ANALISIS PERFORMA CORE FRAMEWORK IOS PADA APLIKASI VISUALISASI *GENERAL-GRAPH* MENGGUNAKAN PERANGKAT *MOBILE*

Shumaya Resty Ramadhani[1]

[1] Jurusan, Teknologi Informasi, Politeknik Caltex Riau
[1]*shumaya@pcr.ac.id*

Abstrak

Perkembangan teknologi yang pesat membawa perubahan kebiasaan pada pengguna teknologi. Perangkat teknologi ikut berevolusi, mulai dari super komputer hingga *smartphone* berukuran kecil dengan performa yang sepadan. Banyaknya penikmat teknologi yang beralih kepada piranti cerdas ini membuka peluang pengembangan aplikasi yang cukup besar. Aplikasi *mobile* harus tetap mampu bekerja secara cepat dan ringan meski dijalankan pada *smartphone* dengan tipe lama atau spesifikasi terbatas. Terutama aplikasi yang mengusung visualisasi dan animasi untuk menarik minat pengguna. Sistem operasi iOS menyediakan *CoreFramework* yang mendukung proses pembuatan objek dan animasi dalam jumlah banyak dengan cepat dan ringan. Oleh sebab itu, dibentuklah sebuah aplikasi visualisasi *general-graph* sederhana dengan implementasi *CoreFramework* guna menguji seberapa besar pengaruh *framework* tersebut terhadap kualitas aplikasi, terutama pada perangkat seri lama. Kriteria pengujian menggunakan tiga variabel dasar, yaitu waktu, alokasi *Central Processing Unit* (CPU) dan *Random Access Memory* (RAM) yang digunakan. Hasil dari pengujian menunjukkan bahwa meski *CoreFramework* menggunakan *Graphic Processing Unit* (GPU) untuk pemrosesannya, tapi setidaknya aplikasi membutuhkan minimal RAM berukuran 2GB pada perangkat *smartphone* agar responsifitas terjaga. Hal ini disebabkan karena ketika kapasitas RAM kecil, maka aplikasi akan menggunakan alokasi CPU dengan cukup signifikan agar bisa berjalan dengan baik.

Kata kunci : Mobile, iOS, smartphone, *CoreFramework*, performa, visualisasi

1. Pendahuluan

Pemanfaatan teknologi telah banyak berkembang selama satu dekade terakhir. Pada tahun 2018, penjualan *smartphone* di masyarakat kalangan menengah keatas cukup tinggi. Dari data tahun 2018 yang dikeluarkan oleh Global Web Index Millenials, sebanyak 97% dari jumlah pengguna adalah pada rentang usia 21-34 tahun dan telah memiliki ponsel pintar secara individu. 75% diantaranya juga memiliki laptop dan 38% memiliki perangkat *tablet* (GLOBALWEBINDEX, 2018). Angka kepemilikan *smartphone* ini sangat tinggi jika dibandingkan dengan perangkat lainnya.

Dengan banyaknya pemilik *smartphone*, aplikasi yang ditawarkan pun semakin beraneka ragam, mulai dari aplikasi *list* yang umum hingga yang menyuguhkan animasi visualisasi interaktif. Ironisnya, menurut data yang diperoleh oleh Quettra, terjadi penurunan angka sekitar 70% dari keseluruhan pengguna ketika berinteraksi dengan aplikasi (Chen, 2019). Hal ini disebabkan aplikasi yang tidak lagi digunakan dalam rentang tiga hari setelah aplikasi di pasang pada perangkatnya. Sehingga dari jutaan jenis aplikasi yang terdaftar pada layanan publikasi, hanya ribuan aplikasi saja yang memiliki pengguna aktif. Fenomena ini terjadi salah satunya adalah karena performa aplikasi yang dinilai kurang baik.

