Journal of Applied Multimedia and Networking (JAMN)
Vol.x, No.x, Maret 201x, pp. xx~xx
e-ISSN: 2548-6853

ANALISIS EFEKTIVITAS PENGGUNAAN *TEXTURE ATLAS* TERHADAP KINERJA *RENDERING* ASET 3D *LOW-POLY* PADA *GAME* ENGINE

Muhammad Fauzul Azim Mahfudz*, Liony Lumombo**
* Animasi, Batam State Polytechnic
** Animasi, Batam State Polytechnic

Article Info

Article history:

Received Jun 12th, 201x
Revised Aug 20th, 201x
Accepted Aug 26th, 201x

Keyword:

Texture Atlas
Rendering
Draw Call
Frames Per Second
Asset 3D *Low-poly*

ABSTRACT

The 3D game industry continues to evolve alongside the increasing demand for high visual quality and optimal performance across various hardware platforms. One of the primary challenges in developing low-poly 3D assets is the high number of draw calls caused by the use of separate textures for each object, which can reduce frame rate stability and rendering efficiency. This study aims to analyze the effectiveness of using a Texture Atlas to improve the rendering performance of low-poly 3D assets in Unity. A quantitative research method with an experimental approach was employed by comparing two texturing techniques, namely Texture Atlas and Separate Textures, under identical scene conditions. Performance evaluation was conducted using Unity's Stats feature based on two parameters: Frames Per Second (FPS) and Draw Calls. Measurements were performed at four predefined camera positions, with five repetitions for each position. The results indicate that the Texture Atlas technique produces fewer Draw Calls and more stable FPS values than the Separate Texture approach. Therefore, the Texture Atlas technique is proven to be more efficient in improving the rendering performance of low-poly 3D assets in Unity.

Corresponding Author:

Third Author,
Departement of Electrical and Computer Engineering,
National Chung Cheng University,
168 University Road, Minhsiung Township, Chiayi County 62102, Taiwan, ROC.
Email: lsntl@ccu.edu.tw

1. INTRODUCTION

Industri *game* global mengalami pertumbuhan yang sangat signifikan pada tahun 2024. Pasar video *game* global diproyeksikan mencapai pendapatan sebesar US$282,3 miliar pada 2024 dan pasar video *game* akan tumbuh sebesar 8,76% year-to-year (YoY) antara tahun 2024 hingga 2027 [1]. Pertumbuhan ini mendorong pengembang *game* untuk tidak hanya menghadirkan kualitas visual yang menarik, tetapi juga memastikan performa *game* tetap optimal agar dapat berjalan dengan baik pada berbagai perangkat. Urgensi optimasi performa semakin tinggi mengingat sebagian besar pengembang *game indie* dan mahasiswa mengembangkan proyek dengan sumber daya perangkat keras terbatas serta tim yang kecil, sehingga efisiensi teknis seperti manajemen *Texture* menjadi salah satu faktor penentu apakah sebuah *game* dapat berjalan lancar tanpa memerlukan investasi perangkat keras tambahan [2].

Pada pengembangan *game* 3D, performa *rendering* dipengaruhi oleh berbagai faktor, salah satunya adalah pengelolaan *Texture* yang digunakan pada aset dalam suatu *scene* [3]. Penggunaan *Texture* yang terpisah pada setiap objek dapat meningkatkan jumlah *Draw Call* yang diproses oleh *game engine* sehingga berpotensi menurunkan nilai *Frames Per Second* (FPS) dan efisiensi *rendering* secara keseluruhan. *Draw Call* merupakan perintah yang dikirim oleh CPU ke GPU untuk merender objek, dan jumlah *Draw Call* yang tinggi dapat

menyebabkan *bottleneck* pada sistem [4]. Maka dari itu, diperlukan teknik optimasi yang mampu mengurangi beban *rendering* tanpa mengurangi kualitas visual yang ditampilkan.

Salah satu teknik optimasi yang umum digunakan adalah *Texture Atlas*, yaitu metode penggabungan beberapa *Texture* ke dalam satu file *Texture* sehingga beberapa objek dapat menggunakan material yang sama [5]. Dengan menggabungkan banyak *Texture* ke dalam satu *Texture* 2D (*Texture Atlas*), beberapa objek dapat dirender dalam satu *Draw Call* [6]. Permasalahan *Draw Call* menjadi semakin relevan pada *environment* 3D *low-poly* karena karakteristik asetnya sendiri, *environment low-poly* umumnya disusun dari banyak objek kecil dan terpisah (bangunan, pagar, properti, vegetasi) yang secara konvensional masing-masing diberi *Texture* dan material sendiri. Semakin banyak objek dengan material berbeda, semakin banyak pula *Draw Call* yang harus dikirim CPU ke GPU meskipun jumlah *polygon* per objek relatif rendah. Kondisi ini membuat performa *rendering* pada *scene low-poly* padat objek justru dapat terhambat, bukan oleh kompleksitas geometri, melainkan oleh jumlah *Draw Call* sebuah *CPU-bound bottleneck* yang kerap luput dari perhatian karena asumsi umum bahwa aset *low-poly* "ringan" untuk direnderer [7]. Hal ini menjadi pertimbangan penting terutama bagi pengembang *game indie* atau mahasiswa yang mengembangkan *game* pada perangkat dengan spesifikasi terbatas. Shen dkk. menunjukkan bahwa optimasi *rendering Texture* berdasarkan tingkat kepentingan *vertex* dapat mengurangi jumlah data *Texture* sekaligus menjaga efisiensi *rendering* model tiga dimensi [8].

