Эргономика компьютерных игр

презентация по курсу “Человеко-компьютерное взаимодействие”

В.Э. Смирнов, Д.В. Сухоловский

ЮФУ, каф. МОП ЭВМ

2017 год

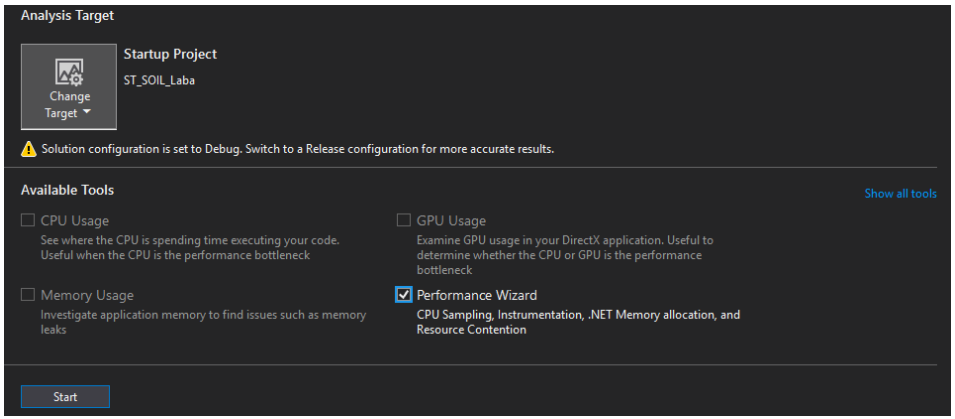
Содержание

Качество ПО

Улучшение

- Качество программ – понятие, аккумулирующее противоречия природы самих программ и противоречия, возникающие при их создании
- Профилирование – сбор характеристик работы программы
- Оптимизация – модификация программы для улучшения её эффективности