Berdasarkan studi, sebuah aksi yang hanya memerlukan waktu kurang dari 100 ms untuk diproses pada perangkat *mobile* tergolong cepat dan efektif. Bahkan menurut studi, sebanyak 49% dari pengguna aplikasi berbasis web tidak akan lanjut menggunakan app tersebut ketika respon gagal tampil dalam kurun waktu 10 detik (B. Gaudette, 2016). Hal itu karena ketika sebuah aksi diproses dan direspon dalam waktu lebih dari satu detik, maka fokus dari pengguna cenderung akan terbagi. Selain itu, aplikasi *mobile* yang menggunakan banyak memori juga menjadi pertimbangan penting bagi para pengguna. Sebab, masih banyak pengguna aplikasi yang menggunakan perangkat model lama untuk beraktivitas. Jika satu aplikasi membutuhkan banyak ruang pada sebuah perangkat *mobile*, maka akan memperlambat proses berjalannya aplikasi tersebut. Oleh sebab itu, performa dari sebuah aplikasi sangat penting untuk diperhatikan agar pengguna tidak mengalami *bad user experience*.

Membuat sebuah aplikasi visualisasi merupakan tantangan tersendiri jika menimbang keterbatasan dari *smartphone*. Hal utama yang perlu diperhatikan adalah terbatasnya memori pada

perangkat *mobile* serta responsifitas yang dirasakan pengguna ketika menjalankan aplikasi visualisasi seperti yang telah disebutkan di atas. *Library* yang digunakan untuk membuat aplikasi visualisasi pada sistem operasi iOS cukup beragam, seperti GraphViz Gephi, iGraph, dll. Akan tetapi, untuk menerapkan *library* tersebut membutuhkan usaha yang cukup besar ketika menggunakan data yang dinamis dan *real-time*. Proses personalisasi grafik sesuai kebutuhan pun menjadi terbatas. Berbeda halnya ketika membuat visualisasi murni dengan menggunakan bahasa pemrograman *native* dari sistem operasi tersebut, yaitu Swift.

Oleh sebab itu, penelitian ini berfokus kepada performa *framework* pada pembuatan aplikasi visualisasi *general-graph* yang dikembangkan menggunakan *CoreFramework*. *Framework* ini disediakan guna menjamin *user experience* dari aplikasi *mobile* keluarannya agar lebih baik. Diharapkan studi yang dilakukan pada penelitian dengan contoh kasus ini bisa mendapatkan nilai sebagai tolak ukur awal pada pengembangan aplikasi visualisasi berbasis iOS yang responsif dan hemat memori.

2. Metode Penelitian

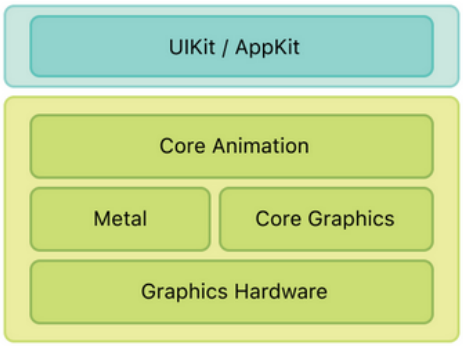
Penelitian ini dilakukan dengan melalui beberapa tahap, yaitu melakukan studi literatur, pengenalan framework yang digunakan, penentuan variabel uji pada penelitian, dan pemilihan perangkat uji.

