Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 4, No. 3, September 2017, hlm. 207-212 e-ISSN: 2528-6579

OPTIMASI PROSES RENDERING OBJEK GAME 3D MENGGUNAKAN PEMROGRAMAN CUDA PADA GAME SANDBOX CRAFT

Hilmi Ilyas Rizaldi[1], Firadi Surya Pramana[2], Bariq Najmi R.[3], Aditya Yudha A.N.[4], Imam Cholissodin[5]

[1,2,3,4,5]Fakultas Ilmu Komputer Universitas Brawijaya
Email: [1]developer.hilmi@gmail.com, [2]fspramana@gmail.com, [3]bariq3396@gmail.com,
[4]adi.kyoudha@gmail.com, [5]imamcs@ub.ac.id

(Naskah masuk: 3 September 2017, diterima untuk diterbitkan: 29 September 2017)

Abstrak

Kemajuan dalam pengembangan game khususnya 3D game menumbuhkan perasaan *immersive* yang lebih nyata. Namun, hal tersebut membutuhkan pengolahan *resource* yang sangat banyak dan cepat. Kerja *Central Processing Unit (CPU)* sendiri saja tidak cukup. Oleh karena itu, dibutuhkan adanya pengelola *resource* tambahan yang dapat digunakan untuk mempercepat proses. Penelitian ini membahas tentang bagaimana *Graphics Processing Unit (GPU)* dapat melakukan optimasi kerja *CPU* dalam memproses *resource* yang sangat banyak. Metode yang dibahas pada penelitian ini adalah shared memory. Shared memory memungkinkan CPU dan GPU untuk berbagi resource yang diproses. Game yang dianalisis pada penelitian ini adalah Craft, yaitu game yang memiliki tipe permainan sandbox layaknya Minecraft. Hasil yang didapatkan membuktikan bahwa metode shared memory dapat melakukan optimasi dari game yang membutuhkan pengolahan *resource* yang banyak dengan cepat.

Kata kunci: *games, shared memory, CPU, GPU, CUDA, comparative analysis, optimization, parallel computing*

Abstract

Game development nowadays especially 3D game bring a more realistic immersive feeling. Although, that needs a lots of resource processing and quick processing. That is said, Central Processing Unit (CPU) alone is not enough. Therefore, additional process manager is needed to make the process faster. This research focuses on how Graphics Processing Unit (GPU) can optimize resource processing of CPU. The proposed approach is to use shared memory. Shared memory allows CPU and GPU to share resource processing. The game used in this research is Craft. Craft is the same as Minecraft. It is a game that focuses on sandbox gameplay. The result showed that this approach can greatly optimize game that needed a lots of resource processing.

Keywords: *games, shared memory, CPU, GPU, CUDA, comparative analysis, optimization, parallel computing*

1. PENDAHULUAN

CPU dapat disebut juga sebagai Central Processing Unit yang digunakan untuk melakukan suatu fungsi yang dibutuhkan untuk menjalankan sebuah proses. CPU mengeksekusi suatu proses seperti perhitungan aritmatika, kalkukasi dan operasi yang berhubungan dengan komputasu matematis lainnya. Dan video games membutuhkan CPU untuk melakukan proses yang berfungsi menentukan kalkulasi khusus untuk game.

GPU (Graphical Processing Unit) adalah suatu komponen yang mempunyai hubungan dengan kecepatan proses, serta dari segi kualitas hasil yang ditampilan. Sebenarnya, computer secara umum dapat berfungsi atau berjalan tanpa GPU (Nvidia, 2017), namun yang sering terjadi adalah pada ketidakmampuannya dalam menampilkan hasil dengan kualitas terbaik di layar atau monitor. GPU sendiri tersedia dalam berbagai jenis, bentuk, maupun ukuran yang umumnya sering disebut dengan card dan dapat dicolokan ke dalam slot PCI-Express pada *motherboard*, hingga bentuk yang lain yaitu chip *onboard* yang tertanam di dalam *motherboard* secara langsung disebut dengan *integrated graphic chip*. Perbedaan yang jelas diantara CPU dan GPU adalah kegunaan GPU yang khusus untuk melakukan pemrosesan grafis dan mempunyai kemampuan untuk melakukan perhitungan hingga banyak kalkulasi per detik. Jumlah banyaknya core yang terdapat di dalam perangkat GPU itu sendiri tergantung dari masing-masing vendor di market. Hingga sekarang Nvidia memiliki spesifikasi yang cukup tinggi pada setiap graphic chip yang mereka tawarkan meski jumlahnya tidak banyak, sementara itu graphic chip dari pabrikan lain yang menjadi penantang dari Nvidia yaitu AMD (Advanced Micro Devices) memiliki banyak chip yang tertanam di dalam kartu grafisnya untuk meningkatkan performa dari pengolahan grafis. Kartu grafis sendiri memiliki kelas masing-masing, tipe high-end biasanya memiliki core yang lebih banyak, mulai dari 68 core hingga 1500 core atau bahkan lebih banyak lagi.

