Implementasi *Bloom* Sebagai Efek Gambar dalam Permainan Video

Amar Fadil - 13520103
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: marfgold1@gmail.com

***Abstract*—Efek gambar sering digunakan dalam permainan video sebagai perbaikan citra untuk menghasilkan gambar yang tampak seperti nyata. Salah satu efek gambar yang umum dipakai adalah *bloom* dengan mensimulasikan mekanisme fisik dari mata manusia sehingga menimbulkan *scattering* atau difraksi terhadap objek yang terang. Makalah ini membahas tentang cara implementasi *bloom* yang digunakan dalam permainan video.**

Keywords—image effect; video games; bloom

I. Pendahuluan

Permainan video merupakan salah satu bentuk dari *Computer Generated Imagery* (CGI) yang dibuat secara *real-time*. Resolusi gambar dan *frame rate* dari permainan sangat mempengaruhi performa dari pemain sehingga berdampak pada *playability* dan *enjoyability* permainan. Pada permainan video *shooter* misalnya, setiap detik permainan membutuhkan sekitar 60 *frames*, biasa disebut sebagai 60 *frames-per-second* (FPS), untuk meningkatkan performa sebesar 7 kali lipat dari 3 FPS yang hampir tidak dapat dimainkan karena pemain tidak dapat menargeti musuh pada *frame rate* yang kecil [1]. Selan itu, membuat gambar yang terlihat nyata dari CGI membutuhkan bantuan pemrosesan gambar untuk mensimulasikan efek pada dunia nyata menjadi efek digital. Namun, efek gambar tersebut harus dapat diimplementasikan secara *real time* sehingga tidak mengurangi kesenangan dari pemain.

Dalam dunia nyata, kita menemukan objek-objek terang dengan cahaya yang terlihat menyebar di sekitarnya. Efek tersebut biasa disebut sebagai *bloom* atau *glow*. *Bloom* merupakan efek visual yang sering digunakan dalam permainan video dan film untuk menciptakan kesan cahaya yang menyebar tersebut. Efek ini dapat membuat objek tampak lebih terang dan menonjol, sehingga menambah suasana pada sebuah *scene*. Efek *bloom* dalam dunia nyata ditimbulkan dari efek yang dihasilkan oleh mata manusia berupa *scattering* pada kornea, lensa, dan retina, serta *diffraction* pada struktur sel koheren pada area radial di luar lensa [2]. *Glow* dan *halo* dari cahaya ini memberikan isyarat visual mengenai kecerahan gambar dan atmosfer dari *scene*. Dengan berkembangnya perangkat keras pemrosesan grafis atau *Graphics Processing Unit* (GPU), kalkulasi efek ini dapat dilakukan hanya dengan beberapa operasi *rendering* sederhana, sehingga dapat dijalankan secara *real time* dan memberikan tampilan yang lebih realistis terhadap objek-objek terang [3].

Terdapat beberapa metode yang dapat digunakan untuk memperbaiki citra, baik dalam domain frekuensi maupun spasial. Salah satu pendekatan yang umum digunakan untuk meningkatkan citra adalah dengan menggunakan penapis lolos rendah seperti *gaussian blur*. Filter ini dapat menciptakan efek visual yang mengesankan dan diaplikasikan untuk memberikan tampilan yang lebih halus pada gambar. Salah satu aplikasi efek ini adalah simulasi cahaya terang (*bloom*) pada objek-objek dalam suatu *scene*.

Pada makalah ini, akan diulas mengenai implementasi efek *bloom* yang telah ada dengan pengolahan citra pada permainan video baik dalam domain spasial maupun frekuensi. Pada domain spasial, akan digunakan penapis lolos rendah seperti Gaussian, sedangkan pada domain frekuensi akan menggunakan gambar kernel. Selain itu, akan dibahas efek gambar *bloom real-time* yang dapat digunakan dalam permainan video, serta implementasinya dalam *game engine* popular.

