

# Компьютерная графика

Домашнее задание 1: График функции на плоскости с  
изолиниями

---

2025

- Функция на плоскости, зависящая от времени  $f(x, y, t)$

- Функция на плоскости, зависящая от времени  $f(x, y, t)$
- Фиксированный прямоугольник  $[x_0, x_1] \times [y_0, y_1]$ , разбитый на сетку из  $W \times H$  прямоугольных ячеек

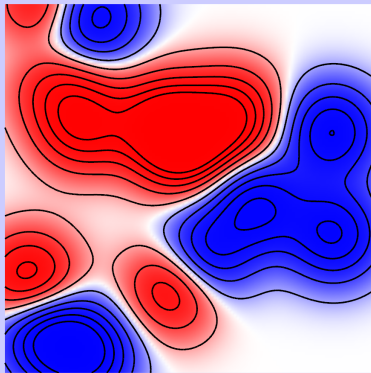
- Функция на плоскости, зависящая от времени  $f(x, y, t)$
- Фиксированный прямоугольник  $[x_0, x_1] \times [y_0, y_1]$ , разбитый на сетку из  $W \times H$  прямоугольных ячеек
- Нужно нарисовать:
  - 'График' функции цветом, используя вершины сетки как основу

# Задание

- Функция на плоскости, зависящая от времени  $f(x, y, t)$
- Фиксированный прямоугольник  $[x_0, x_1] \times [y_0, y_1]$ , разбитый на сетку из  $W \times H$  прямоугольных ячеек
- Нужно нарисовать:
  - 'График' функции цветом, используя вершины сетки как основу
  - Изолинии – линии  $f(x, y, t) = \text{const}$  поверх графика

# Задание

- Функция на плоскости, зависящая от времени  $f(x, y, t)$
- Фиксированный прямоугольник  $[x_0, x_1] \times [y_0, y_1]$ , разбитый на сетку из  $W \times H$  прямоугольных ячеек
- Нужно нарисовать:
  - 'График' функции цветом, используя вершины сетки как основу
  - Изолинии – линии  $f(x, y, t) = \text{const}$  поверх графика
- График и изолинии вычисляются заново на каждый кадр



Любая интересная меняющаяся во времени функция на плоскости

- Metaballs:  $f(x, y, t) = \sum c_i \exp\left(-\frac{(x-x_i)^2 + (y-y_i)^2}{r_i^2}\right)$ , где  $(x_i, y_i)$  – координаты движущейся по какому-то закону точки



Любая интересная меняющаяся во времени функция на плоскости

- Metaballs:  $f(x, y, t) = \sum c_i \exp\left(-\frac{(x-x_i)^2 + (y-y_i)^2}{r_i^2}\right)$ , где  $(x_i, y_i)$  – координаты движущейся по какому-то закону точки
- Шум Перлина: строится на основе сетки двумерных единичных векторов, которые можно крутить в зависимости от времени (эта сетка никак не связана с сеткой использующейся для рендеринга)

Любая интересная меняющаяся во времени функция на плоскости

- Metaballs:  $f(x, y, t) = \sum c_i \exp\left(-\frac{(x-x_i)^2 + (y-y_i)^2}{r_i^2}\right)$ , где  $(x_i, y_i)$  – координаты движущейся по какому-то закону точки
- Шум Перлина: строится на основе сетки двумерных единичных векторов, которые можно крутить в зависимости от времени (эта сетка никак не связана с сеткой использующейся для рендеринга)
- Комбинация синусов/косинусов с разными амплитудами и фазами

Любая интересная меняющаяся во времени функция на плоскости

- Metaballs:  $f(x, y, t) = \sum c_i \exp\left(-\frac{(x-x_i)^2 + (y-y_i)^2}{r_i^2}\right)$ , где  $(x_i, y_i)$  – координаты движущейся по какому-то закону точки
- Шум Перлина: строится на основе сетки двумерных единичных векторов, которые можно крутить в зависимости от времени (эта сетка никак не связана с сеткой использующейся для рендеринга)
- Комбинация синусов/косинусов с разными амплитудами и фазами
- etc.

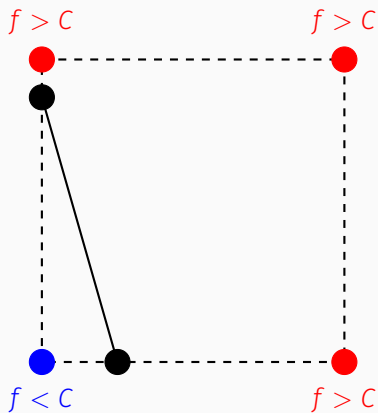
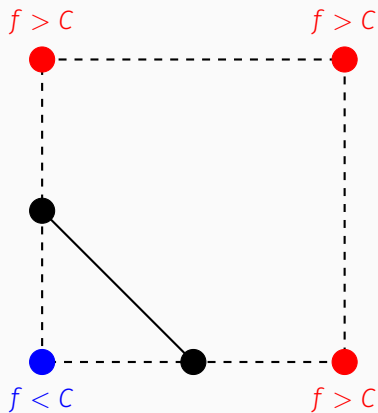
- Вершина сетки –  $(x_i, y_j)$  + цвет
- Раскрасить в зависимости от значения функции