Video game memiliki berbagai jenis maupun tipe permainan yang ditawarkan. Jenis sandbox adalah game yang membebaskan pemain untuk

melakukan apapun dan menjelajahi dunia dalam game tersebut sesuka hati. (Adams dan Ernest, 2010). Seperti pada game Minecraft, terdapat blok-blok yang disusun secara rapi untuk membentuk dunia dari game tersebut. Sama layaknya pada game The Elder Scroll Skyrim. Pemain dihadapkan dengan dunia masa lalu/*medieval*, yang mana pemain dapat menelusuri game tersebut tanpa ada batas. Melalui pengamatan dari game-game tersebut, dalam penelitian ini mengambil kata kunci dari game berjenis sandbox yaitu dunia atau dalam kata lain game *environment*.

Game *environment* adalah dunia yang dikembangkan dalam game untuk dijelajahi. Hal ini merupakan gabungan dari banyak elemen yang saling bekerja sama untuk dapat membangun sebuah kedalaman desain serta perasaan bahwa dunia ini layaknya nyata. Komponen tersebut antara lain desain *environment*, proyeksi cahaya, bayangan, tekstur teraplikasi, partikel, serta material dari objek dalam *environment*. Dari kebutuhan tersebut, CPU mengalami kesulitan dalam mengelola semua komponen di atas. Maka dibutuhkannya kerja sama antar CPU dengan perangkat pada GPU sehingga kerja pada CPU dalam pertukaran data dapat lebih mudah. Namun, desain game *environment* yang semakin hari semakin luas dan semakin detail berdampak kepada performa dari game tersebut apakah game memiliki kemampuan *loading* yang cepat atau lambat (Ruggill et al, 2011).

Game berjenis sandbox memiliki tipikal dengan rendering *environment* yang luas serta detail. Maka dari itu, dibutuhkan sebuah optimasi agar rendering dapat melakukan proses yang lebih cepat dan terasa real-time. Namun, untuk mendapatkan kecepatan render, dibutuhkan kerjasama antar CPU dan GPU dalam prosesnya. Oleh karena itu, pada penelitian ini dilakukan simulasi untuk render dengan CPU dan CPU bersama GPU. Simulasi dilakukan dalam platform C++ dan NVIDIA CUDA dengan bantuan OpenGL serta *library* yang mendukung pengerjaan.

2. DASAR TEORI

2.1 OpenGL dan Game Loop

OpenGL adalah API tingkat rendah (Application Programming Interface), yang memungkinkan programmer, memakai antarmuka untuk perangkat keras grafis (GPU). Keuntungan utama yang dimiliki OpenGL di atas API grafis lainnya adalah platform berjalan pada berbagai platform. OpenGL dapat berjalan di Windows, Linux, dan Mac OS X.

Konsep awal dari game loop adalah memproses *input* dari *user* tetapi tidak menunggu dan selalu melakukan loop secara terus menerus sampai *user* melakukan proses pemberhentian game. Hal tersebut membuat sebuah CPU menjadi lebih berat. Pada Gambar 1 terdapat icon waktu yang menandakan *sleep* untuk menahan kecepatan loop agar sesuai pada tiap framenya. Prinsip menahan kecepatan loop tersebut nantinya akan di proses menjadi sebuah frames per second (FPS).