Dengan demikian, masih terdapat celah penelitian berupa minimnya bukti empiris mengenai efektivitas *Texture Atlas* yang diukur langsung menggunakan parameter performa *game engine* (FPS dan *Draw Call*) pada *environment* 3D *low-poly* bertema *game*, dengan variasi posisi kamera dan perangkat pengujian. Penelitian ini berkontribusi mengisi celah tersebut dengan menyajikan perbandingan kuantitatif langsung antara *Texture Atlas* dan *Texture* Terpisah pada Unity, sekaligus memverifikasi konsistensi hasil pada beberapa perangkat keras berbeda, sehingga dapat menjadi rujukan praktis bagi pengembang *game low-poly* dalam menentukan strategi manajemen *Texture*.

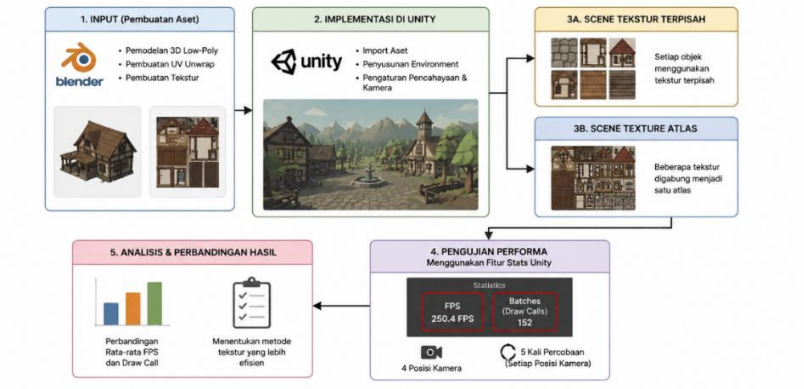
2. RESEARCH METHOD

Penelitian ini menggunakan metode kuantitatif dengan pendekatan eksperimen untuk membandingkan performa *rendering* antara penggunaan *Texture Atlas* dan *Texture* Terpisah pada *environment* 3D *low-poly* di Unity. Metode ini digunakan untuk memperoleh data yang terukur dan objektif berdasarkan parameter *Frames Per Second* (FPS) dan *Draw Call*.

Penelitian ini terdiri dari dua tahap utama yaitu tahap produksi aset yang mengikuti alur kerja standar pengembangan aset 3D *game* (non-eksperimental, bersifat teknis produktif), dan tahap pengujian performa yang menggunakan pendekatan eksperimen kuantitatif untuk membandingkan dua kondisi *Texture*. Dengan demikian, metode eksperimen pada penelitian ini secara spesifik diterapkan pada tahap evaluasi performa, bukan pada keseluruhan proses penelitian.

2.1. Research Stages

Tahapan penelitian dimulai dari pembuatan aset 3D *low-poly* menggunakan Blender, dilanjutkan dengan proses *UV Mapping* dan *Texturing* untuk menghasilkan dua skenario pengujian. Aset yang telah dibuat kemudian diimplementasikan ke dalam Unity untuk membangun *environment* pengujian. Setelah proses implementasi selesai, dilakukan pengujian performa menggunakan fitur *Stats* pada Unity. Alur tahapan penelitian ditunjukkan pada Gambar 1.



Gambar 1. Tahapan Penelitian

2.2. Research Object

Environment penelitian terdiri dari 23 aset unik, meliputi bangunan (rumah, toko, gereja, kincir angin), fasilitas pendukung (pagar, kandang, market stall), serta elemen alam (pohon, batu, rumput), dengan total 313 instance objek dengan total 611.670 *vertex* pada keseluruhan *environment*. Setiap aset dirancang dengan rentang 1.000 – 1.500 *vertex* per-objek agar konsisten dengan standar aset *low-poly*. Pada kedua skenario pengujian, seluruh *Texture* diimpor dengan resolusi yang sama, yaitu 2048×2048 piksel (Max Size), untuk menjaga konsistensi variabel resolusi *Texture* antar-skenario. Perbedaan utama terletak pada jumlah file dan pengelolaan material pada skenario *Texture* Terpisah, setiap objek menggunakan file *Texture* dan material tersendiri, sedangkan pada skenario *Texture Atlas*, *Texture* beberapa objek digabungkan ke dalam beberapa file *atlas* yang dikelompokkan berdasarkan kategori objek (misalnya *atlas* bangunan, *atlas* fasilitas market, dan *atlas* elemen alam), sehingga objek dalam kategori yang sama dapat berbagi material yang lebih sedikit dibandingkan skenario *Texture* Terpisah. Tema *medieval village* dipilih karena merepresentasikan *environment* dengan kepadatan objek dan variasi material yang tinggi, kondisi yang umum dijumpai pada *game* eksplorasi atau simulasi, sehingga relevan untuk menguji efektivitas *Texture Atlas* pada kasus penggunaan nyata.

Penelitian ini tidak menggunakan *game* yang telah dirilis secara komersial sebagai studi kasus, melainkan *environment* prototipe yang dirancang dan dibangun khusus oleh peneliti untuk kebutuhan pengujian, guna memastikan kontrol penuh terhadap variabel penelitian (jumlah objek, material, dan pencahayaan yang identik pada kedua skenario). Secara visual, *environment* ini mengadopsi gaya *stylized low-poly* dengan bentuk geometris disederhanakan, proporsi non-realistis, dan *palet* warna hangat bernuansa *hand-painted* dengan gaya yang umum dijumpai pada *game indie* bergenre eksplorasi maupun simulasi kehidupan desa (*village-life simulation*). Pendekatan prototipe mandiri ini dipilih agar hasil pengujian murni mencerminkan pengaruh metode *Texture*, tanpa dipengaruhi elemen gameplay atau kompleksitas produksi *commercial game* yang berada di luar cakupan penelitian.