II. Landasan Teori

A. *Bloom*

Bloom dalam konteks grafika komputer adalah efek visual di mana cahaya memiliki tampak yang "membesar" atau "menyala" di sekitar objek sangat terang. Hal ini menciptakan kilau yang melebihi batas objek yang terang dan memberikan kesan bahwa cahaya tersebar atau menyinari lingkungan sekitarnya. Efek bloom sering digunakan untuk memberikan nuansa realisme atau dramatisme dalam permainan video, film, dan aplikasi grafis. Dalam pengolahan citra, *bloom* sering dicapai dengan menggunakan *penapis* dan teknik tertentu.

Dalam dunia nyata, tepatnya pada pandangan visual manusia, *bloom* adalah fenomena perseptual yang disebabkan oleh penyebaran cahaya dari tiga komponen penting sistem visual: kornea, lensa, dan retina [2]. Terdapat dua elemen utama dalam efek silau: *flare* yang secara dominan dipengaruhi oleh lensa, dan *bloom* yang muncul sebagai cahaya berkilau mengelilingi objek-objek terang. Cahaya ini menyebabkan pengurangan kontras, mempengaruhi keterlihatan objek-objek di sekitarnya. Faktor-faktor yang berkontribusi terhadap *bloom* adalah penyebaran (*scattering*) dari kornea, lensa kristalin, dan retina, dimana semuanya memiliki peran seimbang dalam menghasilkan efek visual ini. *Bloom* paling sering terlihat

dalam situasi seperti penglihatan pada malam hari dari *grill* di antara dua lampu depan mobil, dimana perubahan kontras mempengaruhi persepsi kita terhadap lingkungan sekitar.

B. Konvolusi

Konvolusi adalah operasi matematika yang menggabungkan dua fungsi untuk menghasilkan fungsi baru [4]. Dalam konteks pengolahan citra, konvolusi digunakan untuk memodifikasi citra dengan menggunakan kernel. Konvolusi dua buah fungsi $f(x)$ dan $g(x)$ didefinisikan sebagai berikut:

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da$$

Tanda $*$ merupakan operasi konvolusi dan a adalah peubah bantu (*dummy variable*). Fungsi $f(x)$ merupakan citra asli, sedangkan fungsi $g(x)$ juga merupakan *kernel* atau *mask* konvolusi. Kernel $g(x)$ dapat dianalogikan sebagai sebuah jendela yang digeser-geser pada citra asli $f(x)$. Nilai setiap pixel pada citra asli yang berada di bawah jendela akan dikalikan dengan nilai masing-masing elemen pada jendela. Hasil perkalian tersebut kemudian dijumlahkan untuk menghasilkan nilai pixel baru pada citra hasil konvolusi $h(x)$. Dalam fungsi diskrit, konvolusi dapat dilakukan dengan formula berikut:

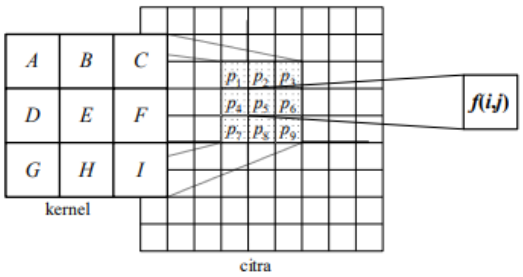
$$h(x) = f(x) * g(x) = \sum_{a=-\infty}^{\infty} f(a)g(x-a)$$

Konvolusi memiliki beberapa sifat, diantaranya adalah komutatif, asosiatif, dan distributif. Pada fungsi dwimatra (dua dimensi) seperti citra, operasi konvolusi untuk fungsi diksrit adalah sebagai berikut:

$$\begin{aligned} & h(x,y) \\ & = f(x,y) * g(x,y) \\ & = \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} f(a,b)g(x-a, y-b) \end{aligned}$$

Pada formula tersebut, $g(x,y)$ juga disebut seabgai *convolution filter* atau *mask*, *kernel*, atau *template*. Biasanya, pada ranah diskrit, *kernel* dinyatakan dalam bentuk matriks, umumnya berukuran kecil, namun ukuran 3x3 sering digunakan. Ukuran kernel dengan pola angka yang berbeda akan membentuk hasil yang berbeda pula pada konvolusi.

