Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 12, No. 5, Oktober 2025, hlm. 1073-1082 e-ISSN: 2528-6579

OPTIMALISASI SIMULASI FISIKA DUA DIMENSI MENGGUNAKAN *COMPUTE SHADER* DAN *BUFFER*

Hadziq Fabroyir[*1], Geizka Wahyu Fahriza[2]

[1]Institut Teknologi Sepuluh Nopember, Surabaya, [2]Stairway Games, Yogyakarta
Email: [1]hadziq@its.ac.id, [2]geizka.fahriza@gmail.com
[*]Penulis Korespondensi

(Naskah masuk: 8 April 2025, diterima untuk diterbitkan: 30 Oktober 2025)

Abstrak

Simulasi fisika merupakan aspek penting dalam pengembangan gim dan aplikasi *extended reality* (XR), khususnya untuk fungsi-fungsi seperti tumbukan antar objek, pergerakan, dan pemeriksaan *ray-cast*. Secara tradisional, simulasi ini dijalankan menggunakan Central Processing Unit (CPU). Namun, seiring meningkatnya kompleksitas interaksi pengguna, simulasi berbasis CPU mulai mengalami kendala skalabilitas, sehingga menyebabkan perlambatan pada eksekusi logika aplikasi secara keseluruhan. Untuk mengatasi kendala tersebut, penelitian ini memanfaatkan kemampuan pemrosesan paralel pada Graphics Processing Unit (GPU) melalui penggunaan *compute shader* dan *compute buffer* yang dinamis guna mengoptimalkan performa simulasi fisika dua dimensi. Penelitian dilakukan dengan mengembangkan sebuah kerangka kerja simulasi di mesin gim Unity, yang kemudian diuji kinerjanya dibandingkan dengan metode berbasis CPU melalui *benchmarking*. Pengujian dilakukan terhadap berbagai konfigurasi perangkat keras dengan jenis *collider* berbeda (lingkaran, kotak, poligon) serta jumlah objek yang beragam (100–3000 objek). Hasilnya menunjukkan bahwa simulasi berbasis GPU memiliki performa yang secara signifikan lebih unggul dibandingkan simulasi berbasis CPU, terutama pada skenario dengan jumlah objek yang tinggi. Temuan ini memberikan panduan praktis bagi pengembang aplikasi yang mengimplementasikan simulasi fisika untuk konfigurasi perangkat keras yang beragam.

Kata kunci: *Fisika, gim, GPU, Unity, shader, buffer* dinamis

OPTIMIZING 2D PHYSICS SIMULATIONS USING COMPUTE SHADERS AND BUFFERS

Abstract

Physics simulation is a crucial element in game development and extended reality (XR) applications, supporting essential functionalities such as object collision detection, movement, and ray-casting inspection. Traditionally, these simulations have been handled by the Central Processing Unit (CPU). However, as user interactions become increasingly complex, CPU-based simulations encounter scalability issues, leading to slower execution of game logic and reduced overall performance. To overcome these limitations, this study leverages the parallel processing capabilities of Graphics Processing Units (GPUs) through compute shaders and dynamic compute buffers to optimize 2D physics simulations. We developed a GPU-accelerated simulation framework using the Unity game engine and benchmarked its performance against traditional CPU-based approaches. Performance evaluations were conducted across multiple hardware configurations, collider types (circle, box, polygon), and varying object counts (ranging from 100 to 3000 objects). The results demonstrate that GPU-based simulations significantly outperform CPU ones, particularly in scenarios involving large numbers of objects. These findings highlight the effectiveness of GPU acceleration for enhancing the scalability and efficiency of game physics systems, providing valuable insights for developers aiming to optimize performance across diverse hardware platforms.

Keywords: *Physics, game, GPU, Unity, shader, dynamic buffer*

1. PENDAHULUAN

Graphics Processing Unit (GPU) telah berkembang menjadi komponen vital dalam komputasi modern, tidak hanya terbatas pada tugas *rendering* grafis, tetapi juga mencakup pemrosesan paralel secara umum. Kemampuan GPU untuk menjalankan ribuan instruksi secara paralel melalui *core* yang banyak memungkinkannya mengungguli Central Processing Unit (CPU) dalam berbagai tugas tertentu. Hal ini mencakup *rendering* gambar dua dimensi maupun tiga dimensi (Irawan, Ma'rufi and

Cholissodin, 2017; Rizaldi et al., 2017; Cholissodin et al., 2022a), pemrosesan video (Peng et al., 2025), hingga komputasi akal imitasi (Cholissodin et al., 2022b) dengan tingkat efisiensi yang jauh melampaui CPU.

Berbagai penelitian terbaru secara konsisten menunjukkan bahwa GPU mampu melampaui solusi berbasis CPU dengan margin yang cukup besar, terutama pada komputasi fisika yang bersifat intensif dan paralel. Sebagai contoh, simulasi termal yang dijalankan pada GPU dengan teknologi CUDA (Lee et al., 2025) atau OpenACC (Xu and Li, 2024) mampu mencapai performa hingga empat belas kali lipat dibanding CPU. Penelitian lain oleh Sung et al. (2025) juga menunjukkan peningkatan performa signifikan pada simulasi fisika tiga dimensi dalam aplikasi *extended reality* (XR) menggunakan mesin gim seperti Unreal dan Unity. Meskipun demikian, penelitian tentang simulasi fisika dua dimensi, khususnya dalam konteks Unity dengan implementasi *buffer* dinamis, masih relatif sedikit dieksplorasi.