2.1 Studi Literatur

Penelitian yang dilakukan oleh Cao Gao, dkk tentang *"A Study of Mobile Device Utilization*" mengungkapkan bahwa aplikasi *mobile* terbaru masih belum mampu untuk memanfaatkan spesifikasi besar yang disediakan oleh *smartphone* (C. GAO, 2015). Sehingga aplikasi *mobile* yang mampu memanfaatkan sumber daya perangkat secara efektif sangat diperlukan. Willocx, dkk melakukan analisis kuantitatif pada aplikasi Android dan iOS dengan menggunakan *cross-platform tools* pada perangkat jenis lama dan baru (M. WILLOCX, 2015). Ditemukan bahwa kuantitas performa tidak hanya bergantung pada jenis perangkat, tetapi juga pilihan arsitektur dan komponen yang digunakan pada sistem. Berdasarkan studi lanjutan oleh Willocx, dkk pada studi berjudul *"Comparing Performance Parameters of Mobile App Development Strategies"* tahun 2016, aplikasi yang dikembangkan secara *native* ternyata lebih hemat dari sisi penggunaan RAM nya dibandingkan pada aplikasi *cross-platform* (M. Willocx, 2016). Penelitian lainnya tentang *cross-platform* juga dimuat pada jurnal berjudul *"Performance Analysis of Native and Cross-Platform Mobile Applications"* pada tahun 2017 (GRZMIL, 2017). Aplikasi *native* yang dibuat menggunakan cara *native* menunjukkan performa yang lebih baik meskipun dalam kasus lainnya terlihat bahwa penggunaan metode *cross-platform* lebih menghemat waktu. Dari studi di atas, belum adanya penelitian yang berfokus pada implementasi pengukuran performa dengan berdasarkan pada pemilihan *framework* dari sisi pemrograman aplikasi..

2.2 Core Framework

Kecepatan dari sebuah aplikasi pada perangkat *smartphone* menjadi salah satu faktor penentu keterikatan pengguna dengan aplikasi. Sebelumnya, perusahaan Apple menyediakan OpenGL yang menjamin kecepatan grafik pada aplikasi iOS. OpenGL ini berjalan dengan cepat karena diproses dengan menggunakan *Graphics Processing Unit* (GPU). Akan tetapi, proses implementasi OpenGL terbilang cukup sulit karena membutuhkan kode program yang relatif panjang.



Gambar 1. Hirarki Core Framework
(DEVELOPER, 2015)

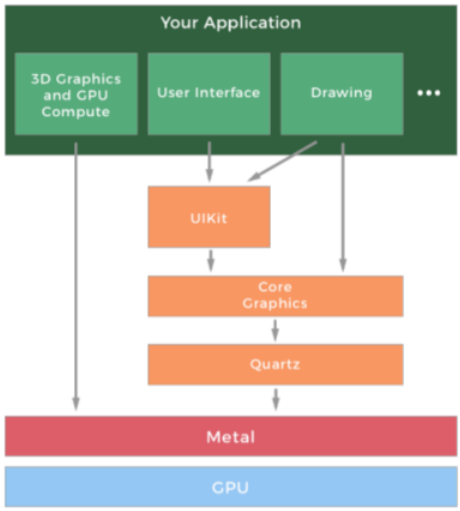
Untuk itu, perusahaan Apple mengembangkan *CoreFramework* untuk membantu pekerjaan yang berkaitan dengan gambar dan animasi. Dalam hal ini, *CoreFramework* terbagi dua, yaitu *Core Graphics* dan *Core Animation.*

1) CoreGraphics

Merujuk pada **Error! Reference source not found.** di atas, Core Graphic merupakan *framework* yang memiliki akses terdekat terhadap perangkat grafik, sehingga proses *render* sebuah gambar atau animasi bisa lebih cepat. *Framework* ini kemudian menjadi solusi yang memberikan kemudahan bagi para pengembang aplikasi yang banyak menghabiskan waktu untuk menggambar dengan menggunakan fungsi OpenGL yang dulu disediakan. Meskipun tidak serumit OpenGL pada proses implementasi, penggunaan *CoreGraphics* untuk pembuatan objek visual sederhana pun masih memerlukan usaha yang cukup banyak.

Oleh sebab itu, untuk menyederhanakan proses itu, Apple mengembangkan *CoreAnimation*. *CoreAnimation* merupakan *framework* yang pada hirarkinya berada di atas *CoreGraphics*. Hal ini menjadikan proses pembuatan gambar menjadi lebih

mudah dan tidak membutuhkan kode program yang rumit.