Gambar 1. Game Loop

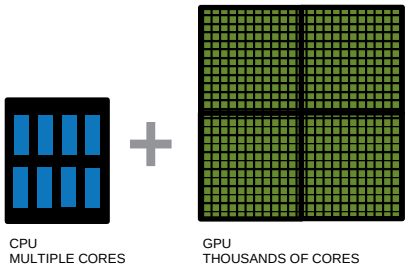
2.2 Pemograman GPU dan CUDA

Pemrograman GPU dengan Nvidia CUDA seperti pada Gambar 2, memiliki tujuan tidak hanya untuk mengolah grafis melainkan juga dapat untuk tujuan umum, misal komputasi ilmiah menggunakan *machine learning* dan rekayasa lainnya pada game development, dan sebagainya. dengan *framework* tertentu, misal CUDA.

Gambar 2. Nvidia CUDA

CUDA adalah sebuah API yang dikembangkan oleh Nvidia yang digunakan untuk melakukan suatu komputasi yang dapat berjalan secara paralel atau dengan kata lain secara bersama-sama. Para *developer* dapat menggunakan CUDA untuk pemrosesan tujuan umum atau disebut dengan pendekatan GPGPU (komputasi *General-purpose on* GPU). Platform CUDA merupakan lapisan akses pada perangkat lunak atau *software* yang memberikan langsung ke set instruksi virtual GPU untuk kernel sebagai pelaksana penghitungan.

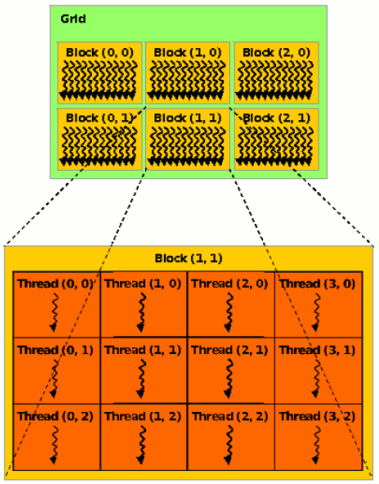
Platform CUDA dirancang untuk bekerja dengan bahasa pemrograman seperti C, C++, dan Fortran. Aksesibilitas ini memudahkan para spesialis dalam pemrograman paralel untuk menggunakan sumber daya GPU, berbeda dengan API sebelumnya seperti Direct3D dan OpenGL, yang memerlukan keterampilan lanjutan dalam pemrograman grafis. Selain itu, CUDA mendukung *framework* pemrograman seperti OpenACC dan OpenCL. Ketika pertama kali diperkenalkan oleh Nvidia, nama CUDA adalah akronim dari Compute Unified Device Architecture. Pada Gambar 3 menunjukkan besarnya perbedaan banyaknya jumlah core yang ada pada CPU dan GPU.



Gambar 3. CPU versus GPU

2.5 Grid, Block, dan Thread

Pada Gambar 4, dalam sebuah grid terdapat block-block. Dan pada masing-masing block terdapat thread-thread. Grid sendiri ialah sebuah grup dari block-block, yang mana block-block tersebut tidak terjadi sebuah proses sinkronisasi antar block. Block adalah sebuah grup dari thread. Thread-thread ini dapat berjalan *concurrent* atau pun secara seri dengan urutan yang tidak pasti. Sehingga, dengan menggunakan fungsi __syncthreads() dapat membuat sebuah thread dapat berhenti pada titik tertentu di dalam kernel sampai proses lainnya juga sampai pada titik yang sama tersebut. Thread adalah sebuah eksekusi dari kernel dengan sebuah index yang diberikan/ditentukan. Setiap thread akan menggunakan index tersebut untuk mengakses element di dalam array seperti koleksi-koleksi dari semua thread yang bekerja sama pada semua data set.



Gambar 4. Grid, Block, dan Thread

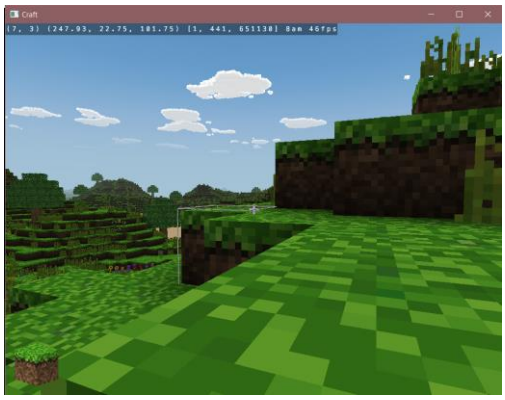
3. PERANCANGAN DAN IMPLEMENTASI

Kode program yang dioptimasi, didapatkan dari sumber ini (https://github.com/fogleman/Craft) pada Kode Program 1, dan hasil optimasi kinerja menggunakan pemrograman CUDA ditunjukkan pada Kode Program 2 dan 3.