Sebagai contoh, terdapat kernel dan citra pada gambar di bawah berikut [4].



Untuk menghitung hasil pada $f(i,j)$, dapat dilakukan kombinasi linier dari vektor *pixel* p dengan vektor kernel sebagai berikut:

$$\frac{Ap_1 + Bp_2 + Cp_3 + Dp_4 + Ep_5 + Fp_6 + Gp_7 + Hp_8 + Ip_9}{M}$$

Dengan M adalah jumlah koefisien pada kernel, atau 1 jika jumlah koefisien bernilai 0. Secara umum, perhitungan konvolusi citra d_{ij} dengan *square kernel* f_{ij} berukuran $q \times q$ dilakukan dengan rumus berikut:

$$V = \left| \frac{\sum_{i=1}^{q} \sum_{j=1}^{q} f_{ij} d_{ij}}{M} \right|$$

Ketika nilai hasil konvolusi berada pada luar rentang nilai keabuan minimum dan maksimum, maka lakukan *clipping*. Misalkan nilai keabuan minimum adalah 0 dan maksimum adalah 255, maka nilai yang lebih kecil dari minimum akan menjadi 0 dan nilai yang lebih besar dari maksimum menjadi 255.

C. Penapisan Citra

Penapisan citra adalah proses memodifikasi pixel-pixel di dalam citra berdasarkan nilai-nilai pixel di sekitarnya [4]. Tujuan penapisan citra adalah untuk meningkatkan kualitas citra, memperbaiki kekurangan citra, atau mengubah tampilan citra sesuai dengan kebutuhan. Penapisan citra termasuk ke dalam tipe operasi aras lokal. Proses penapisan citra dapat dilakukan dengan menggunakan operasi konvolusi.

Terdapat berbagai macam penerapan penapisan citra. Berikut adalah beberapa contoh penapisan citra:

- Penapisan median digunakan untuk mengurangi *noise* dengan cara mengganti setiap piksel dengan nilai median dari piksel-piksel di sekitarnya.
- Penapisan Gaussian digunakan untuk mengurangi noise dengan cara mengaplikasikan kernel *gaussian* pada citra.
- Penapisan tepi digunakan untuk meningkatkan atau mendeteksi piksel tepi yang ada di dalam citra.

D. Penapis Lolos Rendah

Penapis lolos rendah atau *low-pass filter* (LPF) adalah penapis yang melewatkan frekuensi rendah dari sinyal masukan dan meredam frekuensi tinggi. Dalam konteks pengolahan citra, penapis lolos rendah digunakan untuk mengurangi *noise* dan menghaluskan citra dengan cara menghilangkan komponen frekuensi tinggi yang biasanya berhubungan dengan *noise* dan detail-detail kecil [5]. Kernel yang digunakan pada penapis lolos rendah memiliki aturan bahwa semua bobot di dalam penapis harus positif dan jumlah semua bobot penapis adalah 1. Contoh kernel LPF umum adalah kernel rata-rata, kernel *median*, dan kernel Gaussian. Penggunaan penapis lolos rendah akan menghasilkan citra yang lebih halus dan buram, tetapi detail-detail kecil dan tepi-tepi citra mungkin akan menjadi kabur.

Penapis lolos rendah banyak digunakan dalam berbagai aplikasi pengolahan citra, seperti:

- Pengurangan noise pada citra digital yang terganggu oleh noise.
- Pra-pemrosesan citra sebelum dilakukan deteksi tepi atau segmentasi.
- Pengaburan latar belakang untuk membuat objek fokus lebih menonjol.

