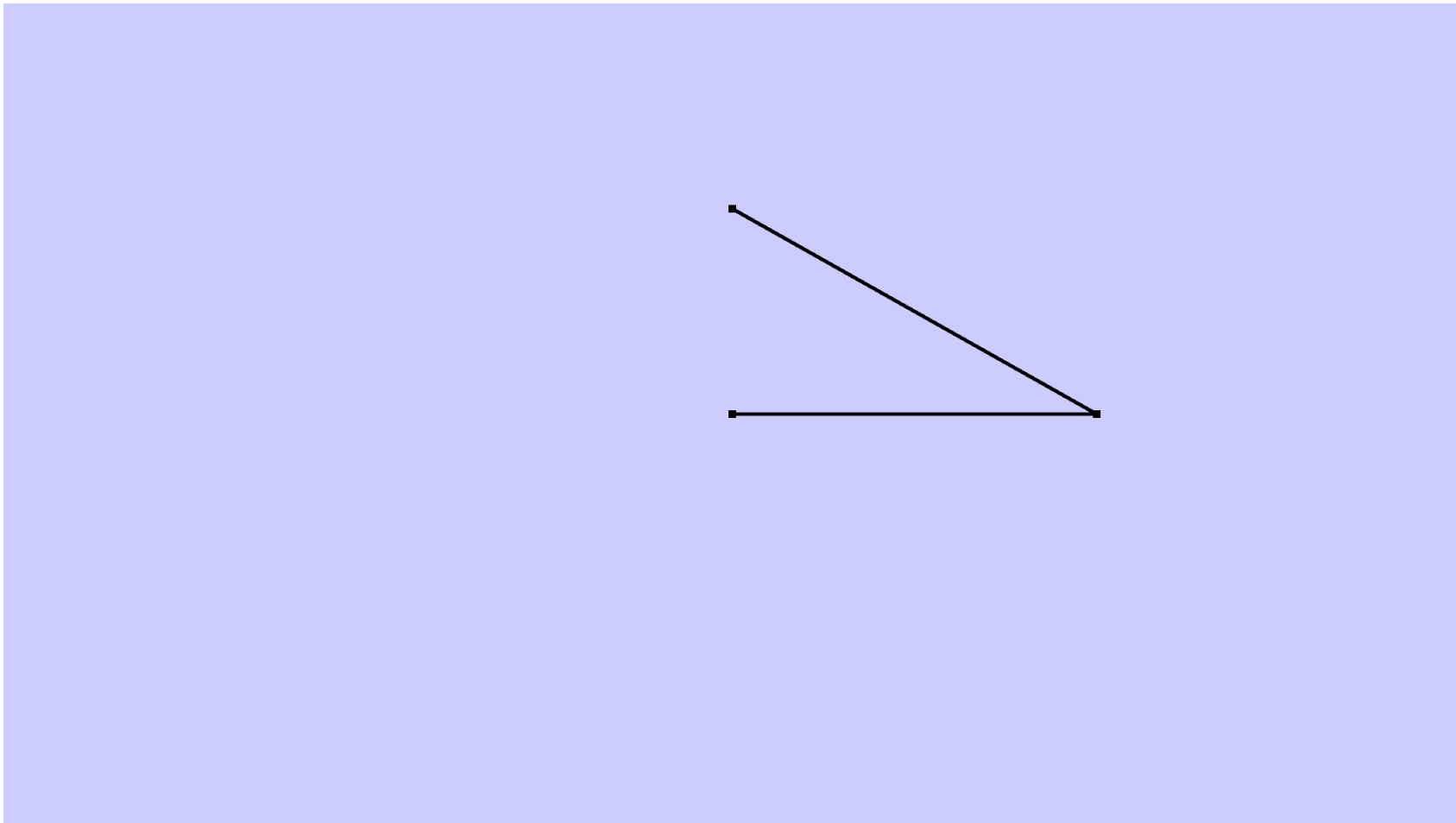


Задание 5

Нарисуем сами точки

- `glPointSize(10)` чтобы точки были заметны
- Ещё один вызов `glDrawArrays` чтобы нарисовать точки (`GL_POINTS`)