Spesifikasi kebutuhan *hardware*, *software*, serta dependency file (*.dll, *.lib, dan header atau kode lain) ketika melakukan implementasi:

- Core i7 7700HQ, RAM 16 GB
- NVIDIA GeForce GT 1050
- Windows 10
- Visual Studio Profesional 2015
- CUDA Toolkit 8.0
- file *.dll
- file *.lib
- file *.h atau kode lainnya, seperti penjelasan di bawah ini

Implementasi yang di ambil tidak termasuk dengan *library* CURL, yang menangani sistem online pada game. Pada kode hasil optimasi dengan CUDA, tidak perlu menggunakan auth.h, client.h, karena game yang diubah adalah *gameplay* yang bertipe *single player offline*, dan lebih fokus pada proses optimasi game.



Gambar 5. Tampilan Game Craft

Dikarenakan game ini 3D dan menggunakan banyak matriks dan vector. Dibuatnya sebuah class matrix untuk meng-*handle* fungsi-fungsi penggunaan matrix pada umumnya, yang mana pada C++ menggunakan array 1 dimensi dan membutuhkan banyak pengulangan. Terdapat sebuah proses perkalian matrix mat_multiply pada Game Craft yang digunakan untuk perkalian matriks untuk perhitungan penyimpanan matriks 3D pada game di method yang digunakan pada file kode matrix.c yang masih menggunakan CPU, seperti pada Kode Program 1.

```
1  void mat_vec_multiply(float
2  *vector, float *a, float *b) {
3   float result[4];
4   for (int i = 0; i < 4; i++) {
5    float total = 0;
6    for (int j = 0; j < 4; j++) {
7     int p = j * 4 + i;
8     int q = j;
9     total += a[p] * b[q];
10   }
11   result[i] = total;
12  }
13  for (int i = 0; i < 4; i++) {
14   vector[i] = result[i];
15  }
16 }
```

Kode Program 1. Kode CPU

Kode di atas dapat diubah menjadi menggunakan GPU dengan menggunakan perhitungan konsep paralel *programming*, yang mana akan menggunakan thread pada GPU untuk perhitungan setiap hasil, jadi perkalian akan di-*handle* oleh thread sehingga tidak perlu menggunakan banyak pengulangan dan penggunaan CPU akan menjadi lebih rendah. Penggunaan jumlah grid, block, dan thread, seperti pada Kode Program 2.

```
1  __global__ void
2  mat_vec_multiply(float *vector,
3  float *a, float *b) {
4   int kolom = threadIdx.x; //
5  threadIdx adalah thread Index
6   int baris = threadIdx.y;
7   float c = 0;
8   int ordoMat = 4;
9   for (int k = 0; k < ordoMat;
10 k++) {
11   c += a[baris*ordoMat + k] *
12 b[k*ordoMat + kolom];
13  }
14  vector[baris*ordoMat + kolom]
15 = c;
16 }
```

Kode Program 2. Kode CUDA ke-1

Penjelasan dari Kode Program CUDA 1:

1. Baris 1-3 merupakan deklarasi fungsi kernel mat_vec_multiply.
2. Baris 4-6 deklarasi kolom mengunakan threadIdx.x dan baris menggunakan threadIdx.y untuk perhitungan matriks.
3. Baris 7 deklarasi c sebagai penyimpan hasil perhitungan sementara.
4. Baris 8 deklarasi hasil dari perkalian ordo matriks.
5. Baris 9-13 proses perhitungan dengan thread GPU yang di-*looping* agar hasil dari perkalian dapat dijumlahkan sebagai hasil.
6. Baris 14 proses memasukan hasil perkalian ke dalam pointer vector yang nantinya akan di pakai pada perhitungan selanjutnya pada game.

