Patryk Kozioł

221610

Wybrane aspekty optymalizacji w grach

Optymalizacja aplikacji Renderer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Opis funkcji przed optymalizacją:** | **Średni czas przed optymalizacją w ms – 20 prób.** | **Opis funkcji po optymalizacji:** | **Średni czas po optymalizacji w ms – 20 prób.** | **Procent zysku czasowego po optymalizacji:** | **Użyty obiekt do testowania:** |
| Własne metody getMax i getMin w klasie Triangle. | 11182,5 | Wbydowane funkcje Math.max i Math.min | 6291 | 43,74 | czajnik.obj |
| Użycie listy do przechowywania obiektów w klasie Painting. | 7261 | Użycie tablicy do przechowywania obiektów w klasie Painting. | 6604 | 9,05 | czajnik.obj |
| Użycie iteratora i++ w pętlach obsługujących tablice w klasie Painting. | 6724 | Użycie iteratora ++i w pętlach obsługujących tablice w klasie Painting. | 6590,6 | 3,51 | czajnik.obj |
| Użycie zagnieżdżonych pętli w klasie Painting. | 186,2 | Użycie jednej spłaszczonej pętli w klasie Painting. | 158,2 | 15,04 | cube.obj |

Etapy optymalizacji:

Do wstępnego sprawdzenia czasu działania i ,,wąskich gardeł ‘’ aplikacji użyłem profilera. Profiler jednoznacznie wykazał, największą ilość wywołań i ogólny czas działania funkcji getMin i getMax. Powyższe funkcje okazały się wysoce niewydajne i powolne w działaniu.

Następnym newralgicznym punktem, który został poddany optymalizacji była klasa Painting. Początkowo do przechowywania zbioru obiektów przeznaczonych do rasteryzacji służyła Lista. Lista ta była przekazywana do klasy Painting, która iterowała po tej liście pobierając obiekty Triangle. W celu przyśpieszenia pobierania obiektów Lista została zmieniona na tablice. Powyższa zmiana była zauważalna chociaż nie tak znaczna jak pierwszy etap optymalizacji.

Kolejną próbą przyśpieszenia była zmiana metody inkrementacji z   
postkremenetacji (i++) na preinkrementacje (++i). Powyższa modyfikacja dała nie wielkie rezultaty, lecz była istotnym etapem do dalszych zmian.

Ostatnim etapem poprawy wydajności aplikacji była zmiana dwóch pętli w klasie Painting, odpowiadających za pozycjonowanie wyświetlanego obrazu w osiach X i Y. Zmiana polegała na usunięciu dwóch pętli i na ich miejsce wstawienie jednej spłaszczonej pętli obsługującej zarówno oś X i Y. Użycie jednej pętli zamiast dwóch dało drugi co do procentowego zysku rezultat optymalizacji.

Wnioski:

Użycie profilera jak narzędzie do badania wydajności aplikacji jest bardzo skutecznym i prostym sposobem na wyszukanie funkcji, które znacznie spowalniają aplikacje.  
Tworząc własną funkcję należy się zastanowić czy nie jest możliwe zastąpieniem jej wbudowaną funkcją z dostępnych bibliotek. Takie działanie może okazać się znacznie wydajniejsze i znacznie przyśpieszyć obliczenia naszej funkcji.  
Aby użyć tablic do przechowania danych, należy określić jej wielkość przy inicjalizacji. Listy możemy dowolnie wydłużać poprzez dodawanie nowych elementów. Jednak jeśli znana jest dokładna długość potrzebowanej tablicy, zastosowanie jej może w znacznym stopniu skrócić czas wykonania programu.  
Znacznym usprawnieniem okazało się zastosowanie pojedynczej pętli obsługującej pozycje tworzonych pikseli na bitmapie.