Implementasi fisika dua dimensi berbasis CPU seperti Box2D, mesin fisika 2D *rigid body* yang kaya fitur dan umum digunakan dalam Unity, memiliki beberapa keterbatasan mendasar (Catto, 2025). Box2D tidak didesain untuk mendukung *multi-threading*, dan determinisme hasil simulasi sangat bergantung pada platform perangkat keras tertentu. Dengan demikian, hasil simulasi yang identik hanya dapat diperoleh jika dijalankan pada perangkat yang sama dengan kondisi awal yang persis serupa. Selain itu, dependensi pada komputasi *floating-point* juga menyebabkan hasil simulasi berbeda antar platform, yang semakin mengurangi determinisme simulasi (Arango, 2025).

Penelitian ini berupaya mengatasi berbagai keterbatasan tersebut dengan mengimplementasikan simulasi fisika dua dimensi berbasis GPU menggunakan *compute shader* pada mesin gim Unity. Pendekatan penelitian mencakup paralelisasi algoritma fisika dua dimensi, khususnya deteksi tumbukan dan dinamika gerak, untuk arsitektur GPU, implementasi menggunakan *compute shader*, serta evaluasi performanya dibandingkan solusi berbasis CPU (Box2D). Solusi ini tidak hanya diharapkan mengatasi kendala skalabilitas dan determinisme yang dimiliki Box2D, tetapi juga membuka peluang baru dalam pemanfaatan GPU untuk komputasi non-grafis, terutama dalam konteks pengembangan gim modern.

2. METODE PENELITIAN

Penelitian ini menggunakan kerangka kerja *benchmarking* multiperangkat untuk mengevaluasi simulasi fisika yang dipercepat GPU pada berbagai konfigurasi perangkat keras. Pendekatan yang digunakan mengombinasikan metode *benchmarking* terkontrol dengan pengujian langsung pada perangkat keras nyata untuk mengidentifikasi batasan performa serta ketergantungan terhadap arsitektur perangkat keras.

Prosedur *benchmarking* disusun untuk mengeksplorasi tiga aspek utama performa simulasi, yaitu: (1) perbandingan efisiensi antara simulasi berbasis CPU dengan simulasi berbasis GPU seiring peningkatan jumlah objek (100–3000 objek), (2) titik ambang performa perangkat keras yang menentukan kapan akselerasi GPU menjadi lebih menguntungkan dibandingkan CPU, serta (3) perbedaan performa antar jenis *collider* (lingkaran, kotak, poligon). Untuk memastikan validitas perbandingan, semua pengujian dilakukan dengan parameter yang konsisten, seperti area fisika tetap sebesar 1000×1000 unit, distribusi massa objek yang distandarisasi (antara 1,0 sampai 5,0 kg), serta dengan fitur sinkronisasi vertikal (V-sync) yang dinonaktifkan.

Protokol pengujian dilakukan melalui beberapa tahapan pengumpulan data:

1. Menjalankan periode pemanasan selama 500 *frames* untuk memastikan kestabilan simulasi.
2. Melakukan pengambilan data utama sebanyak 1800 *frames* untuk setiap peningkatan jumlah objek.
3. Menjalankan uji stres hingga mencapai kapasitas maksimum, yaitu 3000 objek.

Di sisi lain, data performa diperoleh melalui beberapa metode instrumentasi:

1. Pengukuran durasi *frame* menggunakan *unscaled delta time* dari Unity.
2. Pencatatan data mentah dalam format CSV untuk analisis lanjutan secara komparatif.

Metode penelitian ini dirancang secara khusus untuk mencerminkan tantangan riil yang dihadapi oleh pengembang gim, sekaligus tetap mempertahankan standar akademis yang baik. Pendekatan multiperangkat memberikan wawasan praktis bagi para pengembang gim, terutama dalam menentukan spesifikasi minimum perangkat keras pengguna yang akan memperoleh manfaat maksimal dari akselerasi fisika berbasis GPU.

3. PERANCANGAN SISTEM

Landasan utama sistem simulasi fisika yang diusulkan dalam penelitian ini adalah akselerasi GPU melalui *compute shader*. *Compute shader* merupakan program khusus yang dijalankan di GPU dan dirancang untuk mengeksekusi instruksi secara paralel masif (artinya ribuan inti GPU dapat bekerja bersamaan) tanpa harus melewati jalur *rendering* tradisional. Dengan memindahkan beban komputasi berat dari CPU ke GPU, alur kerja fisika menjadi jauh lebih efisien. Sejumlah studi terkini juga mengonfirmasi keunggulan pendekatan ini. Simulasi *ab initio* untuk partikel identik yang dijalankan di GPU, misalnya, menunjukkan peningkatan performa

secara signifikan dibanding CPU klasik (Xiong, 2024).

Implementasi *compute shader* dalam penelitian ini menggunakan bahasa HLSL pada mesin gim Unity. Hasil implementasi ini kompatibel dengan berbagai platform populer seperti Windows (DirectX 11 ke atas), macOS/iOS (Metal), Android/Linux (Vulkan), serta konsol gim modern. Selain itu, implementasi ini juga mendukung platform tambahan seperti OpenGL 4.3 dan OpenGL ES 3.1. Penggunaan *compute shader* memang sudah umum dalam berbagai aplikasi komputasi GPU, termasuk dalam simulasi fisika, sebagaimana dijelaskan dalam studi tentang akselerasi pustaka fisika Bullet Physics (Balamurugan et al., 2016).