2.2.1. Researcher's Role

Seluruh proses produksi aset pada penelitian ini dikerjakan oleh peneliti utama, meliputi pemodelan 3D *low-poly* di Blender, proses *UV Mapping*, pembuatan *Texture Atlas* dan *Texture* Terpisah, implementasi ke Unity, pengujian dan pengambilan data, hingga analisis statistik. Penulis kedua (Liony Lumombo) berperan sebagai dosen pembimbing penelitian, memberikan arahan metodologis, masukan teknis, serta peninjauan terhadap proses dan hasil penelitian selama penyusunan artikel ini. Proses produksi aset mengikuti alur kerja standar pembuatan aset *game low-poly*: *Modeling* → *UV Unwrapping* → *Texturing* → implementasi *engine*, sebagaimana ditunjukkan pada Gambar 1.

2.3. Data Collection

Pengumpulan data dilakukan melalui pengujian langsung menggunakan fitur *Stats* pada Unity. Data yang diperoleh berupa nilai FPS dan *Draw Call* dari masing-masing skenario pengujian. Pengambilan data dilakukan sebanyak lima kali pada setiap posisi kamera untuk memperoleh nilai rata-rata yang lebih representatif.

Pengulangan sebanyak lima kali pada setiap posisi kamera dilakukan untuk mengurangi bias akibat fluktuasi performa sesaat (*frame-time jitter*) yang lazim terjadi pada pengukuran real-time *rendering*, sekaligus memungkinkan perhitungan nilai rata-rata yang lebih stabil sebagai representasi kondisi sebenarnya. Jumlah pengulangan ini merujuk pada praktik umum pengujian performa *game engine* yang menekankan pengambilan sampel berulang pada kondisi kontrol yang identik untuk meminimalkan *noise* pengukuran.

2.4. Testing Devices

Pengujian dilakukan menggunakan satu perangkat utama dan dua perangkat pembanding. Perangkat utama digunakan untuk pengujian utama pada *Unity Editor*, sedangkan perangkat pembanding digunakan untuk pengujian tambahan melalui aplikasi hasil *build* (*.exe*). Spesifikasi perangkat yang digunakan pada penelitian ini ditunjukkan pada Tabel 1.

Tabel 1. Perangkat Pengujian

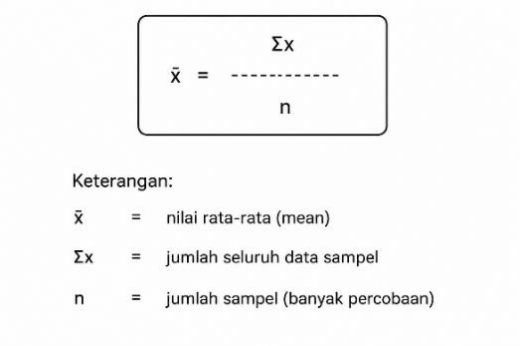
No.	Komponen	Spesifikasi		
		Perangkat Utama	**Perangkat Pembanding 1**	**Perangkat Pembanding 1**
1.	Nama Device	MSI Thin A15 B7VF	AFTERSHOCK MX pro	HP Pavilion 15
2.	Processor	AMD Ryzen 7 7735HS	Intel Core i7-7700HQ	AMD Ryzen 5 5500U
3.	GPU	NVIDIA GeForce RTX 4060	NVIDIA GeForce GTX 1060 6GB	AMD Radeon Vega integrated
4.	RAM	16GB	16GB	16GB

5.	Sistem Operasi	Windows 11	Windows 11	Windows 11

2.5. Measurement Parameters

Parameter pengukuran pada penelitian ini terdiri dari FPS dan *Draw Call*. Pemilihan FPS dan *Draw Call* sebagai parameter ukur didasarkan pada perannya sebagai indikator standar performa *rendering* real-time. FPS merepresentasikan pengalaman visual pengguna secara langsung, sedangkan *Draw Call* merepresentasikan beban kerja CPU dalam mengirim perintah render ke GPU sehingga menjadi indikator paling relevan untuk mengukur dampak penggabungan *Texture* terhadap efisiensi pipeline *rendering* [9].

Pengukuran dilakukan pada empat posisi kamera yang telah ditentukan untuk merepresentasikan sudut pandang yang berbeda pada *environment*. Setiap posisi diuji sebanyak lima kali percobaan dan dihitung menggunakan Rumus Mean (Rata-rata Aritmatika) untuk menghitung nilai rata-rata FPS.



Gambar 2. Rumus Mean

Seluruh proses analisis statistik pada penelitian ini, meliputi uji normalitas (Shapiro-Wilk) dan uji-t berpasangan (*paired sample t-test*), dilakukan menggunakan perangkat lunak JASP. Penggunaan JASP dimaksudkan untuk menjaga transparansi proses analisis data serta memudahkan replikasi hasil penelitian oleh peneliti lain.

2.6. Scope and Limitations of the Experiment

Rendering yang dimaksud dalam penelitian ini adalah real-time *rendering*, yaitu proses render yang dilakukan secara langsung oleh Unity menggunakan *Universal Render Pipeline* (URP) saat aplikasi berjalan (baik pada mode Play di *Unity Editor* maupun pada hasil *build .exe*), bukan render gambar statis (offline *rendering*) seperti pada software render seperti Blender *Cycles*. Batasan eksperimen pada penelitian ini meliputi:

1. *Environment* berjumlah 313 objek dengan total 611.670 *vertex*.
2. *Texture* Terpisah maupun *Texture Atlas* menggunakan resolusi impor yang sama, yaitu 2048×2048 piksel (Max Size), sehingga variabel resolusi *Texture* terkontrol identik pada kedua skenario dan perbedaan performa yang terukur murni disebabkan oleh metode pengelolaan *Texture*.
3. Pengujian dilakukan pada *scene* statis tanpa elemen gameplay interaktif (tidak ada *physics simulation*, AI, maupun animasi karakter yang berjalan bersamaan).
4. Pencahayaan menggunakan *Mixed Lighting* dengan *Baked Global Illumination* aktif dan *Lighting Mode: Shadowmask* (pencahayaan tidak langsung/indirect dipanggang ke dalam *lightmap*, sedangkan pencahayaan langsung/direct dan bayangan tetap dihitung secara real-time), diterapkan identik pada kedua skenario.
5. Pengujian dilakukan pada resolusi layar 1920×1080 (Full HD).
6. *V-Sync* dinonaktifkan selama pengujian (*VSync Count: Don't Sync*), sehingga nilai FPS yang terukur tidak dibatasi oleh *refresh rate* monitor.
7. Fitur *batching* yang aktif adalah *Static Batching* (Player Settings) dan *SRP Batcher (URP Asset),* sedangkan *Dynamic Batching* tidak diaktifkan. Kombinasi ini konsisten dengan praktik umum pada proyek URP, di mana *SRP Batcher* menggantikan peran *Dynamic Batching* versi lama.