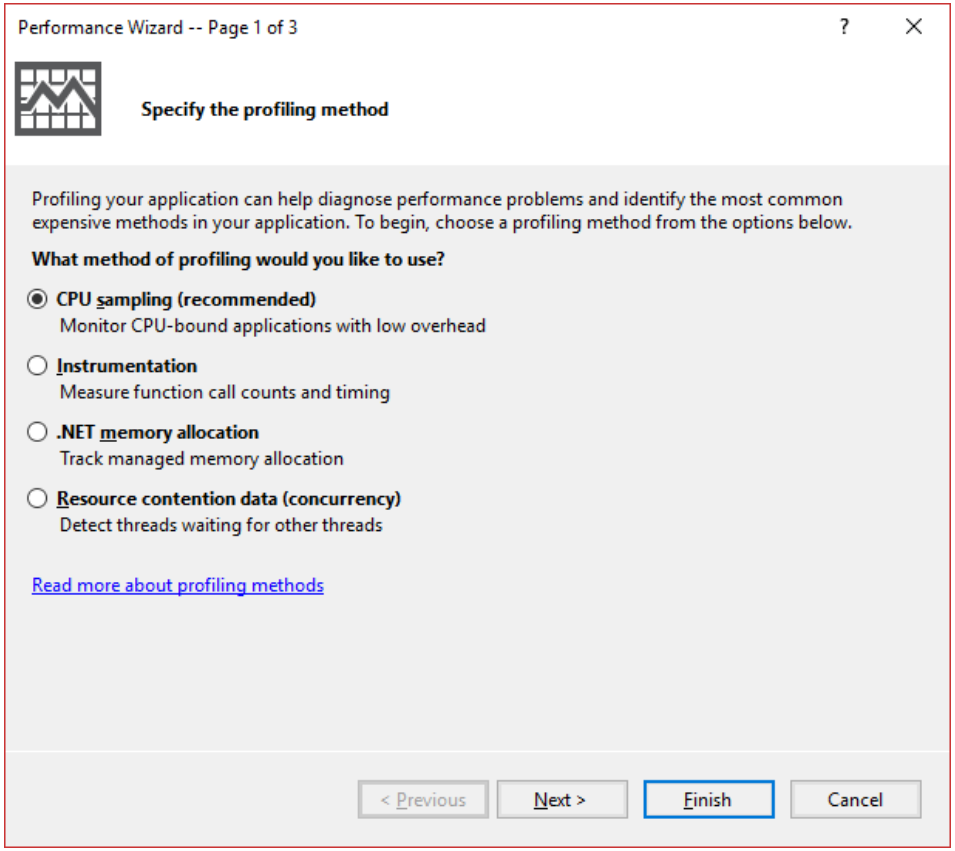
Профилирование в Visual Studio



Режимы работы профайлера

- Visual Studio имеет возможности для профилирования:
  - Использования процессора (CPU Usage)
  - Использования видеокарты (GPU Usage)
  - Использования памяти (Memory Usage)
  - Общей производительности (Performance Wizard)

Performance Wizard



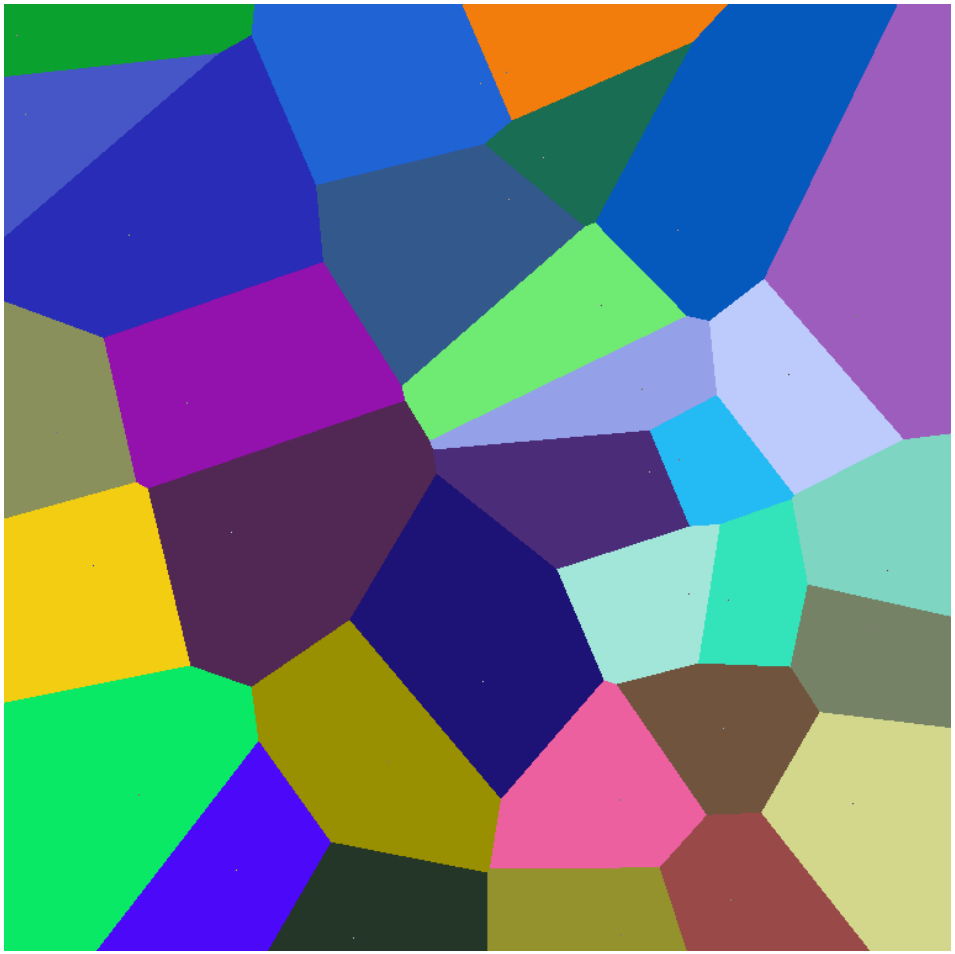
Параметры Performance Wizard

- При выборе Performance Wizard можно дополнительно выбрать параметры для профилирования
- Из предложенных далее будет использоваться только CPU sampling

Исходная программа

- Программа генерирует изображение, на котором изображено
  - N точек разного цвета
- Положение и цвет точек может задаваться
  - Пользователем
  - Случайным образом
- Остальные области изображение закрашиваются цветом, противоположным той точке, к которой они ближе всего находятся