Compute shader dalam penelitian ini berinteraksi erat dengan *compute buffer*, yakni area memori khusus yang digunakan untuk bertukar data antara CPU dan GPU. Meskipun *buffer* ini dapat diakses oleh kedua prosesor, operasi yang dilakukan oleh CPU sering kali menimbulkan latensi yang cukup signifikan karena *overhead* komunikasi antar-prosesor. Oleh karena itu, pengelolaan *buffer* yang teliti menjadi krusial, sebagaimana diungkapkan dalam penelitian sebelumnya tentang simulasi dinamika fluida komputasional (CFD) menggunakan OpenFOAM (Piscaglia and Ghioldi, 2023).

Secara umum, simulasi fisika dalam penelitian ini terdiri dari tiga komponen utama yang saling bekerja sama, yaitu *rigidbody*, *collider*, dan *physics system*. *Rigidbody* merupakan representasi dari objek fisik dengan berbagai properti seperti massa, posisi, kecepatan, dan rotasi. *Collider* berfungsi menentukan batas geometris objek, yang diklasifikasikan dalam tiga tipe primitif: lingkaran yang berbasis radius, kotak berbasis ukuran, serta poligon yang berbasis verteks (Liang and Liu, 2015). Adapun *physics system* bertanggung jawab mengelola keseluruhan simulasi dengan memelihara data objek, mengatur operasi *buffer*, menjalankan *compute shader*, serta menyinkronkan hasil simulasi kembali ke memori CPU (Millington, 2007).

Proses simulasi dalam penelitian ini berlangsung dalam tiga tahap utama untuk setiap *frame*, sebagaimana diilustrasikan pada Gambar 1. Pada tahap pertama (inisialisasi), sistem menyiapkan seluruh *buffer* komputasi berdasarkan status objek saat itu, termasuk parameter *rigidbody* (posisi, kecepatan, gaya) dan geometri *collider*. *Collider* berbentuk lingkaran dan kotak menggunakan *buffer* tunggal untuk menyimpan data *offset* dan dimensi, sedangkan *collider* tipe poligon memerlukan *buffer* tambahan yang terpisah untuk menyimpan data verteks/normal serta indeksnya (lihat Tabel 1).

Inti utama komputasi berlangsung pada tahap kedua melalui eksekusi kernel GPU yang berjalan secara paralel. Kernel simulasi fisika bertugas memperbarui dinamika objek berdasarkan gaya dan kecepatan, sedangkan kernel deteksi tumbukan menangani semua kemungkinan pasangan *collider* (total sembilan jenis pasangan). Setiap kernel deteksi tumbukan juga sekaligus menghitung efek restitusi dan gesekan ketika terjadi tumbukan antar objek. Pengalokasian *thread* (proses paralel asinkron) dirancang mengikuti pola paralel data, dengan prioritas eksekusi kernel diberikan pada tipe *collider* yang jumlahnya lebih dominan dalam suatu simulasi (lihat detail di Tabel 2).

Pada tahap akhir, sistem hanya mengirimkan kembali data *rigidbody* yang telah berubah ke memori CPU guna meminimalkan *overhead* komunikasi antar-prosesor. Adapun *buffer* lainnya tetap berada pada GPU karena isinya relatif tidak berubah di sepanjang *frame* simulasi.

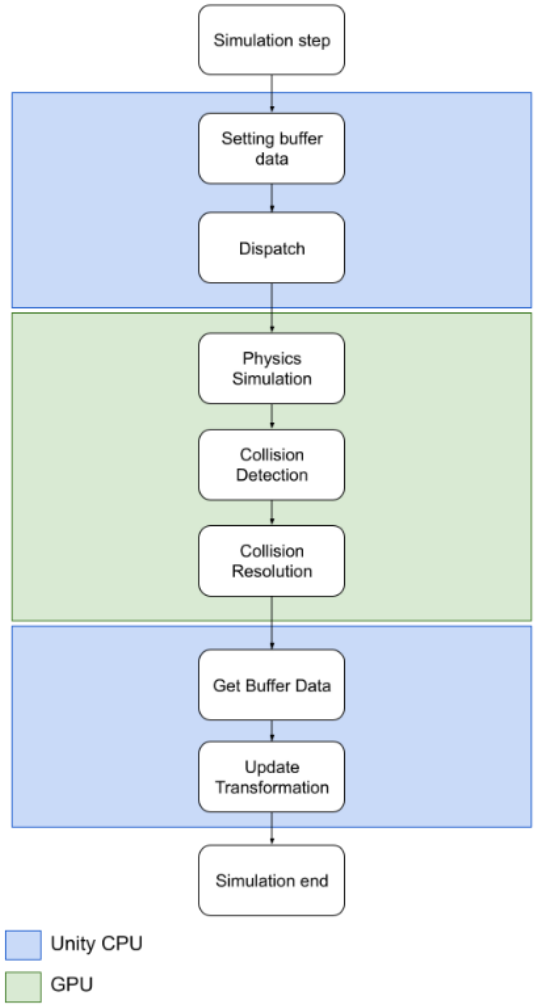
Tabel 1. Atribut *Compute Buffer* untuk Berbagai Jenis *Collider*

No	Nama *Buffer*	Atribut Data
1	*Rigidbody*	*Mass*
		Moment of Inertia
		Position
		Rotation
		Linear Velocity
		Angular Velocity
		Force
		Torque
		Restitution
		Friction Coefficient
2	*Circle*	*Id Rigidbody*
		Offset
		Radius
3	*Box*	*Id Rigidbody*
		Offset
		Size
4	*Vertex*	*Id Rigidbody*
		Offset
		Vertex Start
		Vertex Length
5	*Polygon*	*Position*
		Normal