E. Penapis Gaussian

Penapis Gaussian adalah salah satu jenis penapis lolos rendah yang menggunakan kernel berbentuk lonceng Gaussian. Kernel ini memiliki nilai maksimum di titik pusat dan menurun secara eksponensial ke arah tepi. Bobot di dalam penapis dihitung dengan penerokan (*sampling*) pada fungsi Gaussian. Berikut contoh *mask* dari Gaussian berukuran 3 x 3 dan 7 x 7

$$\frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} \qquad \frac{1}{140}\times\begin{bmatrix}1&1&2&2&2&1&1\\1&2&2&4&2&2&1\\2&2&4&8&4&2&2\\2&4&8&16&8&4&2\\2&2&4&8&4&2&2\\1&2&2&4&2&2&1\\1&1&2&2&2&1&1\end{bmatrix}$$

Ukuran dari penapis bergantung pada standar deviasi σ yang menentukan derajat pelembutan, sehingga σ yang semakin besar akan membuat citra menjadi semakin lembut atau *blur*. Penapis Gaussian memiliki beberapa karakteristik yang membuatnya populer dalam pengolahan citra:

- Respon frekuensi halus: Penapis Gaussian memiliki transisi yang halus antara frekuensi lolos terhadap frekuensi teredam, sehingga tidak menimbulkan efek *ringing* (*halo* di sekitar tepi) pada citra.
- Sifat *separable*: Kernel Gaussian dapat dipisahkan menjadi dua kernel satu dimensi, sehingga perhitungan konvolusi dapat dilakukan lebih efisien.

Konvolusi spasial antara citra dan penapis Gaussian yang tidak terpisah berukuran $N \times N$ membutuhkan total N^2 perkalian dan $N^2 - 1$ penjumlahan per piksel. Akan tetapi, Gaussian memiliki representasi terpisah [6] yang secara matematis didefinsikan sebagai berikut:

$$g(x,y) = g(x)g(y)$$

Persamaan tersebut menunjukkan bahwa penapis terpisah Gaussian dapat diekspresikan sebagai perkalian dua penapis satu dimensi, satu berukuran $N \times 1$ dan yang lainnya berukuran $1 \times N$. Ini berarti bahwa konvolusi menggunakan penapis terpisah dapat dilakukan dalam dua langkah: citra terlebih dahulu dikonvolusi dengan penapis berukuran $N \times 1$, dan kemudian hasilnya dikonvolusi dengan penapis berukuran $1 \times N$. Dalam kasus ini, diperlukan total $2N$ perkalian dan $2N - 2$ penjumlahan, yang jauh lebih sedikit dibandingkan dengan kasus tidak terpisah, terutama untuk penapis berukuran besar. Sifat *separable* Gaussian memiliki keunggulan efisiensi komputasi karena dapat dilakukan dalam dua langkah yang lebih efisien daripada konvolusi menggunakan penapis tidak terpisah. Komputasi juga relatif lebih mudah dan dapat diimplementasikan dalam bentuk *shader* pada GPU secara sederhana.

F. Transformasi Citra

Transformasi citra merupakan suatu teknik untuk mengubah representasi citra dari satu bentuk ke bentuk lain. Citra merupakan sinyal dari gelombang cahaya, maka citra selain dapat diolah dalam ranah spasial, juga dapat diolah dalam ranah frekuensi [7]. Manipulasi nilai-nilai *pixel* dilakukan dalam ranah spasial, sementara dalam ranah frekuensi, citra yang merupakan sinyal dari gelombang cahaya dapat dioperasikan dengan memanipulasi nilai-nilai frekuensi yang merepresentasikan sinyal.

Proses transformasi citra memiliki dua tahap sebagai berikut:

1) Transformasi Citra: citra awal dalam ranah spasial, $f(x,y)$ mengalami transformasi menjadi citra dalam ranah frekuensi, $F(u,v)$. Kemudian, operasi pada citra dalam ranah frekuensi dilakukan dengan manipulasi nilai-nilai $F(u,v)$.

2) Inverse Image Transform: citra yang telah diolah dalam ranah frekuensi $F'(u,v)$ dikembalikan ke ranah spasial $f'(x,y)$.

