Penerapan Singular Value Decomposition (SVD) dalam Kompresi Data untuk Meningkatkan Performa Game

Ivant Samuel Silaban - 13523129[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523129@itb.ac.id
[1]smart058440@gmail.com

***Abstract*—Dalam dunia permainan modern saat ini, kualitas grafis yang tinggi biasanya berbanding terbalik dengan permainan, dimana semakin tinggi kualitas grafis, maka performa yang dihasilkan menurun. Hal ini mendorong para *game developer* menyediakan opsi pengaturan grafis yang dapat disesuaikan, baik dari kualitas rendah hingga tinggi atau disesuaikan dengan performa atau kualitas. Salah satu metode yang dapat digunakan untuk meningkatkan performa grafis tanpa mengorbankan kualitas secara signifikan adalah dengan menerapkan *Singular Value Decomposition* (SVD) dalam kompresi data grafis. SVD ini dapat mereduksi dimensi data dengan efisiensi tinggi namun masih dapat mempertahankan informasi penting. Makalah ini akan membahas konsep SVD dan implementasinya dalam kompresi data grafis untuk game sehingga dapat memberikan pengalaman bermain yang optimal tanpa mengorbankan estetika visual.**

***Keywords*— Singular Value Decomposition (SVD), Grafis, Performa, Kompresi Data**

I. Introduction

Industri game terus berkembang dengan menghadirkan grafis yang semakin realistis, tetapi di sisi lain, kebutuhan akan perangkat keras yang mumpuni menjadi kendala bagi sebagian besar pemain. Banyak pemain yang menggunakan perangkat dengan spesifikasi rendah harus memilih pengaturan grafis rendah demi mendapatkan performa yang lebih baik. Hal ini efektif meningkatkan *frame rate* (FPS), namun kualitas visual seringkali mengurangi daya tarik dan pengalaman bermain.

Salah satu pendekatan untuk mengatasi masalah ini adalah menggunakan metode *Singular Value Decomposition* yang dikenal dalam berbagai aplikasi matematis, termasuk pengolahan citra dan kompresi data. SVD dapat menawarkan kemampuan untuk menyederhanakan data kompleks menjadi bentuk yang lebih menghilangkan karakteristik utamanya. Dalam konteks permainan, SVD dapat digunakan untuk mengurangi kompleksitas data grafis, sehingga menghasilkan aset visual yang tetap memadai meski dengan beban komputasi yang lebih rendah.

II. Theoretical Basis

A. Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) adalah salah satu teknik dekomposisi matriks yang sangat penting dalam aljabar linier dan memiliki banyak aplikasi, termasuk kompresi data. SVD didasarkan pada teorema yang menyatakan sebuah matriks A berukuran m x n dapat dipecah menjadi hasil perkalian dari tiga matriks, yaitu matriks ortogonal m x m (U), matriks berukuran n x n (Vt), dan matriks berukuran m x n yang elemen-elemen diagonal utamanya adalah nilai-nilai singular dari matriks A sedangkan elemen lainnya 0, dimana matriks ortogonal adalah matriks yang kolom-kolomnya adalah vektor yang saling ortogonal satu sama lain (hasil kali titik sama dengan 0)

$$A = U S V^T$$

Dimana:

- U adalah matriks ortogonal berukuran m x m, kolom-kolomnya terdiri dari vektor eigen ortonormal dari AA^T. Matriks ini menyimpan informasi terkait baris-baris matriks awal. Kolom pertama dari U mengandung informasi paling penting.
- S adalah matriks diagonal berukuran m x n yang berisi singular value dari matriks A, terurut dari yang terbesar hingga terkecil. Singular value adalah akar dari nilai eigen matriks AAt atau AtA

- V^T adalah matriks ortogonal berukuran n x n, kolom-kolomnya terdiri dari vektor eigen ortonormal dari $A^T A$. Matriks ini menyimpan informasi terkait kolom-kolom matriks awal. Baris pertama dari V^T mengandung informasi paling signifikan.

B. Kompresi Gambar Menggunakan SVD

SVD dapat digunakan untuk mengaproksimasi sebuah matriks data, termasuk matriks yang merepresentasikan gambar digital. Gambar digital biasanya dinyatakan sebagai matriks dua dimensi di mana setiap elemen matriks merepresentasikan intensitas piksel. Kompresi gambar dilakukan dengan mempertahankan sejumlah k singular value terbesar, di mana k jauh lebih kecil dari jumlah total singular value.