Задание 5

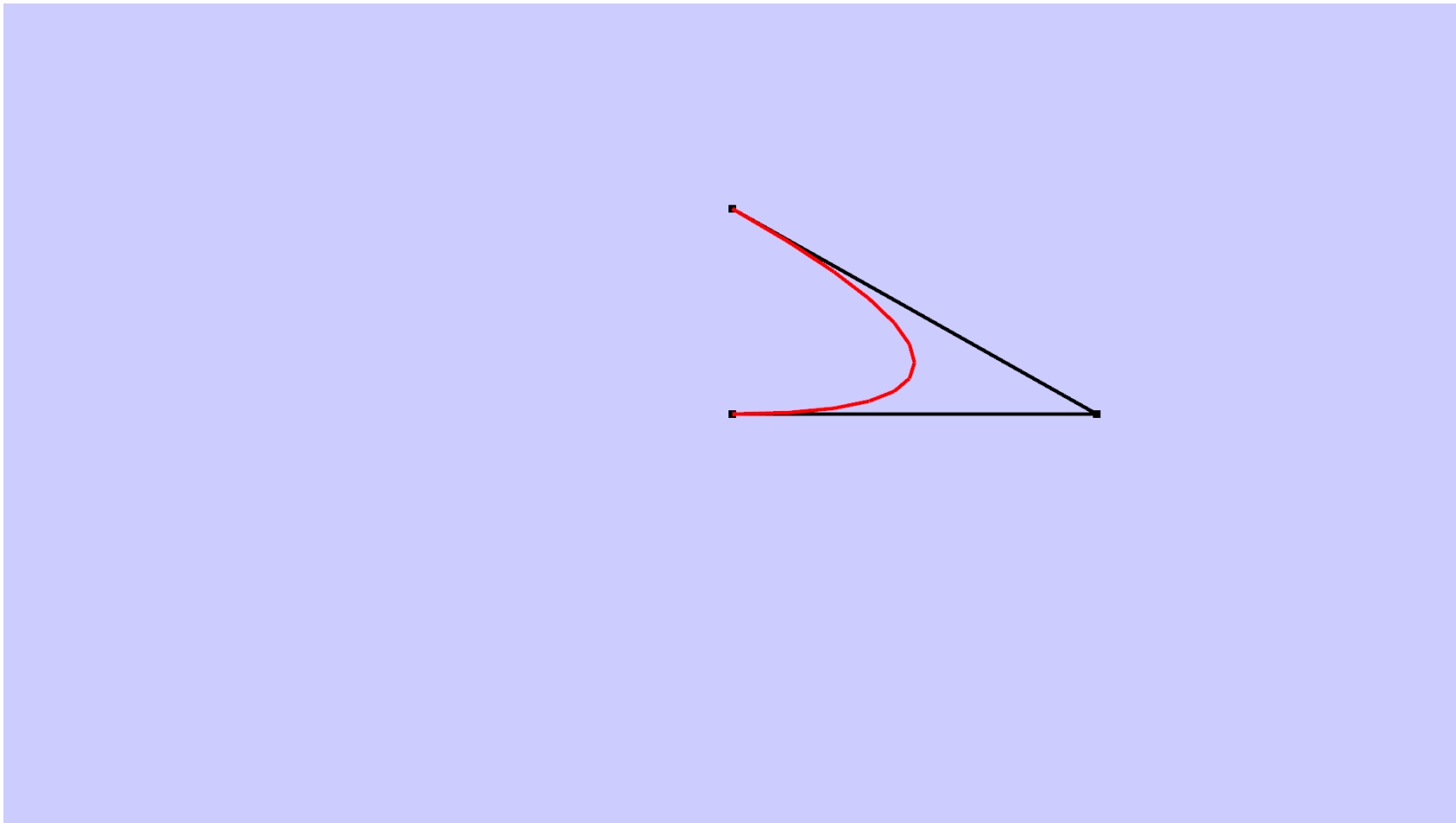


Задание 6

Генерируем и рисуем кривые Безье

- Создаёте ещё один VBO, `std::vector` и VAO для точек кривой Безье (не забудьте настроить атрибуты в VAO)
- Заведите параметр, управляющий уровнем детализации кривой: `int quality = 4;`
- При добавлении/удалении точек пересчитываем заново всю кривую Безье
- Если в исходной ломаной `N` отрезков, то в кривой Безье должно быть `N * quality` отрезков
- Параметр `t` должен равномерно меняться от 0 до 1 вдоль всей кривой
- Цвет - любой, но отличающийся от цвета исходной ломаной
- Данные в VBO должны обновляться только при их изменении
- Рисуем кривую, используя ту же шейдерную программу и новый VAO