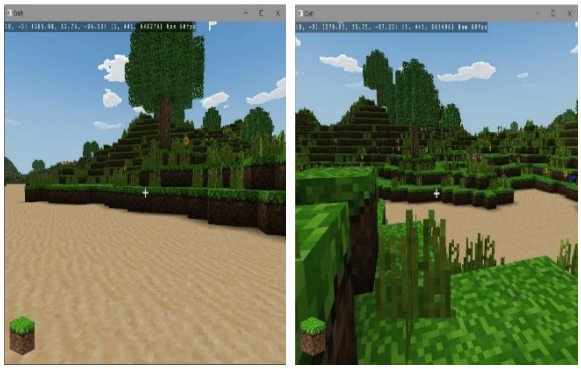
Kode tersebut dibutuhkan pemindahan isi variabel pada memori CPU ke GPU menggunakan cudaMemcpy agar dapat di proses pada method kernel, seperti pada Kode Program 3.

```
1  int jumlahBlock = 1;
2  dim3 threadPerBlock(2, 2);
3      .........
4    cudaMemcpy(d_A, h_A,
5  sizeof(float) * jumlahElemen,
6  cudaMemcpyHostToDevice);
7    cudaMemcpy(d_B, h_B,
8  sizeof(float) * jumlahElemen,
9  cudaMemcpyHostToDevice);
10   mat_vec_multiply << <
11 jumlahBlock, threadPerBlock >>
12 > (d_HasilPerkalian, d_A, d_B);
13   cudaThreadSynchronize();
14   cudaMemcpy(h_HasilRef,
15 d_HasilPerkalian, sizeof(float)
16 * jumlahElemen,
17 cudaMemcpyDeviceToHost);
```

Kode Program 3. Kode CUDA ke-2

Penjelasan dari Kode Program CUDA 2:

1. Baris 1-2 merupakan deklarasi grid, block, dan thread.
2. Baris 4-6 mengalokasikan size memori variabel pada device.
3. Baris 7-9 copy isi variabel dari host ke device.
4. Baris 10-12 pemanggilan method device.
5. Baris 13 sinkronisasi thread pada device agar selesai secara bersamaan.
6. Baris 14-17 copy isi variabel hasil dari device ke host.



(CPU) (GPU)

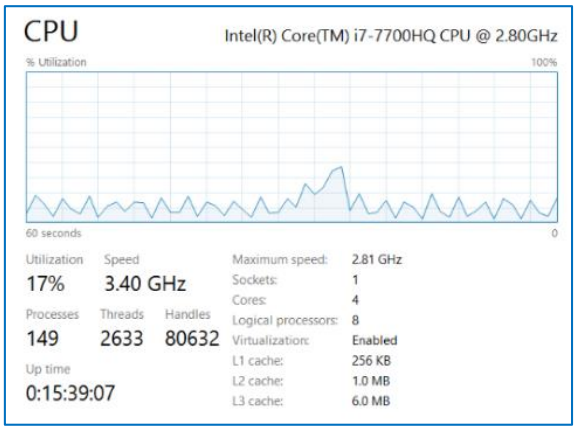
Gambar 6. Hasil running CPU dan GPU

Hasil proses ketika *running* program ketika dijalankan dapat dilihat pada Gambar 6, sekilas dari segi tampilan tidak ada perbedaan yang signifikan. Namun secara detail, nantinya akan dibandingkan pada proses pengujian dan analisis bagaimana visualisasi kinerjanya CPU, tanpa menggunakan CUDA dan dengan menggunakan CUDA.

4. PENGUJIAN DAN ANALISIS

4.1 Pengujian dengan CPU

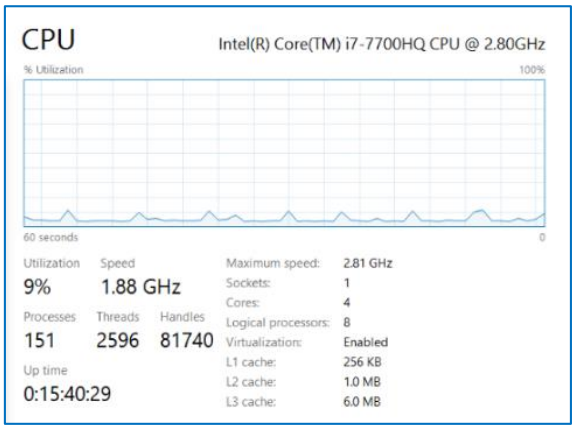
Berdasarkan grafik hasil pengujian CPU pada Gambar 7, lihat utilization, Speed, Process, Threads, dan Handles-nya.