3. RESULTS AND ANALYSIS

3.1. Testing Scenario

Untuk menjaga konsistensi dan validitas data, penelitian ini menggunakan skenario pengujian dengan variabel kontrol yang sama pada setiap percobaan. Rincian skenario pengujian tersebut ditunjukkan pada Tabel 2.

Tabel 2. Skenario Pengujian

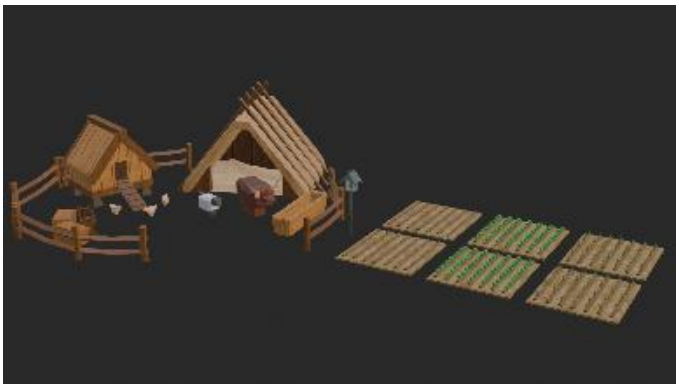
No.	Komponen	Keterangan
1.	Engine	Unity
2.	Objek	Sama pada kedua skenario
3.	Lighting	Sama
4.	Posisi Kamera	4 posisi sama
5.	Jumlah Sampel	5 kali pada setiap posisi
6.	Parameter	FPS dan *Draw Call*
7.	Metode 1	*Texture Atlas*
8.	Metode 2	*Texture* Terpisah

3.2. Asset Implementation

Pada tahap implementasi, aset 3D *low-poly* dibuat menggunakan Blender sesuai konsep *environment* bertema *medieval village*. Aset yang dibuat meliputi bangunan, pagar, batu, pohon, dan berbagai elemen pendukung lainnya. Seluruh aset dirancang dengan gaya *low-poly* untuk menjaga efisiensi jumlah *polygon* dan mendukung proses optimasi *rendering*. Hasil implementasi aset 3D *low-poly* yang digunakan dalam penelitian ini dapat dilihat pada Gambar 3-5.

Gambar 3. 3D Aset Rumah

Gambar 4. 3D Aset *Market*



Gambar 5. 3D Aset *Farm*

3.3. *Texture* Implementation

Penyusunan *Texture Atlas* pada penelitian ini menggunakan beberapa file *atlas* yang dikelompokkan berdasarkan kategori objek (misalnya *atlas* bangunan untuk rumah, *atlas* toko untuk aset market, serta *atlas* tersendiri untuk elemen alam seperti pohon, batu, dan rumput), bukan satu file *atlas* tunggal untuk seluruh *environment*. Pengelompokan per kategori ini dipilih agar *Texture* dengan karakteristik visual serupa (skema warna, tingkat detail, dan skala penggunaan pada objek) dikelola dalam satu ruang *atlas* yang sama, sehingga menjaga konsistensi *texel density* antar objek dalam kategori yang sama.

Setiap file *atlas* disusun menggunakan algoritma *UV Packing* otomatis pada Blender melalui operator *Pack Islands*, dengan metode *Exact Shape* (*Concave*) yang memaketkan setiap *UV Island* berdasarkan kontur bentuk aslinya (bukan kotak pembatas sederhana) untuk memaksimalkan efisiensi pemakaian ruang *atlas*. Pendekatan packing berbasis bentuk aktual (*shape-based packing*) dilaporkan menghasilkan rasio pemakaian ruang *atlas* yang lebih tinggi dibandingkan pendekatan heuristik berbasis kotak pembatas sederhana, terutama pada kasus *UV Island* dengan bentuk tidak beraturan [10]. Opsi Scale dan Rotate (metode rotasi Any) turut diaktifkan sehingga Blender dapat menskalakan dan merotasikan setiap island secara otomatis untuk menghasilkan susunan sepadat mungkin. Margin antar-island ditetapkan sebesar 0,001 (metode Scaled) sebagai celah pencegah *Texture Bleeding*, fenomena percampuran warna antar-*Texture* yang bertetangga akibat proses *mipmap filtering* saat objek dirender dari jarak jauh. Meskipun terdiri dari beberapa file, setiap objek pada skenario *Texture Atlas* tetap menggunakan lebih sedikit material unik dibandingkan skenario *Texture* Terpisah (di mana setiap objek individual memiliki file *Texture*-nya sendiri), sehingga prinsip pengurangan *Draw Call* melalui *batching* tetap berlaku di dalam masing-masing kelompok kategori. Efisiensi penyusunan berbasis bentuk aktual ini turut mendukung efisiensi pembacaan *Texture* oleh GPU saat proses *rendering*, sebagaimana dibahas pada subbab 3.10.