Dalam ranah frekuensi, operasi konvolusi dapat dilakukan dengan perkalian antara citra $F(u,v)$ dengan kernel $G(u,v)$, yaitu perkalian elemen-elemen yang bersesuaian pada kedua buah matriks [7], secara matematis didefinisikan sebagai berikut:

$$h(x,y) = f(x,y) * g(x,y) \leftrightarrow H(u,v) = F(u,v)G(u,v)$$

Salah satu transformasi yang paling banyak digunakan sebagai transformasi citra adalah transformasi Fourier (dan kebalikannya yaitu Inverse Fourier). Transformasi Fourier adalah suatu konsep matematis yang digunakan untuk menganalisis dan merepresentasikan suatu sinyal dalam ranah frekuensi. Transformasi ini ditemukan oleh Jean-Baptiste Joseph Fourier dan digunakan secara luas dalam berbagai bidang, termasuk pengolahan citra, sinyal, dan sistem. Transformasi Fourier mengubah suatu sinyal atau fungsi waktu dalam ranah spasial menjadi representasi frekuensi, memperlihatkan komponen frekuensi yang menyusun sinyal tersebut. Dalam pengolahan citra, Transformasi Fourier Diskret atau Discrete Fourier Transform (DFT) lebih umum digunakan karena mudah diimplementasikan secara komputasional dan juga citra yang berbentuk diskrit hasil penerokan dari sinyal cahaya kontinu.

Transformasi Fourier menghasilkan dua bagian, yaitu bagian real dan bagian imajiner. Kedua bagian ini digunakan untuk merepresentasikan amplitudo dan fase dari komponen frekuensi sinyal. Transformasi Fourier dapat membantu mengidentifikasi pola atau karakteristik frekuensi tertentu dalam suatu sinyal, sehingga berguna dalam analisis dan pemrosesan sinyal, seperti penghilangan derau, pemampatan data, dan pengolahan citra digital. Secara diskrit, DFT dan IDFT untuk dua peubah seperti citra dapat dirumuskan dalam persamaan berikut:

$$F_{u,v} = \frac{1}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} f_{x,y} e^{-i2\pi\left(\frac{ux}{N}+\frac{vy}{M}\right)}$$

$$f_{x,y} = \sum_{u=0}^{N-1}\sum_{v=0}^{M-1} F_{u,v} e^{i2\pi\left(\frac{ux}{N}+\frac{vy}{M}\right)}$$

Dapat dihitung pula spektrum fourier dengan rumus berikut:

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)}$$

Dengan sudut fase sebagai berikut

$$\Theta(u,v) = \tan^{-1}\left(\frac{I(u,v)}{R(u,v)}\right)$$

Terdapat beberapa alasan untuk melakukan penapisan citra dalam ranah frekuensi. Pertama, metode penapisan citra untuk menghilangkan derau lebih efektif dilakukan dalam ranah frekuensi daripada ranah spasial. Kedua, konvolusi dalam ranah spasial memiliki kendala dalam waktu komputasi yang besar. Jika citra berukuran $N \times N$ dan kernel berukuran $M \times M$, jumlah perkalian dalam operasi konvolusi menjadi sekitar N^2m^2 . Sebagai contoh, pada citra berukuran 512 x 512 dan kernel berukuran 16 x 16, diperlukan sekitar 32 juta perkalian [7]. Hal ini menjadi tidak cocok untuk diproses secara *real-time* tanpa perangkat keras *dedicated*, karena membutuhkan sumber daya komputasi yang signifikan.

III. Implementasi dan Hasil

Pada makalah ini, akan diulas dua cara dalam menerapkan efek gambar *bloom* yang telah ada. Pertama adalah yang paling umum digunakan dalam permainan video, yaitu *blur* atau pelembutan dengan penapis lolos rendah (pada umumnya Gaussian). Kedua adalah pendekatan lain yang digunakan oleh Unreal Engine, yaitu kakas untuk membuat permainan video atau sering disebut sebagai *game engine*.