Gambar 2. Arsitektur Core Graphics
(Kharchyshyn, 2019)

2) CoreAnimation

Dalam hal animasi atau pembuatan *shape* pada iOS, tugas tersebut dilakukan oleh *framework* yang berada pada level yang lebih tinggi, yaitu *CoreAnimation*. Dengan menggunakan *framework* ini untuk menggambar proses animasi pada aplikasi, para pengembang hanya perlu melakukan konfigurasi sederhana. Proses selanjutnya akan dilanjutkan oleh *CoreAnimation*. *CoreAnimation* memiliki *class* CALayer yang menyediakan akses paling dasar dari kemampuan grafik yang disediakan *framework* ini. Sederhananya, CALayer disediakan guna menampilkan informasi berupa gambar dengan cepat dan juga mudah. CALayer yang merupakan bagian dari *CoreAnimation* akan mengirim informasi tersebut kepada *CoreGraphics*. CoreGraphic kemudian akan meneruskan permintaan tersebut kepada OpenGL sehingga proses *render* akan semakin cepat karena akan menggunakan GPU. Hal ini jelas meringankan beban kerja dari CPU sehingga aplikasi menjadi lebih ringan dan cepat ketika harus melakukan proses *render* objek dan animasi.

2.3 Kriteria *Variable* Pengujian

Pengujian ini dilakukan pada studi kasus yaitu *general-graph* dengan menggunakan tiga jenis *variable* utama yang akan dimonitor dan di analisis hasilnya. Adapaun kriterianya yaitu:

1) Waktu pembuatan objek

Perhitungan waktu untuk pembuatan objek ini dihitung berdasarkan waktu yang diperlukan dari mulai proses inisialisasi hingga objek lingkaran dan *label* nya tergambar dan berhasil tampil di layar. Waktu mulai dan selesai ini diambil dalam satuan *millisecond* (ms). Waktu ini digunakan untuk mendapatkan selisih sehingga lama proses penggambaran objek dan *label* pada aplikasi bisa ditentukan.

2) Persentase CPU

Pada aplikasi *mobile*, kelancaran dari sebuah aplikasi menjadi poin yang penting untuk diperhatikan. Dalam hal ini, CPU memiliki peran yang penting untuk mengatur pembagian tugas antar setiap komponen terkait untuk menjamin sebuah aplikasi berjalan dengan baik.

3) Memori

Berbeda dengan komputer atau *laptop*, perangkat *mobile* memiliki kapasitas *Random Access Memory* (RAM) yang cukup terbatas. Rata-rata *smartphone* memiliki RAM berkisar antara 2GB hingga 12GB. Perangkat keluaran Apple versi *smartphone* sendiri menawarkan ukuran RAM yang lebih kecil dibandingkan *smartphone* dengan merk lainnya, yaitu hanya 1GB hingga 4GB. Sedikit banyaknya RAM yang terpakai akan mempengatuhi kemampuan perangkat saat melakukan proses *multitasking*. Penggunaan memori pada sistem operasi iOS ini diatur oleh aplikasi itu sendiri, sehingga perlu untuk memperhatikan berapa banyak memori yang terpakai ketika aplikasi visualisasi pada studi kasus ini dijalankan.

2.4 Pemilihan Perangkat Uji

Pengujian performa ini akan dilakukan dengan menggunakan perangkat asli dengan jenis iPhone dan iPad. Perangkat iPhone yang digunakan ada dua tipe, yaitu seri lama 6s dan seri yang cukup baru yaitu iPhone X. Sementara untuk model iPad yang digunakan adalah iPad 4 WiFi keluaran tahun 2012 dan iPad Pro 10.5 tahun 2017. Perbedaan seri antara jenis perangkat ini cukup jauh guna menguji performa pembuatan objek pada perangkat dengan seri lama dan seri baru.