Proses ini melibatkan langkah-langkah berikut:

1. Dekomposisi Matriks
   Matriks A yang mempresentasikan gambar dipecah menjadi U, S, dan V^T

2. Pemilihan Singular Value k
   Hanya k singular value terbesar yang dipertahankan. Kolom pertama hingga kolom ke-k dari U dan V^T, serta elemen diagonal pertama hingga ke-k dari S, digunakan untuk merekonstruksi gambar.

3. Rekonstruksi Matriks Aproksimasi
   Matriks hasil rekontruksi diberikan oleh persamaan:
   $A = U_k S_k V_k^T$
   Dimana U_k, S_k, dan V_k^T hanya menyertakan k singular value terbesar

C. Efisiensi dan Kualitas Rekontruksi

Penggunaan singular value terbesar memungkinkan gambar terkompresi tetap menyerupai gambar asli dengan tingkat kesalahan minimal. Singular value terbesar menyimpan sebagian besar informasi penting dalam matriks, sementara singular value yang lebih kecil cenderung mewakili noise atau detail yang tidak signifikan. Oleh karena itu, dengan memilih k yang sesuai, ukuran gambar dapat dikurangi secara signifikan tanpa kehilangan kualitas visual.

D. Hubungan Rank Matriks dan Kompresi

Jumlah singular value yang dipertahankan (k) berkaitan langsung dengan rank matriks hasil aproksimasi. Rank ini menunjukkan kompleksitas data hasil rekonstruksi. Dalam konteks gambar, rank rendah menghasilkan matriks yang lebih sederhana, yang berarti ukuran data lebih kecil tetapi dengan potensi kehilangan detail visual.

III. Experiment

A. Setup

1. Perangkat Lunak
   - Operating System: Windows 11
   - Python 3.x
   - Libraries: NumPy, Pillow, Matplotlib, SciPy
2. Parameter Eksperimen
   Eksperimen dilakukan dengan memvariasikan nilai k (jumlah singular values yang dipertahankan):
   - k = 10 (kompresi tinggi)
   - k = 20 (kompresi menengah-tinggi)
   - k = 50 (kompresi menengah)
   - k = 100 (kompresi rendah)

B. Metrik Evaluasi

1. Compression Ratio
   Compression Ratio = Original Size / Compressed Size
   Semakin tinggi nilai rasio, semakin efektif kompresi dilakukan

2. Peak Signal-to-Noise Ratio (PSNR)
   PSNR = 20 * log10(MAX_PIXEL_VALUE / sqrt(MSE))
   PSNR ini digunakan untuk mengukur kualitas gambar hasil kompresi, dimana MSE (*Mean Squared Error*) adalah rata-rata kuadrat selisih nilai pixel antara gambar asli dan hasil kompresi.

3. Processing Time
   Waktu yang dibutuhkan untuk melakukan kompresi

C. Ilustrasi dan Simulasi Kompresi dalam Python

1. Diagram konseptual
   Berikut adalah diagram yang menjelaskan proses dekomposisi matriks dengan SVD:

   A (gambar asli)
   ↓
   SVD → U × S × V^T
   (m×k) (k×k) (k×n)
   ↓
   Aproksimasi → $A_k \approx$ A

2. Simulasi dalam python

Untuk simulasi, eksperimen akan dilakukan untuk kompresi gambar dengan kode sebagai berikut:

```
import numpy as np
from PIL import Image
import time
import matplotlib.pyplot as plt
from scipy.linalg import svd
import os

def compress_image_rgb(image_path, k):
    """
```