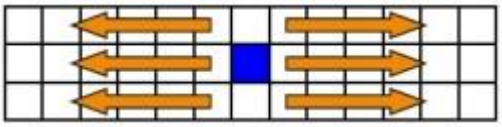
A. Pelembutan dengan Penapis Lolos Rendah Gaussian

Untuk membuat efek citra berupa *bloom*, cara umum yang sering digunakan adalah menggunakan penapis lolos rendah utnuk melembutkan citra atau *blur*. Berikut adalah tahapan dalam melakukan implementasi *bloom*:

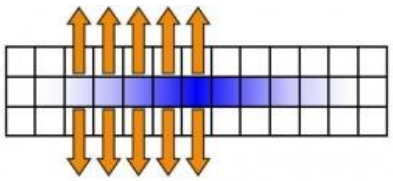
1. Tentukan asal citra yang akan dilakukan *bloom*.
2. Lakukan *downsampling* terhadap tektur untuk performa sehingga pemrosesan akan dilakukan lebih cepat.
3. Terapkan *blur* menggunakan penapis lolos rendah pada citra.
4. Hasil *blur* kemudian ditambahkan ke citra asli.

Lebih lanjut, implementasi *blur* dalam menciptakan efek *bloom* melibatkan penggunaan penapisan untuk pemrosesan citra dua dimensi menggunakan penapis lolos rendah seperti Gaussian. Proses *blur* atau pelembutan ini sangat penting untuk mendapatkan efek *bloom* yang lembut dan alami. Efisiensi operasi *blur* secara signifikan mempengaruhi kecepatan dalam menghasilkan efek *bloom*. Waktu yang diperlukan untuk melakukan *blur* bergantung pada ukuran *kernel* penapis lolos rendah. Untuk mengatasi pemrosesan citra dengan ukuran *blur* yang lebih besar, digunakan pendekatan dua langkah yang disebut dengan *separable convolution*. Pelembutan menggunakan Gaussian dapat memanfaatkan sifat *separable* dari fungsi Gaussian. Pendekatan ini akan mengurangi perhitungan dari pangkat dua ukuran kernel Gaussian k^2 menjadi hanya dua kali ukuran kernel Gaussian $2k$, sehingga beban untuk membuat *bloom* dengan ukuran yang lebih besar akan menjadi ringan. Berikut adalah tahapan dalam melakukan *separable convolution* [8]

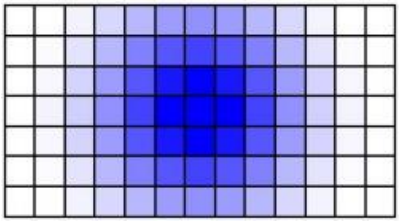
1. Lakukan pelembutan citra asli secara horizontal.



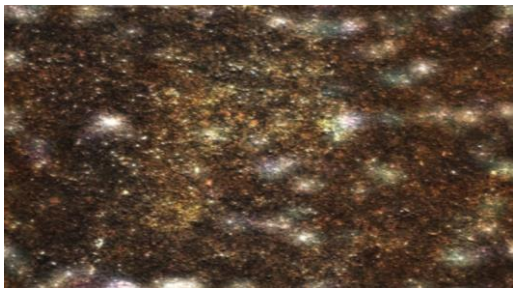
2. Hasil dari pelembutan secara horizontal, kemudian dilembutkan secara vertikal



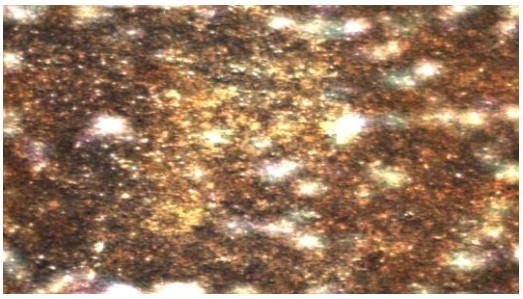
3. Hasil *blur* akan terlihat



Berikut adalah hasil *bloom* menggunakan penapis lolos rendah Gaussian:



Citra Sebelum Konvolusi



Citra Setelah Konvolusi

Dapat dilihat bahwa citra memiliki kecerahan yang lebih tinggi, terutama pada objek yang terang. Kecerahan lebih tersebar secara merata.