Table 1. Spesifikasi iPhone 6s dan iPhone X

Spek/Jenis	***iPhone 6s***	***iPhone X***
Ukuran	4.7”	5.8”
Processor	Chip A9	Chip A11 Neural Engine
SO	12.4.1	12.4.1
RAM	2 GB	3 GB
Layar	Super Retina HD	Retina HD
Resolusi	2436 x 1125 (458 ppi)	1334 x 750 (326 ppi)
CPU	Dual-core 1.84 GHz Apple Twister	Hexa-core 2.39 GHz (2x Monsoon + 4x Mistral)
GPU	PowerVR GT7600 (six-core graphics)	Apple GPU (three-core graphics)
Tahun Produksi	2015	2017

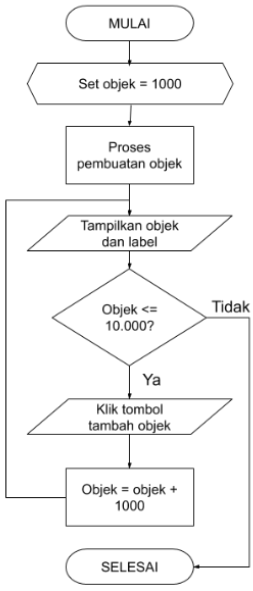
Table 2. Spesifikasi iPad 4 dan iPad Pro

Spek/Jenis	*iPad 4 WiFi*	*iPad Pro 10.5*
Ukuran	9.7”	10.5”
Processor	Apple A6X	Apple A10X
SO	10.3.4	12.4.1
RAM	1 GB	4 GB
Layar	IPS LCD	Retina
Resolusi	1536 x 2048 (264 ppi)	2048 x 1536 (264 ppi)
CPU	ARM Apple Cortex A7	2360 MHZ Cores 3
GPU	PowerVR SGX554MP4 (quad-core graphics)	PowerVR Cores 12
Tahun Produksi	2012	2017

iPhone dan iPad yang menjadi perangkat uji ini menggunakan aplikasi yang dikembangkan dengan menggunakan metode dan bahasa pemrograman yang sama, yaitu dengan bahasa Swift atau objective-C. Aplikasi *mobile* yang baik adalah aplikasi yang masih dapat berjalan dengan baik walaupun beroperasi pada perangkat dengan spesifikasi yang cukup rendah. Adapun spesifikasi dari perangkat yang digunakan dapat dilihat pada Table 1 dan Table 2 di atas.

2.5 Perancangan Sistem

Aplikasi ini akan dikembangkan pada sistem operasi iOS dengan bahasa pemrograman Swift. Tampilan aplikasi akan terdiri dari objek, warna dan *label*. Adapun alur sistem dari *prototype* aplikasi ini tergambar pada *flowchart* Gambar **3**. Alur sistem dari *prototype* aplikasiGambar 3 berikut.



Gambar 3. Alur sistem dari *prototype* aplikasi

Pengujian ini dilakukan dengan menggunakan kode program sesuai *variable* uji yang disematkan pada aplikasi. Penggunaan kode program secara langsung membuat proses uji menjadi lebih cepat dan juga dapat memberi hasil yang lebih akurat untuk mengukur interval saat sebuah *method* dijalankan. Sebuah aplikasi visualisasi berbasis iOS sederhana dibentuk dengan menggunakan bahasa pemrograman Swift. Objek yang digambar adalah objek berbentuk lingkaran yang diberi *label* sebagai penanda nomor urut objek tersebut.

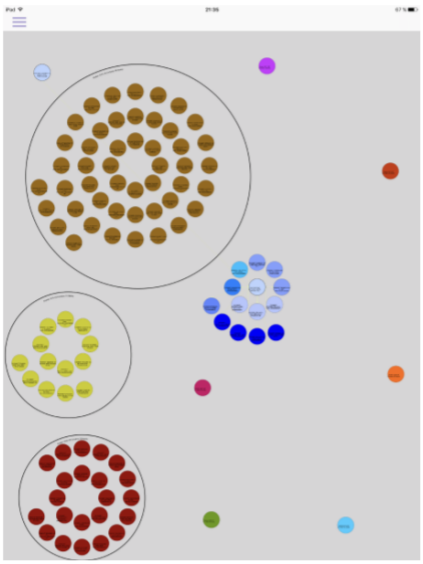
Penambahan objek akan terjadi setiap kali tombol ditekan atau data baru ditambahkan. Pada pengujian ini, jumlah objek akan terus bertambah hingga terbentuk objek dengan total 10.000 buah. Pengujian dengan tiga variabel ini dilakukan sebanyak tiga kali dengan jumlah objek yang sama pada setiap perangkat (total empat perangkat). Pemilihan pengujian sebanyak tiga kali ini berdasarkan dari selisih hasil pengujian yang memiliki perbedaan sekitar 0.5% antara percobaan satu dan lainnya. Selanjutnya nilai dari tiga kali pengujian tersebut akan dicari nilai rata-ratanya untuk dilakukan perbandingan.