Tabel 2. Kondisi Eksekusi untuk Berbagai Kernel Simulasi

No	Peran	Nama	Kondisi Eksekusi
1	Simulasi fisika	*Physics Simulation*	Tanpa kondisi
2	Deteksi tumbukan	*Circle-Circle*	Tanpa kondisi
3	Deteksi tumbukan	*Circle-Box*	Ketika lingkaran lebih banyak daripada kotak
4	Deteksi tumbukan	*Box-Circle*	Ketika kotak lebih banyak daripada lingkaran
5	Deteksi tumbukan	*Box-Box*	Tanpa kondisi
6	Deteksi tumbukan	*Circle-Polygon*	Ketika lingkaran lebih banyak daripada poligon
7	Deteksi tumbukan	*Polygon-Circle*	Ketika poligon lebih banyak daripada lingkaran
8	Deteksi tumbukan	*Box-Polygon*	Ketika kotak lebih banyak daripada poligon

No	Peran	Nama	Kondisi Eksekusi
9	Deteksi tumbukan	*Polygon-Box*	Ketika poligon lebih banyak daripada kotak
10	Deteksi tumbukan	*Polygon-Polygon*	Tanpa kondisi

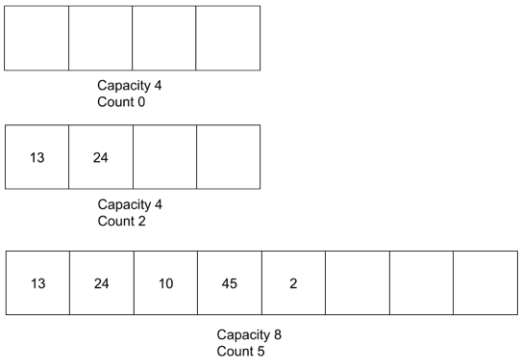


Gambar 1. Simulasi berlangsung melalui tiga fase per *frame*

3.1. Desain *Buffer* Dinamis

Compute buffer dalam Unity umumnya dialokasikan secara statis, sehingga kapasitasnya tidak berubah ketika jumlah objek simulasi meningkat. Untuk mengatasi keterbatasan ini, penelitian menerapkan *buffer* dinamis yang otomatis menggandakan kapasitas (2×) setiap kali terjadi *overflow* sebagaimana diilustrasikan pada Gambar 2. Pendekatan ini menjaga performa simulasi tetap stabil selama ekspansi sekaligus menyediakan fleksibilitas yang memadai untuk menampung pertumbuhan jumlah objek.

Penggandaan kapasitas menurunkan frekuensi realokasi sehingga kompleksitas amortisasi alokasi turun dari $O(n^2)$ menjadi $O(n)$ (Cormen et al., 2022), dan karena realokasi lebih jarang, data simulasi lebih lama bertahan di memori GPU, menekan *cache-miss* serta kontensi memori, faktor penting bagi latensi rendah kernel *real-time* (Aghilinasab et al., 2020). Selain itu, kapasitas *buffer* selalu dijaga sebagai kelipatan dua agar selaras dengan ukuran *memory page* GPU modern sekaligus mempermudah operasi *bitwise* di dalam kernel *shader* (NVIDIA, 2025).



Gambar 2. Pengubahan ukuran *buffer* secara dinamis

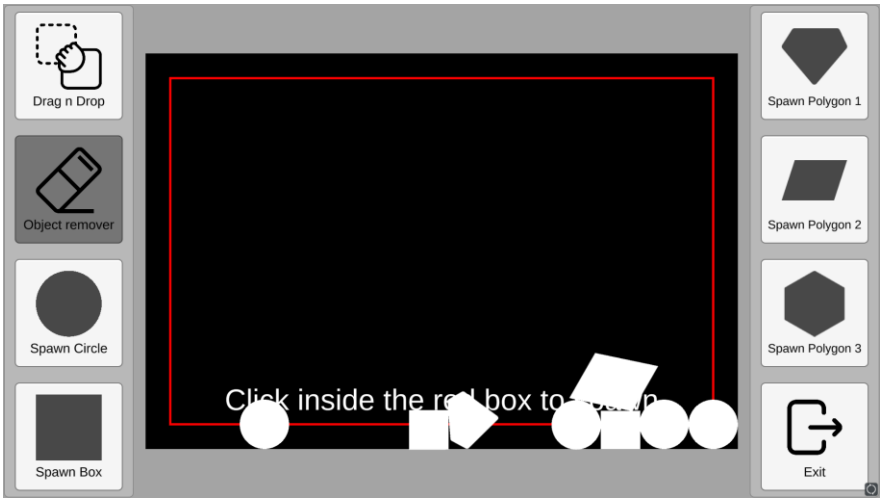
Tabel 3. Kombinasi Skenario *Benchmarking*

No	Tipe *Collider*	Basis Implementasi
1	*Circle*	CPU (Unity *rigidbody*)
2	*Box*	CPU (Unity *rigidbody*)
3	*Polygon*	CPU (Unity *rigidbody*)
4	*Circle*	GPU (*compute shader*)
5	*Box*	GPU (*compute shader*)
6	*Polygon*	GPU (*compute shader*)

3.2. Desain Aplikasi *Benchmarking*

Penelitian ini menggunakan aplikasi *benchmarking* yang dirancang khusus untuk membandingkan performa antara simulasi fisika berbasis CPU (menggunakan Unity *rigidbody*) dengan simulasi berbasis GPU (menggunakan *compute shader*). Perbandingan tersebut dilakukan dalam enam skenario utama, yang melibatkan *collider* tipe lingkaran, kotak, dan poligon pada masing-masing implementasi. Daftar lengkap skenario *benchmarking* ini disajikan pada Tabel 3.