3.3.1. Material Configuration

Shader Universal Render Pipeline/*Lit* pada Unity menyediakan beberapa slot *Texture* pada bagian *Surface Inputs*, meliputi *Base Map*, *Metallic*, *Smoothness*, *Normal Map*, *Height Map*, dan *Occlusion Map* [11]. Material pada penelitian ini dikonfigurasi menggunakan *Shader Universal Render Pipeline*/*Lit* pada Unity, dengan hanya memanfaatkan slot *Base Map* (setara *Albedo*/warna dasar) sebagai satu-satunya peta *Texture* yang diterapkan pada seluruh objek, slot *Texture* lainnya pada material dibiarkan kosong (None). Slot *Texture* lain yang tersedia pada *shader* tersebut, seperti *Metallic* (tingkat kemiripan permukaan dengan logam), *Smoothness* (tingkat kehalusan permukaan yang memengaruhi pantulan cahaya), *Normal Map* (detail permukaan mikro tanpa menambah geometri), *Height Map* (simulasi kedalaman permukaan melalui parallax), dan *Occlusion Map* (bayangan kontak antar-permukaan), sengaja tidak digunakan dengan dua pertimbangan utama. Pertama, pertimbangan gaya visual: aset *low-poly* yang dikembangkan mengusung gaya *stylized flat-shaded*, di mana detail permukaan mikro dan efek material berbasis PBR (*Physically Based Rendering*) justru bertentangan dengan estetika *low-poly* yang mengandalkan bidang datar (*flat faceted*) sebagai ciri khas visualnya, penambahan *Normal Map* atau *Height Map* akan menampilkan detail permukaan yang tidak konsisten dengan gaya geometri yang disederhanakan [12]. Kedua, pertimbangan kontrol variabel penelitian seperti penggunaan peta *Texture* tambahan akan menambah jumlah *Texture sampler* yang harus diproses GPU per objek, sehingga berpotensi menjadi variabel perancu (*confounding variable*) yang memengaruhi *Draw Call* dan FPS di luar variabel utama yang diuji, yaitu metode penggabungan *Texture* (*Atlas* vs Terpisah) [13]. Dengan membatasi material hanya pada satu peta *Base Map*/*Albedo*, penelitian ini memastikan bahwa perbedaan performa yang terukur murni disebabkan oleh perbedaan metode pengelolaan *Texture*, bukan oleh

kompleksitas material itu sendiri. Perbedaan penerapan metode *Texture Atlas* dan *Texture* Terpisah pada aset penelitian ditunjukkan pada Tabel 3.

Tabel 3. Implementasi *Texture*

Texture Atlas	*Texture* Terpisah
	

3.4. *UV Mapping* Implementation

Setelah proses pembuatan *Texture* selesai, dilakukan proses *UV Mapping* pada setiap objek untuk mengatur koordinat *Texture* pada model 3D. Proses ini bertujuan untuk memastikan *Texture* dapat diterapkan dengan tepat pada permukaan objek tanpa distorsi. Pada tahap ini dilakukan proses *unwrap* dan penyesuaian susunan *UV Islands* agar penggunaan ruang *Texture* lebih efisien. Proses implementasi *UV Mapping* pada kedua metode *Texture* ditunjukkan pada Tabel 4.

Tabel 4. Implementasi *UV Mapping*

Texture Atlas	***Texture* Terpisah**
	

3.5. *Texture* Comparison

Perbandingan visual antara metode *Texture Atlas* dan *Texture* Terpisah ditampilkan untuk menunjukkan perbedaan struktur *Texture* yang digunakan dalam penelitian. Kedua metode dirancang untuk mempertahankan kualitas visual aset yang sama, sehingga perbedaan yang diuji berfokus pada efisiensi pengelolaan *Texture* dan performa *rendering*. Visualisasi kedua metode tekstur ditunjukkan pada Tabel 5.

Tabel 5. Perbandingan Visual *Texture Atlas* dan *Texture* Terpisah

No.	**Nama Objek**	***Texture Atlas***	***Texture* Terpisah**
1.	Rumah		

2.	Toko Buah		
3.	Kandang		
4.	Rumput, Pohon, Batu		

Berdasarkan Tabel 5, metode *Texture Atlas* menggabungkan beberapa elemen *Texture* ke dalam satu file tanpa mengurangi detail visual yang ditampilkan pada aset. Dengan pendekatan ini, beberapa objek dapat menggunakan material yang sama sehingga jumlah proses *rendering* dapat dikurangi. Sementara itu, metode *Texture* Terpisah menggunakan file *Texture* yang berbeda pada setiap objek dengan kualitas visual yang setara. Perbedaan struktur tersebut menjadi dasar utama dalam pengujian untuk melihat pengaruhnya terhadap performa *rendering* pada Unity.

3.6. *Scene* Implementation

Aset yang telah melalui proses *Texturing* kemudian diimplementasikan ke dalam Unity untuk membangun *environment* pengujian. *Scene* dibuat dengan kondisi yang sama pada kedua metode, meliputi jumlah objek, pencahayaan, tata letak, dan material. Perbedaan utama hanya terletak pada metode *Texture* yang digunakan. Untuk proses pengujian, ditentukan empat posisi kamera yang digunakan sebagai titik pengambilan data. Posisi kamera dipilih untuk merepresentasikan sudut pandang yang berbeda dalam *scene*. Posisi kamera yang digunakan sebagai titik pengambilan data pengujian ditunjukkan pada Tabel 6.

Tabel 6. Implementasi Letak Kamera

No.	Kamera	Viualisasi Kamera
1.	Kamera 1	

2.	Kamera 2	
3.	Kamera 3	
4.	Kamera 4	

3.7. Performance Testing Result

Pengujian dilakukan menggunakan fitur *Stats* pada Unity untuk memperoleh data FPS dan *Draw Call*. Setiap posisi kamera diuji sebanyak lima kali pada masing-masing metode *Texture* untuk memperoleh nilai rata-rata. Hasil pengujian performa *rendering* pada seluruh posisi kamera ditunjukkan pada Tabel 7.