3. Hasil dan Pembahasan

Secara teori, *CoreAnimation* dan *Core Graphics* yang digunakan pada aplikasi ini tidak melakukan proses penggambaran objek secara langsung pada aplikasi. Konten yang akan dibuat akan disimpan *cache* berbentuk bitmap. Sehingga ketika ada perubahan yang terjadi, maka yang diubah adalah lapisan objek tersebut, bukan menggambar ulang objeknya. *CoreAnimation* akan meneruskan perubahan itu ke perangkat grafis pada *smartphone*. Sebelumnya, penggambaran objek akan memanggil method *drawRect:* yang bertanggung jawab untuk membuat ulang gambar jika diperlukan. Akan tetapi, menggambar ulang objek tentu akan membutuhkan waktu yang cukup lama dan proses yang panjang sehingga memakan banyak CPU (DEVELOPER, 2015). *CoreFramework* dalam hal ini menghindarkan alur panjang tersebut untuk memberi hasil yang serupa.

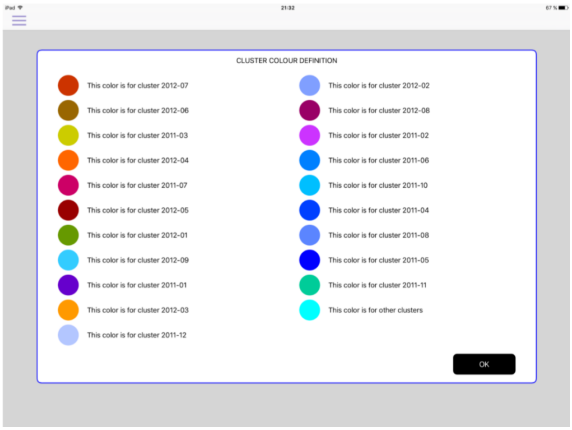
Aplikasi visualisasi ditampilkan seperti pada Gambar 4 yang dijalankan pada perangkat iPad. Objek akan dikelompokkan berdasarkan tanggal yang tersimpan pada data *dummy*. Setiap kali data bertambah, maka grafik akan disusun kembali berdasarkan tanggal yang sama.

Jika mengacu pada teori di atas, maka inisialisasi konten awal (objek lingkaran) yang digunakan pada aplikasi akan tersimpan pada memori sementara berupa *bitmap*. Sehingga ketika objek bertambah dan harus disusun ulang, maka *CoreFramework* akan memanggil kembali *cache* tersebut, dan mengatur kembali posisi, warna, *label* dan memperbanyak jumlahnya. Ini yang membuat proses menampilkan objek animasi pada aplikasi visualisasi menjadi sangat cepat.



Gambar 4. Tampilan Visualisasi *General-Graph* dengan *Cluster*

Pemberian warna pada objek lingkaran mengikuti pengelompokannya. Gambar 5 di bawah menunjukkan informasi tentang representasi warna yang ditampilkan pada aplikasi.



Gambar 5. Keterangan kriteria Pengelompokan Grafik

Setelah aplikasi visualisasi selesai terbentuk, maka dilakukan pengujian seperti pada perancangan sistem. Berikut ini merupakan hasil rata-rata dari data yang dikumpulkan dalam tiga kali pengujian yang dilakukan.

Ketika diuji menggunakan perangkat iPhone 6s dengan kapasitas RAM 2GB, pembuatan 10 ribu objek hanya memakan waktu 1.8 detik. Penggunaan memori juga masih tergolong hemat untuk jumlah objek dan animasinya yang cukup banyak. Tetapi sebagai konsekuensinya, penggunaan CPU mencapai 20% untuk aplikasi ini.