B. Konvolusi pada Domain Frekuensi

Alternatif lain dalam melakukan implementasi efek gambar *bloom* adalah melakukan konvolusi dengan kernel pada domain frekuensi. Pendekatan ini dilakukan oleh Unreal Engine [9] dengan motivasi bahwa penapis Gaussian menyebabkan *bloom* yang tersebar secara simetris atau sirkular. Selain itu, *bloom* memiliki beban komputasi konvolusi dalam domain spasial yang tinggi, yakni $O(N^2)$. Oleh karena itu, Unreal Engine memanfaatkan konvolusi Fast Fourier Transform (FFT) untuk meningkatkan efisiensi komputasi. Berikut adalah langkah-langkah implementasi yang digunakan:

1. FFT pada Citra Awal (Image_Frequencies)

   Citra yang akan diterapkan konvolusi diubah ke ranah frekuensi melalui FFT. Transformasi ini dilakukan untuk menciptakan representasi citra dalam domain frekuensi, memfasilitasi operasi konvolusi dengan *kernel*.

2. FFT pada Kernel (Filter_Frequencies)

   Kernel yang akan digunakan untuk konvolusi juga diubah ke ranah frekuensi melalui FFT. Proses ini menghasilkan representasi kernel dalam domain frekuensi yang akan digunakan dalam langkah selanjutnya. Langkah ini dilakukan hanya sekali, lalu hasilnya di-*cache* sehingga tidak perlu melakukan transformasi kernel berulang kali. Jika sudah ada *cache*, maka cukup gunakan hasil yang sudah di-*cache*.

3. Konvolusi dalam Domain Frekuensi

   Pada tahap ini, akan dilakukan proses konvolusi sebenarnya dalam domain frekuensi dengan mengalikan representasi citra pada domain frekuensi (Image_Frequencies) dengan representasi kernel pada domain frekuensi (Filter_Frequencies). Langkah ini memanfaatkan sifat perkalian dalam domain frekuensi untuk menghasilkan citra yang telah mengalami konvolusi.

4. Inverse FFT pada Hasil Konvolusi (Convolved_Image)

   Setelah konvolusi dalam domain frekuensi selesai, hasilnya dikembalikan ke domain spasial melalui *inverse* FFT. Langkah ini menghasilkan citra yang telah mengalami konvolusi sesuai dengan kernel yang diaplikasikan sehingga menghasilkan citra dengan efek *bloom*.

Kernel yang digunakan untuk konvolusi merepresentasikan respons dari perangkat optik seperti kamera terhadap sumber titik tunggal di tengah *view field*. Setiap piksel pada sumber berkontribusi terhadap Sebagian dari kecerahannya kepada tetangga-tetangganya sesuai dengan kernel. Semakin terang piksel sumber, semakin terlihat kilau yang dihasilkannya. Kernel harus selalu ada di GPU dan tersedia dalam resolusi penuh. Jika tidak, kualitas citra hasil *bloom* akan sangat buruk. Contoh kernel yang digunakan adalah sebagai berikut:

Kiri adalah gambar kernel asli dan kanan adalah gambar kernel yang diperbesar untuk memperjelas bentuk *bloom*.