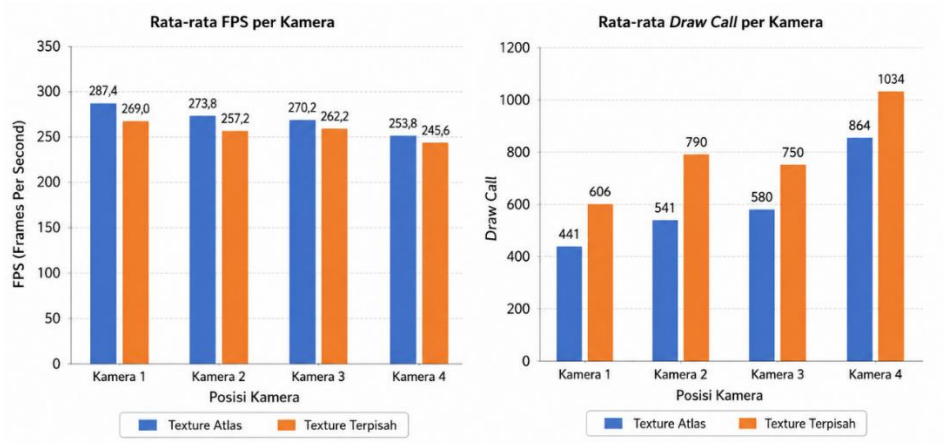
Tabel 7. Data Pengujian pada Unity

Kamera	Metode *Texture*	Parameter	Sampel 1	Sampel 2	Sampel 3	Sampel 4	Sampel 5	Rata-rata
Kamera 1	*Texture Atlas*	FPS	286	284	287	291	289	287,4
		Draw Call	441	441	441	441	441	441
	Texture Terpisah	FPS	270	269	270	268	268	269
		Draw Call	606	606	606	606	606	606
Kamera 2	*Texture Atlas*	FPS	273	275	275	273	273	273,8
		Draw Call	541	541	541	541	541	541
	Texture Terpisah	FPS	260	258	255	260	253	257,2
		Draw Call	790	790	790	790	790	790
Kamera 3	*Texture Atlas*	FPS	270	271	270	271	269	270,2
		Draw Call	580	580	580	580	580	580
	Texture Terpisah	FPS	263	260	263	265	260	262,2
		Draw Call	750	750	750	750	750	750
Kamera 4	*Texture Atlas*	FPS	253	255	253	256	252	253,8
		Draw Call	864	864	864	864	864	864
	Texture Terpisah	FPS	247	245	246	245	245	245,6
		Draw Call	1034	1034	1034	1034	1034	1034

Berdasarkan data pada Tabel 7, setiap posisi kamera menunjukkan variasi nilai FPS dan *Draw Call* yang berbeda. Perbedaan ini dipengaruhi oleh jumlah objek yang berada dalam area pandang kamera serta tingkat kompleksitas visual pada setiap posisi pengujian. Posisi dengan cakupan objek yang lebih padat cenderung menghasilkan beban *rendering* yang lebih tinggi, sedangkan posisi dengan objek yang lebih sedikit menunjukkan kondisi *rendering* yang lebih ringan. Data tersebut menjadi dasar untuk melihat pola performa pada tahap visualisasi dan analisis berikutnya.

3.8. Average Performance Graph

Untuk mempermudah interpretasi hasil pengujian, data rata-rata FPS dan *Draw Call* pada setiap posisi kamera divisualisasikan dalam bentuk grafik. Visualisasi ini memberikan gambaran yang lebih jelas mengenai pola perubahan performa *rendering* pada kedua metode *Texture*. Grafik rata-rata hasil pengujian ditunjukkan pada Gambar 6.



Gambar 6. Grafik Rata-rata Performa

Pada Gambar 6, terlihat pola perubahan nilai FPS dan *Draw Call* pada setiap posisi kamera. Nilai FPS menunjukkan kecenderungan menurun pada posisi dengan kompleksitas visual yang lebih tinggi, sementara nilai *Draw Call* mengalami peningkatan seiring bertambahnya jumlah objek yang dirender. Pola tersebut menunjukkan adanya hubungan antara kompleksitas *scene* dengan beban *rendering* yang dihasilkan selama pengujian.

3.9. Performance Testing on Comparison Devices

Untuk melihat konsistensi performa pada kondisi penggunaan nyata, pengujian tambahan dilakukan pada perangkat pembanding menggunakan hasil *build* aplikasi dalam format *.exe*. Pada tahap ini, parameter yang diukur hanya berupa nilai FPS karena pengujian dilakukan di luar Unity Editor. Hasil rata-rata FPS pada perangkat pembanding ditunjukkan pada Tabel 8.

Tabel 8. Hasil Pengujian FPS pada Perangkat Pembanding

No.	Perangkat	*Texture Atlas* (FPS)	*Texture* Terpisah (FPS)
1.	AFTERSHOCK MX Pro Series	140 FPS	130 FPS
2.	HP Pavilion 15	75 FPS	50 FPS

Berdasarkan Tabel 8, hasil pengujian pada perangkat pembanding menunjukkan bahwa metode *Texture Atlas* tetap menghasilkan nilai FPS yang lebih tinggi dibandingkan metode *Texture* Terpisah. Meskipun pengujian dilakukan pada perangkat dengan spesifikasi yang berbeda dan dalam kondisi *build* akhir, pola performa yang dihasilkan masih menunjukkan kecenderungan yang sama dengan perangkat utama. Hal ini memperkuat bahwa penggunaan *Texture Atlas* dapat memberikan peningkatan performa *rendering* secara konsisten pada berbagai perangkat.

3.10. Comparative Analysis

Perbandingan antara metode *Texture Atlas* dan *Texture* Terpisah menunjukkan adanya perbedaan performa yang konsisten pada seluruh posisi kamera. Metode *Texture Atlas* menghasilkan nilai FPS yang lebih tinggi serta jumlah *Draw Call* yang lebih rendah dibandingkan metode *Texture* Terpisah.