Model iPhone	Jumlah Objek	Waktu Create	CPU	Memory MB
iPhone 6S	1000	68.0	2.3	18.6
iPhone 6S	2000	196.0	5.1	48.8
iPhone 6S	3000	325.3	4.7	107.3
iPhone 6S	4000	446.3	5.5	151.9
iPhone 6S	5000	655.0	7.0	196.4
iPhone 6S	6000	832.3	8.5	241.9
iPhone 6S	7000	1165.3	11.8	289.3
iPhone 6S	8000	1465.0	14.7	336.7
iPhone 6S	9000	1635.0	16.4	384.8
iPhone 6S	10000	1813.0	18.1	423.3

Gambar 6. Rata-rata pengujian pada iPhone 6s

Sedikit berbeda dengan hasil uji pada iPhone 6s, iPhone X (Gambar 7) dapat lebih memaksimalkan penggunaan CPU. Hanya sekitar 10% CPU terpakai untuk membentuk 10 ribu objek pada aplikasi ini. Tetapi perlu diperhatikan bahwa untuk meringankan kerja CPU, penggunaan memori pada iPhone X untuk aplikasi ini menjadi lebih boros. Penambahan objek dua kali lipat nantinya bisa memakan sepertiga dari total RAM yang disediakan.

Model iPhone	Jumlah Objek	Waktu Create	CPU	Memory MB
iPhone X	1000	40.7	2.3	19.1
iPhone X	2000	120.3	2.7	80.4
iPhone X	3000	180.7	4.3	139.9
iPhone X	4000	239.0	3.8	200.4
iPhone X	5000	365.3	4.6	261.7
iPhone X	6000	454.0	5.3	323.5
iPhone X	7000	668.3	7.0	386.9
iPhone X	8000	848.0	8.5	451.0
iPhone X	9000	920.3	9.3	516.5
iPhone X	10000	1001.7	10.1	584.4

Gambar 7. Rata-rata pengujian pada iPhone X

Ketika dijalankan pada iPad Pro, hasil yang didapatkan tidak jauh berbeda dengan hasil pengujian pada perangkat *smartphone*. Penggunaan CPU dan memori saat menghasilkan puluhan ribu objek cukup seimbang.

Model iPad	Jumlah Objek	Waktu Create	CPU	Memory MB
iPad Pro 10.5	1000	43.0	4.7	18.5
iPad Pro 10.5	2000	129.7	5.8	64.1
iPad Pro 10.5	3000	208.3	4.0	107.8
iPad Pro 10.5	4000	287.7	4.1	152.7
iPad Pro 10.5	5000	434.7	5.2	198.4
iPad Pro 10.5	6000	547.7	6.2	243.8
iPad Pro 10.5	7000	785.7	8.2	291.9
iPad Pro 10.5	8000	992.0	10.1	340.2
iPad Pro 10.5	9000	1102.0	11.1	389.0
iPad Pro 10.5	10000	1208.0	12.1	437.1

Gambar 8. Rata-rata pengujian pada iPad Pro 10.5

Hasil yang cukup berbeda terjadi ketika aplikasi di uji pada iPad 4 Wifi keluaran tahun 2012 (Gambar 9). Jika pembuatan objek pada tiga perangkat lainnya memakan waktu kurang dari dua detik untuk 10 ribu objek, iPad ini membutuhkan waktu yang sama hanya untuk menghasilkan sekitar 300 buah objek lingkaran. Bahkan ketika dicoba membentuk 10 ribu objek, aplikasi langsung mengalami *error* setelah menunggu dalam kurun waktu yang cukup lama. Penggunaan CPU juga sangat tinggi yaitu mencapai 60% pada pembentukan 9000 objek.