Protokol pengujian dirancang secara sistematis dengan mengambil data sebanyak 1800 *frames* untuk setiap konfigurasi objek, yang jumlahnya bertambah dalam kelipatan 100 objek. Selama pengujian, nilai *unscaled delta time* dari Unity dicatat sebagai dasar perhitungan FPS minimum, maksimum, dan rata-rata. Durasi pengujian akan otomatis menyesuaikan dengan beban komputasi, yakni semakin besar jumlah objek yang disimulasikan, semakin lama pula durasi yang diperlukan untuk pengambilan data sampai selesai. Sebagai pelengkap, aplikasi *benchmarking* dilengkapi dengan fitur *sandbox* sebagaimana ditampilkan pada Gambar 3 untuk memverifikasi secara visual interaksi fisika secara *real-time*, di mana pengguna dapat secara dinamis menempatkan objek dalam area simulasi.



Gambar 3. Fitur *sandbox* untuk membantu visualisasi

4. HASIL DAN PEMBAHASAN

4.1. Konfigurasi Perangkat Uji

Pengujian dilakukan pada lima perangkat keras yang dipilih untuk merepresentasikan beragam spektrum performa laptop konsumen. Pemilihan perangkat ini didasarkan pada tolok ukur standar dari CPU-Monkey[1] (menggunakan skor Cinebench R23) dan UserBenchmark[2] (menggunakan skor Effective 3D Speed). Pendekatan pengambilan sampel bertingkat ini memastikan keragaman CPU, GPU, serta variasi dalam arsitektur komputasi perangkat yang diuji. Spesifikasi lengkap masing-masing perangkat ditampilkan pada Tabel 4, Tabel 5, dan Tabel 6. Kategori Rendah, Sedang, dan Tinggi diadaptasi dari standar distribusi skor terkait. Nilai skor dibagi ke dalam tiga rentang persentil, misal 0–33, 34–66, dan 67–100, pada kategori performa GPU.

4.2. Kategorisasi Hasil *Benchmark*

Berdasarkan analisis performa FPS terhadap peningkatan jumlah objek (dari 100 hingga 3000 objek), teridentifikasi tiga pola utama performa simulasi, yang masing-masing direpresentasikan dalam kategori berikut:

1. *Benchmark category* 1: **CPU unggul pada jumlah objek rendah, GPU mengambil alih pada objek tinggi.** Pada skenario ini, CPU menunjukkan performa lebih baik (*delta time* lebih rendah) saat jumlah objek masih rendah (kurang dari 500 objek), namun performa CPU melambat dan GPU mengungguli seiring bertambahnya objek. Gambar 4 memperlihatkan performa simulasi berbasis GPU yang melampaui CPU ketika jumlah objek mencapai sekitar 2300.
2. *Benchmark category* 2: **GPU secara konsisten lebih unggul.** Dalam pola ini, GPU menunjukkan keunggulan performa secara konsisten dibanding CPU di seluruh rentang jumlah objek, dan margin keunggulannya terus meningkat seiring bertambahnya objek. Gambar 5 menunjukkan performa GPU yang tidak hanya lebih cepat daripada CPU, tetapi juga memiliki laju perubahan waktu simulasi yang lebih stabil.
3. *Benchmark category* 3: ***CPU tetap unggul di semua kondisi.*** Kategori ini menunjukkan bahwa performa CPU tetap lebih baik dibanding GPU di seluruh jumlah objek yang diuji. Sebagaimana ditampilkan oleh Gambar 6, simulasi berbasis GPU tidak mampu melampaui performa CPU, bahkan ketika jumlah objek mencapai 3000.

Distribusi *benchmark category* untuk masing-masing kombinasi perangkat dan jenis *collider* disajikan pada Tabel 7, sementara detail lengkap dalam bentuk matriks dapat dilihat pada Tabel 8.

Benchmark category tidak dibentuk secara arbitrer, tetapi diturunkan dari pola empiris data *benchmark* serta diadaptasi dari pola *crossover* dan ambang batas yang lazim dilaporkan pada penelitian performa CPU/GPU terdahulu (Che et al., 2009; Morsy et al., 2018; Yenpure, Pugmire and Childs, 2025), dan kategorisasi ini semata-mata bertujuan memudahkan pembacaan tabel tanpa memengaruhi kesimpulan utama.

4.3. Korelasi Perangkat dan Performa Hasil

Analisis hubungan spesifikasi CPU–GPU dengan pola performa menunjukkan bahwa perangkat kelas atas, misalnya Laptop 3, menawarkan keseimbangan ideal di antara keduanya. Dibekali CPU bertenaga (Cinebench SC 1427/MC 12179) dan GPU cepat (3D Speed 89.9), sistem ini beralih dari eksekusi CPU ke GPU secara mulus ketika populasi objek mencapai 500–800. Peralihan tersebut menjaga FPS tetap stabil pada seluruh tipe collider, menegaskan bahwa keseimbangan CPU–GPU krusial bagi performa simulasi optimal.