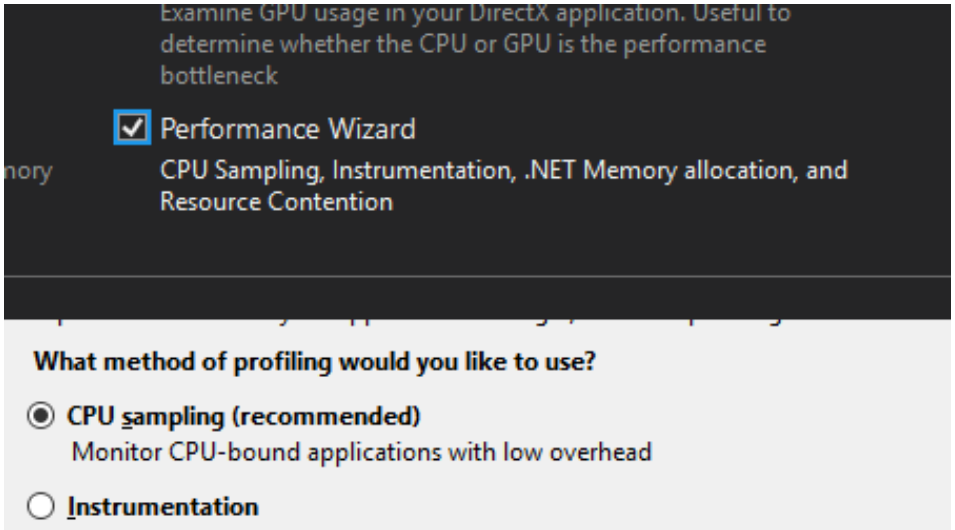
Диаграмма Вороного



Результат работы программы

- Такое разбиение плоскости (в данном случае изображения) называется диаграммой Вороного
- Результатом работы программы является изображение, которое разделено на 32 области

Первичная оценка программы



Выбор режима профилирования

- Тля того, чтобы знать откуда начать и куда двигаться, стоит сначала сделать первичную оценку того состояния, в котором находится программу
- Для этого можно запустить анализ в Performance Wizard

Результаты первичного анализа

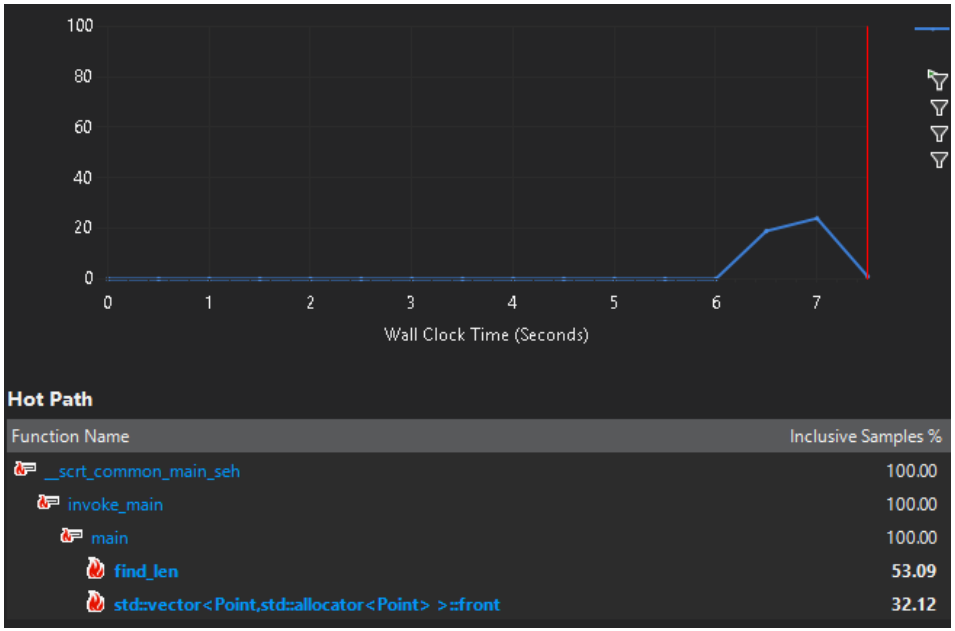


График выполнения программы

- По результатам анализа видно, что основное процессорное время берут
  - vector::begin() – 32.1%
  - find_len() – 53.1%