Model iPad	Jumlah Objek	Waktu Create	CPU	Memory MB
iPad 4 WiFi	1000	278.7	3.2	9.9
iPad 4 WiFi	2000	708.0	7.2	51.3
iPad 4 WiFi	3000	1205.0	12.1	91.5
iPad 4 WiFi	4000	1720.0	17.2	132.4
iPad 4 WiFi	5000	2410.7	21.8	173.2
iPad 4 WiFi	6000	3105.0	31.1	214.5
iPad 4 WiFi	7000	4101.7	41.1	256.3
iPad 4 WiFi	8000	5058.3	50.6	298.4
iPad 4 WiFi	9000	5972.3	59.7	339.1
iPad 4 WiFi	10000	error	error	error

Gambar 9. Rata-rata pengujian pada iPad 4 Wifi

4. Kesimpulan dan Saran

Berdasarkan hasil pengujian di atas, ada beberapa poin yang diketahui dari hasil pengujian performa *CoreFramework* pada memori, CPU dan waktu pengerjaan.

1) Meskipun *CoreFramework* lebih banyak menggunakan GPU untuk prosesnya penggambaran objek, tetapi hal itu juga bergantung dengan kapasitas RAM yang ditawarkan pada perangkat yang digunakan. Semakin rendah RAM nya, maka alokasi CPU yang terpakai akan semakin tinggi. Hal ini terlihat dari hasil uji pada iPad 4 yang memiliki RAM hanya sebesar 1GB.
2) Sesuai dengan hasil yang didapat, bahwa dengan menggunakan *CoreFramework*, maka visualisasi tetap dapat dijalankan dengan baik pada perangkat seri lama walau memiliki keterbatasan. Pembuatan objek pada perangkat seri lama dengan spesifikasi terbatas hanya mampu membuat kurang dari 11 ribu objek.
3) Dalam hal ini, aplikasi yang menggunakan banyak objek dan animasi membutuhkan *processor* dan RAM yang tinggi. Semakin tinggi sumber daya tersebut, akan semakin meringankan proses pembuatan gambar objek pada aplikasi. Terlihat bahwa 10 ribu objek hanya membutuhkan waktu kurang dari dua detik untuk digambarkan. Waktu tersebut masih dapat ditoleransi oleh pengguna ketika berinteraksi dengan aplikasi visualisasi. Hal ini karena kemampuan *CoreFramework* yang memungkinkan untuk membentuk objek sekaligus pekerjaan lainnya secara bersamaan.

Untuk mendukung hasil uji yang lebih akurat, maka diperlukan hasil perbandingan dengan menggunakan sebuah *tool* pengujian. Xcode Instrument merupakan sebuah *tool* bawaan Xcode dapat digunakan untuk menguji ketiga *variable* di atas.

Daftar Pustaka

B. Gaudette, C. W. (2016). Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee. *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 52-63.

C. GAO, A. G. (2015). A study of mobile device utilization. *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Philadelphia.

Chen, A. (2019, 10 12). *New data shows losing 80% of mobile users is normal, and why the best apps do better* . Retrieved from @andrewchen: https://andrewchen.co/new-data-shows-why-losing-80-of-your-mobile-users-is-normal-and-that-the-best-apps-do-much-better/

DEVELOPER, A. (2015, March 9). *Core Animation Programming Language.* Retrieved Oktober 24, 2019, from https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreAnimation_guide/Introduction/Introduction.html

GLOBALWEBINDEX. (2018). *Examining the attitudes and digital behaviors of internet users aged 21-34.* London: Global Web Index.

GRZMIL, P. &.-P. (2017). Performance Analysis of Native and Cross-Platform Mobile Applications. Lublin: Informatics Control Measurement in Economy and Environment Protection.

Kharchyshyn, A. (2019, 11 12). *Core Graphics Tutorial Part 2: Gradients and Contexts*. Retrieved from Raywenderlich.com: https://www.raywenderlich.com/410-core-graphics-tutorial-part-2-gradients-and-contexts

M. WILLOCX, J. a. (2015). A Quantitative Assessment of Performance in Mobile App Development Tools. *IEEE International Conference on Mobile Services.* New York.

M. Willocx, J. V. (2016). Comparing Performance Parameters of Mobile App Development Strategies. *IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 38-47.