[1] https://www.cpu-monkey.com/en/

[2] https://www.userbenchmark.com/

Tabel 4. Spesifikasi Perangkat (Laptop) untuk Pengujian

No	Spesifikasi CPU	Spesifikasi GPU
1	Ryzen 7 4800H	GeForce RTX 2060 Mobile
2	Ryzen 7 4800HS	GeForce GTX 1650 Mobile
3	Ryzen 7 5800H	GeForce RTX 3060
4	Ryzen 7 2700X	Radeon RX 580 Series
5	Ryzen 5 5500U	Radeon (TM) Graphics

Tabel 5. *Benchmark* CPU Perangkat (Laptop)

No	*Cinebench Single Core R23*	*Cinebench Multi Core R23*	Kategori Performa
1	1242	10943	Sedang
2	1242	10943	Sedang
3	1427	12179	Tinggi
4	1071	9971	Rendah
5	1165	6784	Sedang

Tabel 6. *Benchmark* GPU Perangkat (Laptop)

No	*Effective 3D Speed*	Kategori Performa
1	70.9	Sedang
2	41.2	Sedang
3	89.9	Tinggi
4	51.6	Sedang
5	9.92	Rendah

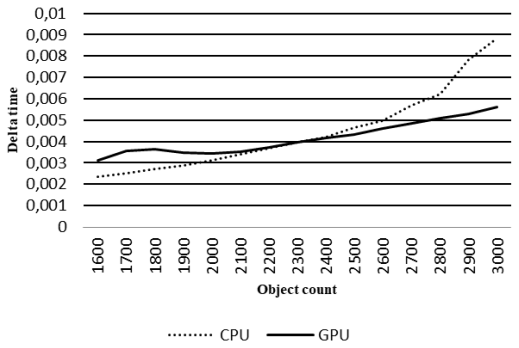
Tabel 7. Klasifikasi Hasil *Benchmark*

Category	Contoh Kasus	Karakteristik
1	Laptop 1 (semua *collider*)	*Crossover point* 500-800 objek
2	Laptop 2 (semua *collider*)	GPU 15-30% lebih cepat
3	Laptop 5 (*collider* poligon)	CPU 2x lebih cepat

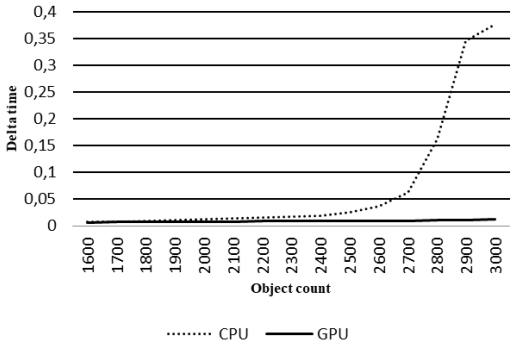
Tabel 8. Matriks *Benchmark Category* Hasil Pengujian

	Circle	Box	Polygon
Laptop 1	*1*	*1*	*1*
Laptop 2	*2*	*2*	*2*
Laptop 3	*1*	*2*	*1*
Laptop 4	*1*	*3*	*3*
Laptop 5	*1*	*1*	*3*

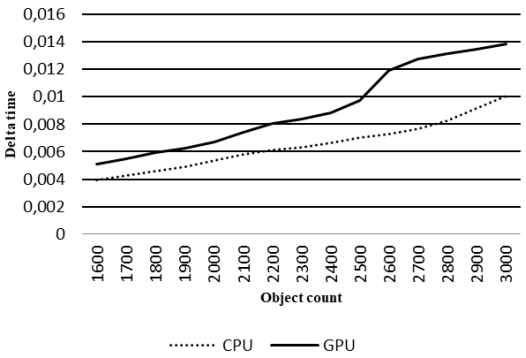
Untuk perangkat dengan spesifikasi menengah tetapi tidak seimbang, seperti Laptop 1 dan 2, performa yang dihasilkan bervariasi tergantung pada kekuatan relatif CPU dan GPU-nya. Laptop 1, yang dilengkapi GPU lebih kuat, menunjukkan pola *benchmark category* 1, yakni CPU unggul pada jumlah objek rendah tetapi GPU mengambil alih pada objek yang lebih banyak. Sebaliknya, Laptop 2 yang memiliki GPU lebih lemah, mengarah kepada *benchmark category* 2, yaitu GPU langsung unggul sejak jumlah objek rendah. Rasio performa CPU-GPU lebih dari 1.5 pada Laptop 1 menunjukkan bahwa semakin besar rasio ini, semakin besar pula jumlah objek yang diperlukan sebelum GPU benar-benar menunjukkan keunggulannya, sekali lagi menegaskan pentingnya keseimbangan antara CPU dan GPU.



Gambar 4. Performa pada uji coba *collider* poligon di Laptop 1



Gambar 5. Performa pada uji coba *collider* poligon di Laptop 2



Gambar 6. Performa pada uji coba *collider* poligon di Laptop 5

Sementara itu, perangkat dengan spesifikasi rendah seperti Laptop 4 dan 5 menunjukkan bahwa GPU yang relatif lemah tidak mampu mengungguli CPU bahkan dalam skenario jumlah objek yang besar sekalipun. Laptop 5, dengan GPU yang sangat lemah (3D Speed 9.92), konsisten berada pada *benchmark category* 3, yaitu CPU tetap unggul di semua jumlah objek. Laptop 4, yang memiliki GPU kelas menengah tetapi CPU lebih lemah, menunjukkan pola performa yang bervariasi tergantung jenis *collider*: GPU hanya mampu unggul untuk *collider* sederhana seperti lingkaran, namun gagal pada *collider* kompleks seperti poligon. Hal ini menegaskan bahwa GPU

dengan skor 3D Speed ≤10 tidak cukup memadai untuk mendukung CPU dalam kondisi beban tinggi.