Результаты первичного анализа. Взгляд на код

```
        for(int i = 0, h = img.height(); i < h; ++i)
        {
            for(int j = 0, w = img.width(); j < w; ++j)
            {
15.5%           Point* v_begin = &v_points.front();
16.9%           Point* v_end = &v_points.front() + n_points;
                Point* p = v_begin;
 1.9%           int min_len = find_len(p->x, p->y, j, i);

 1.2%           for(Point* it = v_begin; it != v_end; ++it)
                {
52.8%               int cur_len = find_len(it->x, it->y, j, i);
                    if(cur_len < min_len)
                    {
 0.4%                   p = it;
                        min_len = cur_len;
                    }
 4.3%           }
 6.4%           img.set_pixel_rgb(j, i,
                                  (uint8_t)( 0xFF - p->r ),
                                  (uint8_t)( 0xFF - p->g ),
                                  (uint8_t)( 0xFF - p->b ) );
            }
        }
```

Исходный код места проблемы

- Тот факт, что метод vector::begin() брал на себя столько процессорного времени был неожиданностью
- Функция find_len() отбирает наибольшую часть процессорного времени, поэтому ее тоже стоит рассмотреть

Улучшение 1

```
        for(int i = 0, h = img.height(); i < h; ++i)
        {
            for(int j = 0, w = img.width(); j < w; ++j)
            {
 7.2%           Point* v_begin = v_points.data();
 0.3%           Point* v_end = v_begin + n_points;
                Point* p = v_begin;
 1.2%           int min_len = find_len(p->x, p->y, j, i);

 0.6%           for(Point* it = v_begin; it != v_end; ++it)
                {
57.5%               int cur_len = find_len(it->x, it->y, j, i);
 0.2%               if(cur_len < min_len)
                    {
 0.6%                   p = it;
< 0.1 %                min_len = cur_len;
                    }
 3.6%           }
 7.1%           img.set_pixel_rgb(j, i,
                                  (uint8_t)( 0xFF - p->r ),
                                  (uint8_t)( 0xFF - p->g ),
                                  (uint8_t)( 0xFF - p->b ) );
```

Улучшение

- Замена метода vector::begin() на vector::data()
- Переиспользование ранее определенных переменных вместо вызова метода повторно

Проблемы и варианты решений

- Неоптимальный алгоритм
  - Изменить алгоритм построения диаграммы Вороного
- Чрезмерное использование функции find_len()
  - Заранее просчитать все возможные значения функции, а в дальнейшем обращаться лишь к двумерному массиву
- Однопоточная программа
  - Распараллелить алгоритм на все ядра процессора

Кэширование результатов функции find_len()

```
14.9%   Point* v_begin = v_points.data();
 0.2%   Point* v_end = v_begin + n_points;
        Point* p = v_begin;
        //int min_len = find_len(p->x, p->y, j, i);
 0.2%   int a = abs(p->x - j);
 0.2%   int b = abs(p->y - i);
        if(a < b)
        {
            int c = a;
            a = b;
            b = a;
        }

 0.8%   int min_len = find_len_computed_p[a][b];

 2.4%   for(Point* it = v_begin; it != v_end; ++it)
        {
 6.6%       int a = abs(it->x - j);
 7.5%       int b = abs(it->y - i);
 1.7%       if(a < b)
            {
 0.2%           int c = a;
 0.8%           a = b;
 0.2%           b = a;
            }

            //int cur_len = find_len(it->x, it->y, j,
18.5%       int cur_len = find_len_computed_p[a][b];
 1.1%       if(cur_len < min_len)
```

Код рассматриваемого места

- Дальнейшее улучшение программы будет двигаться в сторону кэширования из за того что
  - Функция find_len() находится внутри 3-х уровневого цикла
  - Очень много вызовов функции с одними и теми же параметрами
  - Другие алгоритмы сложны в реализации