Gambar 7. Hasil Kinerja CPU tanpa CUDA

4.2 Pengujian Kinerja dengan GPU

Berdasarkan grafik hasil pengujian CPU pada Gambar 8, maka terlihat besarnya nilai *utilization*, *clock speed*, *processes*, *threads*, dan *handles*-nya.

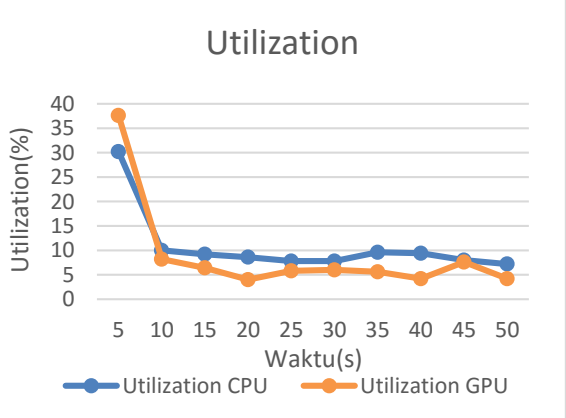


Gambar 8. Hasil Kinerja CPU dengan CUDA

4.3 Perbandingan CPU dengan GPU

Berdasarkan pengujian terhadap kinerja dari CPU dan GPU berdasarkan nilai-nilai variabel pada *task manager* didapatkan perbandingan-perbandingan data *utilization*, *clock speed*, *processes*, *threads* dan *handles*. Pada perbandingan ini akan dibahas mengenai besarnya persentase peningkatan kinerja CPU pada program yang dibuat dibandingkan dengan menggunakan GPU CUDA, sebelum dan sesudah diberikan perlakukan yang berbeda-beda. Data perbandingan ini didapatkan dari hasil analisis ketika menjalankan permainan yang batas pengamatannya dari detik 0 hingga 50. Setiap 5 detik akan di cari rata-rata dari tiap variabel yang diuji.

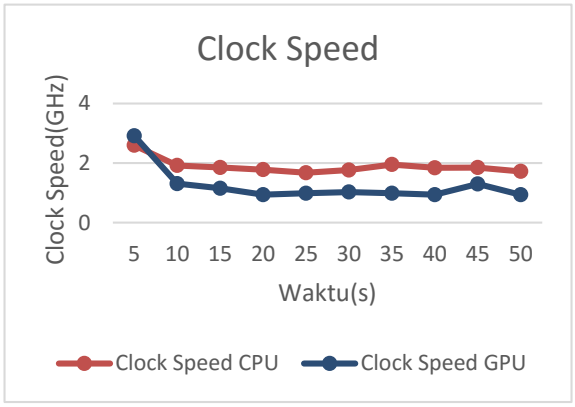
1. Utilization



Gambar 9. Utilization

Pada Gambar 9 dapat dilihat bahwa GPU menggunakan *utilization* cenderung lebih rendah, yaitu sekitar 16%, jika dibandingkan dengan hanya dijalankan pada CPU.

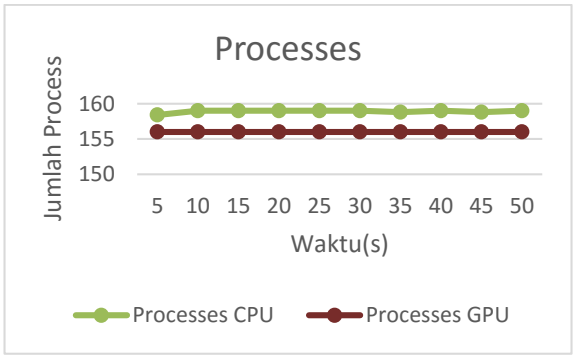
2. Clock Speed



Gambar 10. Clock Speed

Pada Gambar 10 dapat dilihat GPU menggunakan *clock speed* lebih rendah 34% dibandingkan dengan CPU.