Kompleksitas *collider* juga memperkuat hubungan antara performa GPU dengan FPS hasil pengujian. *Collider* sederhana seperti lingkaran dan kotak dapat mencapai *benchmark category* 1 atau 2 dengan GPU kelas menengah yang memiliki skor 3D Speed ≥41.2. Namun, untuk *collider* kompleks seperti poligon, hanya GPU *high-end* seperti RTX 3060 yang mampu secara konsisten masuk *benchmark category* 1, sedangkan GPU lain cenderung masuk *benchmark category* 3. Contoh kasus kritis tampak pada Laptop 4 yang menggunakan GPU RX 580, yang meskipun memiliki skor 3D Speed cukup tinggi (51.6), tetap gagal mencapai performa optimal untuk *collider* poligon. Hal ini menunjukkan adanya ketidakefisienan pada optimasi driver GPU AMD dalam menangani komputasi *collider* kompleks.

Persamaan (1) menunjukkan secara matematis korelasi antara *benchmark category* (BenchCat) perangkat keras dan hasil pendataan FPS melalui *delta time* yang dapat dirumuskan sebagai fungsi dari dua variabel utama, yaitu rasio performa CPU berdasarkan Cinebench Single Core (pCPU) terhadap performa GPU berdasarkan skor 3D Speed (pGPU), serta *collider complexity* (CldrComplex).

$$BenchCat = f\left(\frac{pCPU}{pGPU}, CldrComplex\right) \quad (1)$$

Sebagai contoh penerapan rumus ini, *benchmark category* 1 mencakup perangkat dengan rasio performa CPU/GPU antara 15 hingga 25 dengan *collider* sederhana. *Benchmark category* 2 mencakup perangkat dengan rasio CPU/GPU antara 10 hingga 15 dan GPU dengan skor 3D Speed ≥40. Sementara *benchmark category* 3 mencakup perangkat dengan rasio CPU/GPU di atas 30 atau GPU dengan skor 3D Speed <10.

5. KESIMPULAN DAN SARAN

Penelitian ini mengimplementasikan sepuluh kernel *compute-shader* (sembilan deteksi tumbukan dan satu integrator) dilengkapi skema *buffer* dinamis 2×, kemudian mengevaluasinya melalui *benchmark* terkontrol pada lima laptop (100–3000 objek, 500 *frames* pemanasan, 1800 *frames* pengukuran) dengan metrik FPS (latensi per *frame*) melalui *unscaled delta time* yang disediakan oleh Unity.

Berdasarkan hasil implementasi serta evaluasi tersebut, beberapa kesimpulan utama yang dapat ditarik adalah:

1. *Compute shader* terbukti efektif untuk menjalankan simulasi fisika secara paralel dengan *compute buffer* sebagai media penghubung komunikasi data antara CPU dan GPU.
2. Arsitektur sistem simulasi yang terdiri dari tiga komponen utama, yaitu simulasi fisika (*physics simulation*), deteksi tumbukan (*collision detection*), dan resolusi tabrakan (*collision resolution*), memungkinkan optimasi secara modular.
3. Proses transfer data antara CPU dan GPU merupakan tantangan utama dan menjadi *bottleneck* performa, sehingga perlu diminimalkan sebisa mungkin.
4. Meskipun demikian, transfer data kembali ke CPU masih tetap dibutuhkan untuk menjaga aspek interaktivitas pengguna dan kompatibilitas sistem secara keseluruhan.
5. Pola performa simulasi yang diamati menunjukkan adanya transisi keunggulan performa dari CPU ke GPU pada jumlah objek tertentu, yang bergantung pada spesifikasi perangkat keras yang digunakan.

Meskipun cakupan simulasi fisika dua dimensi dalam penelitian ini dibatasi pada perilaku dasar seperti gerak linier dan tumbukan sederhana, pendekatan berbasis GPU yang diterapkan membuka peluang besar untuk pengembangan lebih lanjut. Secara teoritis, arsitektur *compute shader* yang memiliki kemampuan pemrosesan paralel dapat diaplikasikan pada simulasi-simulasi yang lebih kompleks, seperti dinamika fluida secara *real-time*, sistem partikel dengan skala besar, atau simulasi deformasi benda lunak. Tentu saja, implementasi lanjutan tersebut akan tetap membutuhkan solusi untuk mengatasi tantangan utama terkait transfer data CPU-GPU.

Dari sisi praktis, temuan tentang pentingnya keseimbangan spesifikasi CPU-GPU serta optimasi *collider* dapat menjadi panduan awal yang berguna bagi pengembang dalam menentukan batasan kompleksitas simulasi yang sesuai dengan perangkat target pengguna. Penelitian berikutnya disarankan mengeksplorasi pendekatan sistem hibrid yang memanfaatkan keunggulan masing-masing CPU (presisi tinggi) dan GPU (skalabilitas besar), khususnya untuk simulasi fisika dengan kompleksitas yang lebih tinggi.

DAFTAR PUSTAKA

AGHILINASAB, H., ALI, W., YUN, H. AND PELLIZZONI, R., 2020. Dynamic Memory Bandwidth Allocation for Real-Time GPU-Based SoC Platforms. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Institute of Electrical and Electronics Engineers Inc. pp.3348–3360. https://doi.org/10.1109/TCAD.2020.3012210.

ARANGO, R., 2025. *Determinism with 2D physics*. [daring] Unity Support. Tersedia pada: <https://support.unity.com/hc/en-us/articles/360015178512-Determinism-with-2D-Physics> [Diakses 1 April 2025].

BALAMURUGAN, B., SKARIAH, A., NALINIPRIYA, G., MAHESWARI, K.G. AND V., M., 2016. Acceleration of Bullet Physics Physical Simulation Library Using GPU and Demonstration on a Set-Top Box Platform. In: *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*. [daring] New York, NY, USA: ACM. pp.1–4. https://doi.org/10.1145/2905055.2905266.

CATTO, E., 2025. *What is Box2D?* [daring] Box2D. Tersedia pada: <https://box2d.org/documentation/md_faq.html> [Diakses 1 April 2025].