Задание 6

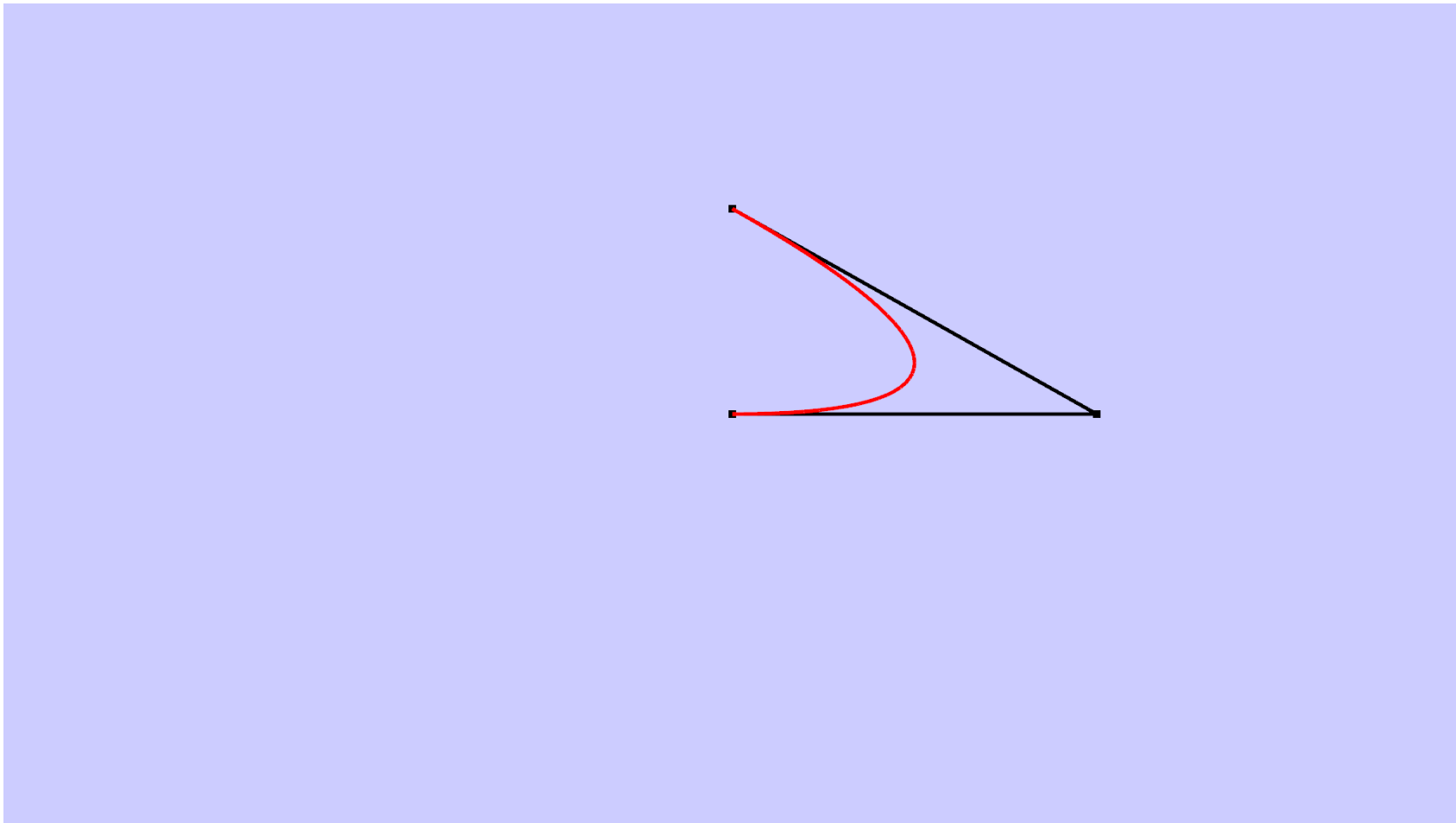


Задание 7

Динамически меняем уровень детализации

- При нажатии на стрелку влево (`SDL_LEFT`) уровень детализации `quality` должен уменьшаться (но не может быть меньше 1)
- При нажатии на стрелку вправо (`SDL_RIGHT`) уровень детализации `quality` должен увеличиваться

Задание 7



Задание 8*

Добавляем ползающий пунктир к кривой Безье

- Пунктира можно добиться, не рисуя половину пикселей кривой в зависимости от расстояния до начала
- Например, пиксели кривой на расстоянии `[0, 20]` от начала рисуются, на расстоянии `[20, 40]` от начала не рисуются, `[40, 60]` рисуются, и т.д.
- Для этого нужно добавить в вершины кривой расстояние вдоль кривой от её начала
- Расстояние до первой вершины = 0
- Расстояние до вершины `i` = расстояние до вершины `(i-1)` + расстояние от вершины `(i-1)` до вершины `i`
- Для расстояния между вершинами пригодится функция C++ `std::hypot`
- Расстояние нужно интерполировать между вершинами и передать во фрагментный шейдер
- Чтобы понять, нужно ли рисовать пиксель, пригодится функция GLSL `mod` (например, `mod(distance, 40.0) < 20.0`)
- Чтобы отменить рисование пикселя, нужно в функции `main` фрагментного шейдера вызвать команду `discard;`
- Само расстояние нужно добавить как поле вершины и как атрибут в VAO и вершинный шейдер
- Чтобы пунктир двигался, можно прибавить к расстоянию что-то, зависящее от времени (придётся передать время в шейдер как `uniform`)
- Два варианта организации шейдеров:
  - Два отдельных шейдера: один для ломаной, один для пунктирной кривой Безье
  - Один общий шейдер: `uniform`-настройка, нужно ли рисовать пунктир (например, `uniform int dash;`, `dash == 1` означает рисовать пунктир)

Задание 8