Implementasi konvolusi FFT ini memanfaatkan kecepatan transformasi citra yang relatif lebih cepat dalam domain frekuensi, memiliki kompleksitas transformasi dan *inverse* transformasi sebesar $O(NlogN)$ dengan konvolusi sebesar $O(N)$, dibandingkan pada domain spasial dengan kompleksitas konvolusi $O(N^2)$. Dalam *compute shader*, implementasi FFT dilakukan dengan tiga tahap: pertama *forward horizontal transform*, kedua *forward vertical transform* dilanjutkan dengan konvolusi lalu *inverse vertical transform* pada tahap yang sama, dan tahap ketiga atau terakhir *inverse horizontal transform*. Tahap-tahap ini memastikan bahwa proses konvolusi FFT dilakukan dengan efisien, mengoptimalkan waktu komputasi untuk menghasilkan *bloom* yang lebih akurat

Hasil dari *bloom* menggunakan teknik ini, sesuai yang ada pada dokumentasi Unreal Engine [9] adalah sebagai berikut:

Gambar Sebelum Diterapkan Efek *Bloom*

Gambar Setelah Menerapkan Efek *Bloom*

Terlihat bahwa efek *bloom* yang dihasilkan terasa lebih sinematik dan realistis. Ketidaksimetrian dari pendekatan ini membuat *bloom* bukan menjadi efek yang membosankan.

IV. Kesimpulan

Efek gambar *bloom* pada permainan video dapat di-implementasikan dengan menggunakan penapis lolos rendah seperti *Gaussian* sehingga menciptakan *blur* atau citra yang halus. Penapis ini mensimulasikan efek yang menciptakan cahaya terang atau *halo* di sekitar objek terang. Selain itu, terdapat pendekatan menggunakan gambar kernel yang melakukan konvolusi pada domain frekuensi. Konvolusi dengan gambar kernel pada domain frekuensi memberikan hasil yang lebih bagus daripada penapis Gaussian sederhana. Terakhir, untuk optimisasi sehingga dapat dijalankan secara *real-time*, dilakukan *downsampling* dan implementasi dalam bentuk program GPU.

Link Video Youtube

https://youtu.be/ATIvdUqftCE

Ucapan Terima Kasih

Dengan selesainya tugas makalah ini, penulis mengucapkan rasa syukur kepada Tuhan Yang Maha Esa atas karunia-Nya sehingga saya bisa menyelesaikan makalah ini dengan baik. Penulis juga berterima kasih kepada orang tua dan teman-teman yang telah membantu penulis sebagai *support system* sehingga membuat saya semangat untuk menyelesaikan makalah ini. Terakhir, saya berterima kasih kepada bapak Dr. Ir. Rinaldi Munir, MT. selaku dosen yang telah mengajar saya selama perkuliahan mata kuliah IF4073 Interpretasi dan Pengolahan Citra.

Referensi

[1] M. Claypool, K. Claypool, dan F. Damaa, "The effects of frame rate and resolution on users playing first person shooter games," S. Chandra dan C. Griwodz, Ed., Jan 2006, hlm. 607101. doi: 10.1117/12.648609.

[2] G. Spencer, P. Shirley, K. Zimmerman, dan D. P. Greenberg, "Physically-based glare effects for digital images," dalam *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95*, New York, New York, USA: ACM Press, 1995, hlm. 325–334. doi: 10.1145/218380.218466.

[3] G. James dan J. O'Rorke, "Real-Time Glow," dalam *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, 1 ed., R. Fernando, Ed., Pearson Higher Education, 2004.

[4] R. Munir, "Penapisan Citra dan Konvolusi." Bandung, 2023.

[5] R. Munir, "Image Enhancement (Bagian 3)." 2023.

[6] J. L. Mitchell, M. Y. Ansari, dan E. Hart, "Advanced Image Processing with DirectX® 9 Pixel Shaders."

[7] R. Munir, "Transformasi Citra." 2023.

[8] Game Rendering, "Gaussian Blur Filter Shader." Diakses: 19 Desember 2023. [Daring]. Tersedia pada: https://web.archive.org/web/20140709054723/https://www.gamerendering.com/2008/10/11/gaussian-blur-filter-shader/

[9] Unreal Engine, "Bloom." Diakses: 19 Desember 2023. [Daring]. Tersedia pada: https://docs.unrealengine.com/5.0/en-US/bloom-in-unreal-engine/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Desember 2023

Amar Fadil
13520103