Secara teknis, perbedaan performa ini terjadi karena Unity melakukan proses *batching* yaitu menggabungkan beberapa *mesh* menjadi satu perintah render, hanya jika objek-objek tersebut berbagi material dan *Texture* yang sama [14]. Pada skenario *Texture* Terpisah, setiap objek memiliki material unik sehingga Unity tidak dapat menggabungkannya, setiap objek memerlukan *Draw Call* tersendiri beserta proses *state change* pada GPU (pergantian *shader binding* dan *Texture sampler*), yang membebani CPU dalam menyiapkan dan mengirim perintah render. Sebaliknya, pada skenario *Texture Atlas*, banyak objek berbagi satu material dan satu file *Texture*, sehingga Unity dapat menggabungkan objek-objek tersebut ke dalam *Draw Call* yang jauh lebih sedikit melalui mekanisme *static*/*dynamic batching*. Berkurangnya jumlah *Draw Call* ini secara langsung mengurangi *overhead* pada sisi CPU, yang pada *scene* dengan banyak objek kecil (khas *environment low-poly*) umumnya menjadi *bottleneck* utama dibandingkan beban pemrosesan GPU. Inilah yang menjelaskan mengapa peningkatan FPS pada metode *Texture Atlas* terlihat semakin signifikan pada posisi kamera dengan jumlah objek lebih banyak (Kamera 4), sebagaimana ditunjukkan pada Tabel 7 dan Gambar 6.

3.11. Statistical Analysis

Sebelum dilakukan uji-t berpasangan, terlebih dahulu dilakukan uji normalitas terhadap data selisih FPS antara metode *Texture Atlas* dan *Texture* Terpisah menggunakan uji Shapiro-Wilk pada JASP. Hasil uji menunjukkan nilai statistik $W = 0{,}919$ dengan $p = 0{,}093$ ($p > 0{,}05$), yang berarti data selisih FPS berdistribusi normal, sehingga asumsi normalitas untuk penggunaan uji-t berpasangan (uji parametrik) pada penelitian ini terpenuhi.

Untuk memverifikasi apakah perbedaan performa antara *Texture Atlas* dan *Texture* Terpisah signifikan secara statistik, dilakukan uji-t berpasangan (*paired sample t-test*) terhadap data FPS dari kedua metode pada seluruh 20 sampel pengujian (4 posisi kamera × 5 pengulangan). Uji berpasangan dipilih karena setiap sampel *Texture Atlas* dan *Texture* Terpisah diambil pada kondisi *scene*, posisi kamera, dan pengulangan yang identik, sehingga variabel yang berbeda hanya metode *Texture*-nya. Rata-rata FPS *Texture Atlas* ($M = 271{,}3$; $SD = 12{,}4$) lebih tinggi dibandingkan *Texture* Terpisah ($M = 258{,}5$; $SD = 9{,}0$), dengan selisih rata-rata 12,8 FPS. Hasil uji-t berpasangan menunjukkan $t(19) = 10{,}35$, $p < 0{,}001$, yang mengindikasikan bahwa perbedaan rata-rata FPS antara kedua metode signifikan secara statistik pada taraf kepercayaan 95% (bahkan 99,9%), sehingga bukan sekadar variasi acak pengukuran. Uji serupa dapat diterapkan pada data *Draw Call* untuk mengonfirmasi konsistensi temuan, mengingat nilai *Draw Call* pada setiap pengulangan bersifat konstan (tidak bervariasi), pembahasan *Draw Call* dapat difokuskan pada besaran persentase penurunan dibandingkan uji signifikansi.

3.12. Research Limitations

Penelitian ini memiliki beberapa keterbatasan. Pertama, pengujian hanya dilakukan pada satu *environment* bertema *medieval village* sehingga hasil belum tentu representatif untuk gaya visual, tema, atau kepadatan objek yang berbeda. Kedua, pengujian perangkat pembanding hanya melibatkan tiga unit laptop, belum mencakup perangkat mobile yang memiliki karakteristik GPU, memori, dan *thermal throttling* yang berbeda dari perangkat desktop/laptop. Ketiga, penelitian ini hanya mengukur FPS dan *Draw Call* tanpa memperhitungkan metrik lain seperti penggunaan memori GPU, *frame time*, atau ukuran file *build*, yang juga relevan dalam evaluasi performa *rendering* secara menyeluruh. Selain itu, efektivitas *Texture Atlas* berpotensi menurun pada beberapa kondisi tertentu. Pada *scene* dengan jumlah objek yang sedikit, pengurangan *Draw Call* dari penggabungan *Texture* tidak akan berdampak signifikan karena beban CPU sudah rendah sejak awal, sehingga skenario menjadi lebih GPU-*bound* daripada CPU-*bound* dan potensi peningkatan FPS menjadi minim. Selain itu, penggunaan *atlas* beresolusi tinggi untuk menampung banyak *Texture* kecil dapat meningkatkan konsumsi memori GPU dan berisiko menimbulkan *Texture Bleeding* (kebocoran warna antar-*Texture* akibat mipmap) apabila *padding* antar-*Texture* di dalam *atlas* tidak memadai. Kompleksitas proses *UV Mapping* juga meningkat pada metode *Texture Atlas* dibandingkan *Texture* Terpisah, karena setiap objek harus disusun ulang koordinat UV-nya agar sesuai dengan posisi barunya di dalam *atlas*, hal ini menambah beban kerja pada tahap produksi aset meskipun menguntungkan pada tahap *rendering*. Oleh karena itu, penerapan *Texture Atlas* sebaiknya mempertimbangkan kepadatan objek dalam *scene* serta kebutuhan resolusi *Texture*, bukan diterapkan secara seragam pada semua kondisi.