CHE, S., BOYER, M., MENG, J., TARJAN, D., SHEAFFER, J.W., LEE, S.H. AND SKADRON, K., 2009. Rodinia: A benchmark suite for heterogeneous computing. In: *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009*. pp.44–54. https://doi.org/10.1109/IISWC.2009.5306797.

CHOLISSODIN, I., JONEMARO, E.M.A., RAHAYUDI, B., KSATRIA, W.E., SUKMAWATI, A. AND MUZAYYANI, M.F., 2022a. Pengembangan Fast Render Objek Grafis Menggunakan Shader dan Non-Shader Berbasis WebGL dari Primitive Object untuk Membuat Raw Metaverse Material Objek Skybox 3D di Filkom UB. *Jurnal Teknologi Informasi dan Ilmu Komputer*, [daring] 9(7), p.1357. https://doi.org/10.25126/jtiik.2022976739.

CHOLISSODIN, I., SYAUQY, D., FIRMANDA, D.A., AJI, I., RAHMAN, E., HARAHAP, S. AND SEPTINO, F., 2022b. Pengembangan Auto-AI Model Generatif Analisis Kompleksitas Waktu Algoritma Untuk Data Multi-Sensor IoT Pada Node-RED Menggunakan Extreme Learning Machine. *Jurnal Teknologi Informasi dan Ilmu Komputer*, 9(7), p.1349. https://doi.org/10.25126/jtiik.2022976738.

CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L. AND STEIN, C., 2022. *Introduction to algorithms*. MIT press.

IRAWAN, F.T., MA'RUFI, M.R. AND CHOLISSODIN, I., 2017. Optimasi Rendering Game 2D Asteroids Menggunakan Pemrograman CUDA. *Jurnal Teknologi Informasi dan Ilmu Komputer*, [daring] 4(4), pp.221–226. https://doi.org/10.25126/jtiik.201744488.

LEE, R.P., PIERCE, J.R., MILLER, K.G., ALMANZA, M., TABLEMAN, A., DECYK, V.K., FONSECA, R.A., ALVES, E.P. AND MORI, W.B., 2025. Acceleration of the particle-in-cell code OSIRIS with graphics processing units. *Journal of Plasma Physics*, 91(1). https://doi.org/10.1017/S0022377824001569.

LIANG, C. AND LIU, X., 2015. The research of collision detection algorithm based on separating axis theorem. *Int. J. Sci*, 2(10), pp.110–114.

MILLINGTON, I., 2007. *Game physics engine development*. CRC Press.

MORSY, M.M., GOODALL, J.L., O'NEIL, G.L., SADLER, J.M., VOCE, D., HASSAN, G. AND HUXLEY, C., 2018. A cloud-based flood warning system for forecasting impacts to transportation infrastructure systems. *Environmental Modelling and Software*, 107, pp.231–244. https://doi.org/10.1016/j.envsoft.2018.05.007.

NVIDIA, 2025. *CUDA C++ Best Practices Guide*. [daring] Tersedia pada: <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/> [Diakses 6 June 2025].

PENG, X., ZHENG, Z., SHEN, C., YOUNG, T., GUO, X., WANG, B., XU, H., LIU, H., JIANG, M., LI, W., WANG, Y., YE, A., REN, G., MA, Q., LIANG, W., LIAN, X., WU, X., ZHONG, Y., LI, Z., GONG, C., LEI, G., CHENG, L., ZHANG, L., LI, M., ZHANG, R., HU, S., HUANG, S., WANG, X., ZHAO, Y., WANG, Y., WEI, Z. AND YOU, Y., 2025. Open-Sora 2.0: Training a Commercial-Level Video Generation Model in $200k. [daring] pp.1–21. Tersedia pada: <http://arxiv.org/abs/2503.09642>.

PISCAGLIA, F. AND GHIOLDI, F., 2023. GPU Acceleration of CFD Simulations in OpenFOAM. *Aerospace*, [daring] 10(9), p.792. https://doi.org/10.3390/aerospace10090792.

RIZALDI, H.I., PRAMANA, F.S., R, B.N., A. N., A.Y. AND CHOLISSODIN, I., 2017. Optimasi Proses Rendering Objek Game 3D Menggunakan Pemrograman CUDA Pada Game Sandbox Craft. *Jurnal Teknologi Informasi dan Ilmu Komputer*, [daring] 4(3), p.207. https://doi.org/10.25126/jtiik.201743444.

SUNG, N.J., MA, J., HOR, K., KIM, T., VA, H., CHOI, Y.J. AND HONG, M., 2025. Real-Time Physics Simulation Method for XR Application. *Computers*, 14(1). https://doi.org/10.3390/computers14010017.

XIONG, Y., 2024. GPU acceleration of ab initio simulations of large-scale identical particles based on path integral molecular dynamics.

[daring] pp.1–23. Tersedia pada: <http://arxiv.org/abs/2404.02628>.

XU, A. AND LI, B.T., 2024. Particle-resolved thermal lattice Boltzmann simulation using OpenACC on multi-GPUs. *International Journal of Heat and Mass Transfer*, 218(October 2023). https://doi.org/10.1016/j.ijheatmasstransfer.2023.124758.

YENPURE, A., PUGMIRE, D. AND CHILDS, H., 2025. Steady-State Particle Advection Speed-Ups from GPU and CPU Parallelism. In: *IS and T International Symposium on Electronic Imaging Science and Technology*. Society for Imaging Science and Technology. https://doi.org/10.2352/EI.2025.37.12.HPCI-179.

Halaman ini sengaja dikosongkan