- Вершина сетки –  $(x_i, y_j)$  + цвет
- Раскрасить в зависимости от значения функции
- Прямоугольники сетки придётся разбить на пары треугольников

- Линии  $z = f(x, y, t) = C_i$  для некоторых значений  $C_i$
- Набор значений  $C_1, C_2, C_3, \dots, C_k$  выбрать на основе вашей функции

- Линии  $z = f(x, y, t) = C_i$  для некоторых значений  $C_i$
- Набор значений  $C_1, C_2, C_3, \dots, C_k$  выбрать на основе вашей функции
- Строить алгоритмом *marching squares* с линейной интерполяцией (или аналогичным алгоритмом на треугольниках) на основе той же сетки, что и график

# Marching squares





# Marching squares

- Есть вариант алгоритма, соединяющий центры рёбер, – его **не нужно использовать**
- Есть вариант алгоритма, линейно интерполирующий значение функции вдоль ребра чтобы найти точку  $f = C$  – **нужно использовать именно его**

# Marching squares

- Есть вариант алгоритма, соединяющий центры рёбер, – его **не нужно использовать**
- Есть вариант алгоритма, линейно интерполирующий значение функции вдоль ребра чтобы найти точку  $f = C$  – **нужно использовать именно его**
- Есть вариант алгоритма (иногда называется marching triangles), использующий треугольники и избавляющийся от неоднозначностей алгоритма на квадратах, – **можно** использовать его

# Marching squares

- Есть вариант алгоритма, соединяющий центры рёбер, – его **не нужно использовать**
- Есть вариант алгоритма, линейно интерполирующий значение функции вдоль ребра чтобы найти точку  $f = C$  – **нужно использовать именно его**
- Есть вариант алгоритма (иногда называется marching triangles), использующий треугольники и избавляющийся от неоднозначностей алгоритма на квадратах, – **можно** использовать его
- Ещё marching triangles называют трёхмерный алгоритм построения модели по облаку точек – это **не то, что нам нужно**

## Немного деталей

- Часть данных вершин будут обновляться каждый кадр – цвета точек графика, координаты изолиний
- Часть данных постоянна – XY-координаты вершин сетки – их имеет смысл хранить в отдельном VBO
- Как для графика, так и для изолиний имеет смысл использовать индексированный рендеринг
- Можно использовать простые `GL_LINES` и `GL_TRIANGLES`, а можно `GL_LINE_STRIP/GL_LINE_LOOP` и `GL_TRIANGLE_STRIP` вместе с `primitive restart`

- Чтобы избавиться от дублирования вершин в изолиниях, придётся для каждого ребра исходной сетки запомнить индекс вершины изолинии, лежащей на этом ребре: можно использовать, например, `std::unordered_map` или просто `std::vector`, придумав какую-то нумерацию для рёбер

- Обратите особое внимание на работу с VAO и буферами:
  - Создание VAO и буферов должно быть **только** в начале программы
  - Настройка атрибутов вершин (т.е. настройка VAO) должна быть **один раз** в начале программы
  - Данные в буферы должны загружаться **только** при их реальном изменении

- Обратите особое внимание на работу с VAO и буферами:
  - Создание VAO и буферов должно быть **только** в начале программы
  - Настройка атрибутов вершин (т.е. настройка VAO) должна быть **один раз** в начале программы
  - Данные в буферы должны загружаться **только** при их реальном изменении
- В идеале у вас должно быть 2 *draw call* (вызова функции `glDraw*`) на кадр: один на график, один на изолинии

# Баллы

- **3 балла:** рисуется динамический график функции (т.е. цвета вершин меняются во времени) за один draw call
- **3 балла:** рисуются динамические изолинии за один draw call
- **2 балла:** все данные в VBO обновляются только при их изменении, статические данные хранятся в отдельных VBO
- **2 балла:** используется индексированный рендеринг для графика и его вершины не дублируются
- **2 балла:** используется индексированный рендеринг для изолиний и их вершины не дублируются
- **1 балл:** можно динамически менять количество изолиний
- **1 балл:** можно динамически менять детализацию сетки
- **1 балл:** корректно обрабатывается *aspect ratio*, т.е. соотношение ширины к высоте видимого графика не меняется при изменении размеров окна (свободное пространство окна можно оставить пустым), и график всегда полностью влезает в окно

Всего: **15 баллов**

Защита заданий на практике **через 2 недели.**



- [jamie-wong.com/2014/08/19/metaballs-and-marching-squares](http://jamie-wong.com/2014/08/19/metaballs-and-marching-squares)
- [jacobzelko.com/marching-squares](http://jacobzelko.com/marching-squares)
- [ckcollab.com/2020/11/08/Marching-Squares-Algorithm.html](http://ckcollab.com/2020/11/08/Marching-Squares-Algorithm.html)