4. CONCLUSION

Berdasarkan hasil penelitian yang telah dilakukan, penggunaan *Texture Atlas* terbukti lebih efektif dalam meningkatkan performa *rendering* dibandingkan metode *Texture* Terpisah pada *environment* 3D *low-poly* di Unity. Hasil pengujian pada seluruh posisi kamera menunjukkan bahwa metode *Texture Atlas* menghasilkan rata-rata FPS yang lebih tinggi (271,3) dibandingkan *Texture* Terpisah (258,5) dengan jumlah *Draw Call* yang lebih rendah secara konsisten. Perbedaan tersebut terbukti signifikan secara statistik melalui

uji-t berpasangan ($t(19) = 10{,}35$; $p < 0{,}001$), sehingga bukan sekadar variasi acak pengukuran. Secara teknis, peningkatan performa ini disebabkan oleh kemampuan Unity melakukan *batching* objek yang berbagi material dan *Texture* yang sama, sehingga jumlah *Draw Call* dan *overhead* CPU dalam proses *rendering* dapat ditekan secara signifikan, terutama pada *scene* dengan jumlah objek yang padat. Temuan ini memperkuat sekaligus melengkapi penelitian terdahulu yang membahas *Texture Atlas* pada domain di luar *game engine* real-time, dengan menghadirkan bukti empiris berbasis pengukuran performa langsung (FPS dan *Draw Call*) pada *environment* 3D *low-poly* bertema *game*.

Penelitian ini menunjukkan bahwa pengelolaan *Texture* menggunakan *Texture Atlas* dapat menjadi strategi optimasi yang efektif dalam pengembangan *game* berbasis Unity, khususnya pada *environment* dengan jumlah objek yang banyak, sehingga dapat menjadi rujukan praktis bagi pengembang *game* dalam menentukan strategi manajemen *Texture* yang lebih efisien. Dengan mempertimbangkan keterbatasan yang telah diuraikan pada subbab 3.12, penelitian selanjutnya disarankan untuk memperluas pengujian pada tema *environment* dan jumlah objek yang lebih bervariasi, menambahkan parameter performa lain seperti penggunaan memori GPU dan *frame time*, serta menguji kondisi *scene* dengan kepadatan objek rendah untuk mengidentifikasi ambang batas (*threshold*) di mana *Texture Atlas* mulai memberikan manfaat performa yang signifikan.

ACKNOWLEDGEMENTS

Penulis mengucapkan terima kasih kepada Politeknik Negeri Batam atas dukungan dalam pelaksanaan penelitian ini. Ucapan terima kasih juga disampaikan kepada dosen pembimbing yang telah memberikan arahan, masukan, dan bimbingan selama proses penelitian hingga penyusunan artikel ini. Selain itu, penulis juga mengucapkan terima kasih kepada seluruh pihak yang telah membantu dan mendukung jalannya penelitian ini.

REFERENCES

[1] Statista, “Mobile gaming industry report,” pp. 1–33, 2024.
[2] J. Linåker, E. Bjarnason, and F. Fagerholm, “Pre-release Experimentation in Indie Game Development: An Interview Survey,” *Lect. Notes Bus. Inf. Process.*, vol. 539 LNBIP, pp. 293–308, 2025, doi: 10.1007/978-3-031-85849-9_24.
[3] Unity Technologies, “Optimizing Graphics Performance.” Accessed: Aug. 12, 2026. [Online]. Available: https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html
[4] Android Developers, “Textures.” [Online]. Available: https://developer.android.com/games/optimize/textures
[5] X. Zhang, W. Liu, B. Liu, X. Zhao, and Z. Hu, “Size-adaptive texture atlas generation and remapping for 3D urban building models,” *ISPRS Int. J. Geo-Information*, vol. 10, no. 12, 2021, doi: 10.3390/ijgi10120798.
[6] R. Fernando, “Graphics Pipeline Performance,” in *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, 2004, ch. 28. [Online]. Available: https://developer.nvidia.com/gpugems/gpugems/part-v-performance-and-practicalities/chapter-28-graphics-pipeline-performance
[7] Microsoft, “Understanding Performance for Mixed Reality.” Accessed: Jul. 28, 2026. [Online]. Available: https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/understanding-performance-for-mixed-reality
[8] W. Shen, L. Huo, T. Shen, M. Zhang, and Y. Li, “Optimization of Texture Rendering of 3D Building Model Based on Vertex Importance,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, vol. 48-2–2024, no. June, pp. 371–378, 2024, doi: 10.5194/isprs-archives-XLVIII-2-2024-371-2024.
[9] G. Koulaxidis and S. Xinogalos, “Improving Mobile Game Performance with Basic Optimization Techniques in Unity,” *Modelling*, vol. 3, no. 2, pp. 201–223, 2022, doi: 10.3390/modelling3020014.
[10] Z. Yang, Z. Pan, M. Li, K. Wu, and X. Gao, “Learning based 2D Irregular Shape Packing,” *ACM Trans. Graph.*, vol. 42, no. 6, pp. 1–16, 2023, doi: 10.1145/3618348.
[11] Unity Technologies, “Lit shader material Inspector window reference for URP.” Accessed: Jul. 25, 2026. [Online]. Available: https://docs.unity3d.com/6000.4/Documentation/Manual/urp/lit-shader.html
[12] Juego Studio, “Low Poly 3D Modeling for Game Assets.” Accessed: Jul. 28, 2026. [Online]. Available: https://www.juegostudio.com/blog/low-poly-3d-modeling-for-game-assets
[13] Android Developers, “Materials and Shaders.” Accessed: Jul. 28, 2026. [Online]. Available: https://developer.android.com/games/optimize/materials
[14] Unity Technologies, “Introduction to optimizing draw calls.” Accessed: Jul. 25, 2026. [Online]. Available: https://docs.unity3d.com/6000.2/Documentation/Manual/optimizing-draw-calls.html