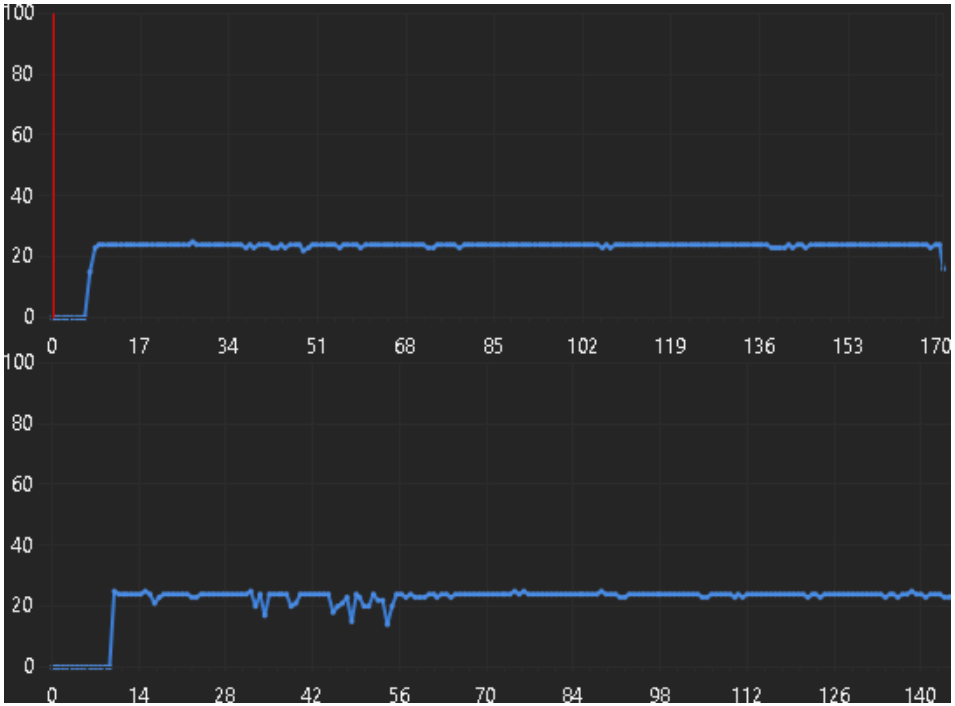
Улучшение 2

```
14.9%   Point* v_begin = v_points.data();
 0.2%   Point* v_end = v_begin + n_points;
        Point* p = v_begin;
        //int min_len = find_len(p->x, p->y, j, i);
 0.2%   int a = abs(p->x - j);
 0.2%   int b = abs(p->y - i);
        if(a < b)
        {
            int c = a;
            a = b;
            b = a;
        }

 0.8%   int min_len = find_len_computed_p[a][b];

 2.4%   for(Point* it = v_begin; it != v_end; ++it)
        {
 6.6%       int a = abs(it->x - j);
 7.5%       int b = abs(it->y - i);
 1.7%       if(a < b)
            {
 0.2%           int c = a;
 0.8%           a = b;
 0.2%           b = a;
            }

            //int cur_len = find_len(it->x, it->y, j,
18.5%       int cur_len = find_len_computed_p[a][b];
 1.1%       if(cur_len < min_len)
```

Улучшенный код

- Перед выполнением 3-х уровневого цикла были
  - Закэшированы все возможные значения функции find_len()
  - Добавлены оптимизации по уменьшению использования памяти
- Внутри цикла
  - Обращение к функции заменено на обращение к массиву

Анализ изменений



Графики выполнения. До и после

- С одинаковыми параметрами на входе программы она отработала
  - На 28 секунд быстрее
  - Примерно на 15% быстрее памяти
- При определенных условиях относительный прирост может быть больше

Заключение

- Visual Studio представляет собой мощное средство для разработки ПО которое включает в себя очень много различных инструментов
- Используя возможности профилирования, в программе были найдены места, нуждающиеся в оптимизации, которые были улучшены
- Дополнительно к проделанной работе можно распараллелить программу, что даст значительный прирост, если компьютер имеет больше одного ядра процессора
- Проделанные улучшения не являются единственно верными. Кроме них можно провести так же другие улучшения, которое, возможно, приведут к лучшему результату