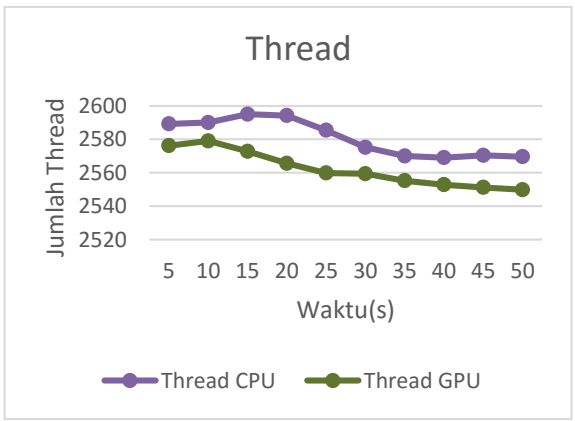
3. Processes



Gambar 11. Processes

Pada Gambar 11 dapat dilihat GPU menggunakan *processes* lebih rendah 1.8% dibandingkan dengan CPU.

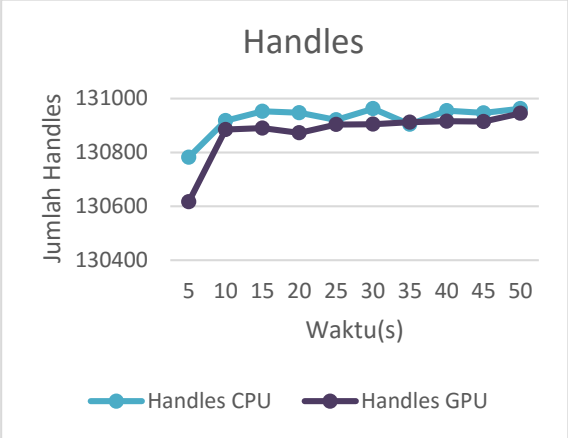
4. Thread



Gambar 12. Thread

Pada Gambar 12 dapat dilhat GPU menggunakan *threads* lebih rendah 0.7% dibandingkan dengan CPU.

5. Handles



Gambar 13. Handles

Pada Gambar 13 dapat dilihat bahwa GPU menggunakan 0.03% *handles* lebih rendah dibandingkan CPU.

5. KESIMPULAN DAN SARAN

Implementasi pada penelitian ini berhasil dilakukan dengan pengujian yang menunjukan bahwa komputasi GPU terbukti hanya menggunakan *resource* lebih rendah dibandingkan dengan CPU. Game sandbox ini berhasil berjalan sesuai dengan optimasi yang diharapkan. Pada pengujian dapat diihat bahwa *clock speed* yang digunakan pada GPU jauh lebih rendah dibandingkan yang digunakan pada CPU hingga 34%. Tingkat kerendahan tersebut cukup besar dalam hasil optimasi game sandbox ini.

Penelitian ini baru merubah salah satu dari fungsi yang ada pada game untuk dioptimasi dengan menggunakan pemrograman GPU CUDA, yang mana masih terdapat banyak fungsi lain yang dapat di panggil dalam device GPU. Penulis menyarankan agar fungsi-fungsi lainnya juga dilakukan optimasi agar mendapatkan performa yang lebih baik, serta dalam penggunaan CUDA dapat dirubah menjadi OpenCL agar permainan ini tidak hanya dapat dioptimasi pada kartu grafis Nvidia.

6. DAFTAR PUSTAKA

ADAMS, ERNEST, 2010. *Fundamentals of Game Design. New Riders*. pp. 161, 268.

WILLHALM, T., DEMENTIEV, R., FAY P., 2017. *Intel® Performance Counter Monitor - A better way to measure CPU utilization*. software.intel.com.

RUGGILL, J.E., MCALLISTER, K.S., 2011. *Gaming Matters: Art, Science, Magic, and the Computer Game Medium*. University Alabama Press; 1st Edition edition.

TARJAN, D., K. SKADRON, & P. MICIKEVICIUS, 2009. *The art of performance tuning for CUDA and many core architectures*. Birds-of-a-feather session di SC'09.

NVIDIA, 2017. *Graphics Processing Unit (GPU)*. http://www.nvidia.com/object/gpu.html (Diakses tanggal 6 Juni 2017).

NVIDIA, 2017. *What is CUDA?*. http://www.nvidia.com/object/cuda_home_new.html (Diakses tanggal 7 Juni 2017).

UNIVERSITY OF VIRGINIA, ENGINEERING, COMPUTER SCIENCE, 2017. *CUDA Optimization Techniques*. http://www.cs.virginia.edu/~mwb7w/cuda_support/optimization_techniques.html (Diakses tanggal 5 juni 2017).