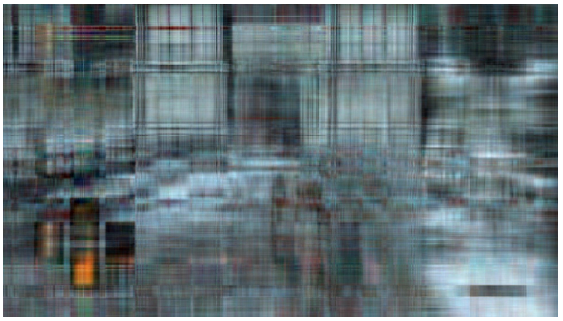
```
            Mengkompresi gambar RGB
menggunakan SVD dengan k komponen
singular values.
      """
      # Baca gambar dengan mode RGB
      img = Image.open(image_path)
      # Konversi ke array numpy
      img_array = np.array(img)

      # Catat waktu mulai
      start_time = time.time()

        # Kompresi untuk setiap channel
warna
      compressed_channels = []
        for channel in range(3):   # RGB
channels
          # Lakukan SVD
                          U,   s,   Vt   =
svd(img_array[:,:,channel],
full_matrices=False)

               # Rekonstruksi dengan k
komponen
                   compressed_channel   =
np.dot(U[:,     :k],     np.multiply(s[:k,
None], Vt[:k, :]))

compressed_channels.append(compressed_ch
annel)

      # Gabungkan channels
                        compressed      =
np.stack(compressed_channels, axis=2)

      # Hitung waktu kompresi
        compression_time  =  time.time()  -
start_time

      # Hitung ukuran data
       original_size = img_array.shape[0]
* img_array.shape[1] * 3  # 3 channels
             compressed_size   =   k   *
(img_array.shape[0] + img_array.shape[1]
+ 1) * 3  # 3 channels
       compression_ratio = original_size
/ compressed_size

                    return    compressed,
compression_ratio, compression_time

  def
save_compressed_image(compressed_array,
output_path, quality=85):
      """
        Menyimpan hasil kompresi sebagai
gambar dengan optimasi ukuran
      """
        # Normalize nilai pixel ke range
0-255
                    compressed_array    =
np.clip(compressed_array, 0, 255)
                    compressed_image    =
Image.fromarray(compressed_array.astype(
np.uint8))

          # Simpan sebagai JPEG dengan
kualitas tertentu untuk mengoptimalkan
ukuran
                        output_path     =
output_path.replace('.png', '.jpg')
       compressed_image.save(output_path,
'JPEG', quality=quality)

  def     analyze_compression(image_path,
k_values):
      """
         Menganalisis performa kompresi
untuk berbagai nilai k.
      """
      results = []
                             original     =
np.array(Image.open(image_path))

      for k in k_values:
           compressed, ratio, comp_time =
compress_image_rgb(image_path, k)
```

```
            # Hitung PSNR
                mse = np.mean((original - compressed) ** 2)
                psnr = 20 * np.log10(255 / np.sqrt(mse))

            # Simpan hasil
                                output_path = f"compressed_k{k}.jpg"

            save_compressed_image(compressed, output_path)

            # Hitung ukuran file hasil
                            compressed_size = os.path.getsize(output_path) / 1024  # Convert to KB
                             original_size = os.path.getsize(image_path) / 1024  # Convert to KB
             actual_ratio = original_size / compressed_size

            results.append({
                'k': k,
                       'compression_ratio': ratio,
                            'actual_ratio': actual_ratio,
                'psnr': psnr,
                'time': comp_time,
                'size_kb': compressed_size
            })

        return results, original_size
```

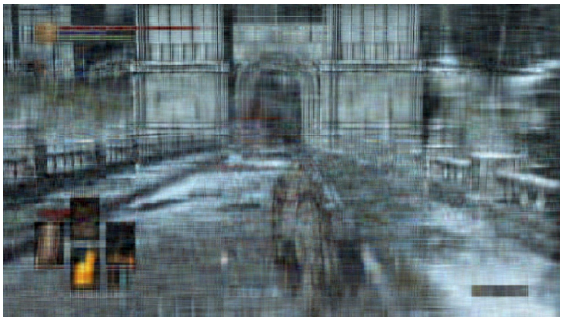
IV. Result

Experiment dilakukan pada salah satu frame dari permainan dark souls:

1. Hasil kompresi:
   k = 10



k = 20



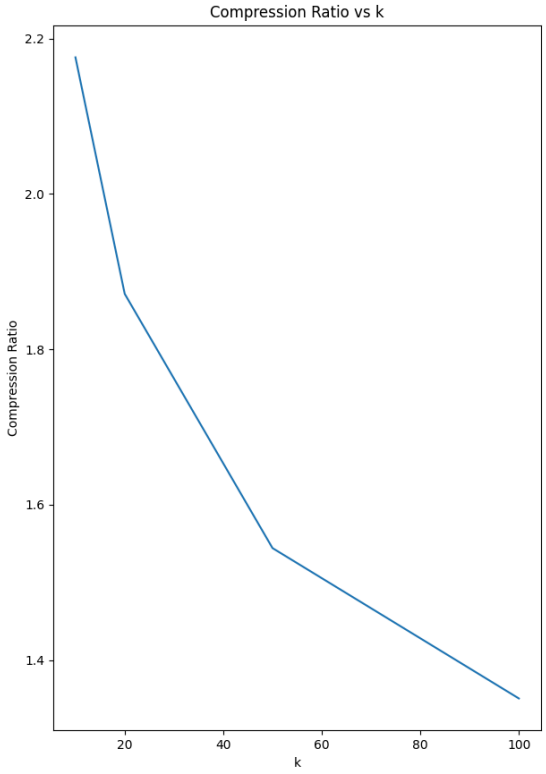
k = 50

k = 100

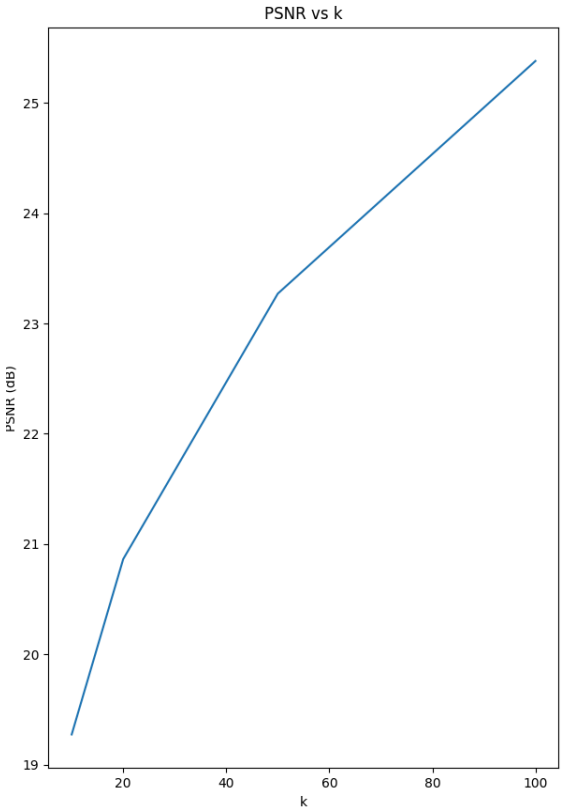
2. Tabel Perbandingan:

```
=== Eksperimen Kompresi SVD ===
Input image: darksouls.jpg
Original size: 623.90 KB

Hasil untuk berbagai nilai k:
------------------------------------------------------------------
  k |      Ratio |       PSNR |   Time (s) |   Size (KB)
------------------------------------------------------------------
 10 |       2.18 |      19.27 |     0.8719 |     286.75
 20 |       1.87 |      20.86 |     1.6038 |     333.39
 50 |       1.54 |      23.27 |     1.2176 |     404.04
100 |       1.35 |      25.38 |     0.8270 |     461.92
```

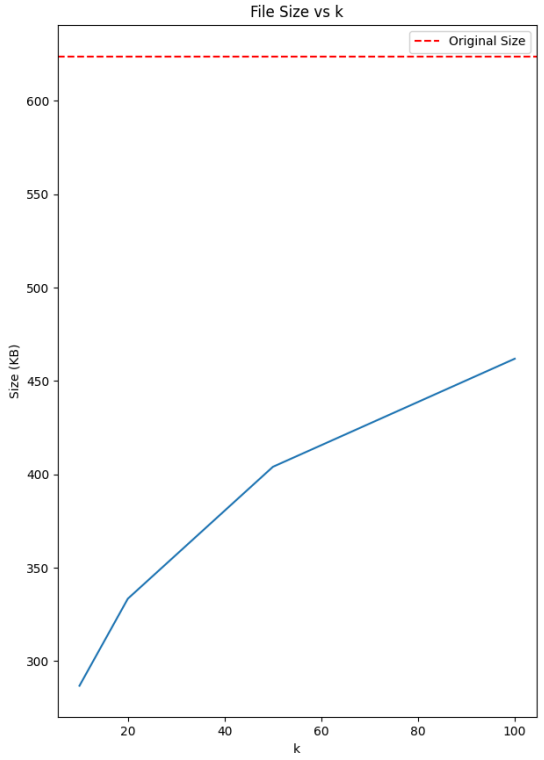
3. Analisis Compression Ratio



4. Analisis PSNR



5. Analisis Ukuran



V. Some Common Mistakes

1. Kesalahan dalam Pemilihan Parameter Kompresi. Kesalahan tersebut mengakibatkan:
   - Over-Compression (menyebabkan kehilangan detail yang signifikan)
   - Under-Compression (menyebabkan

pemborosan *resource storage* dan *memory*)

2. Kesalahan dalam Implementasi
    - Color Handling (dapat mengakibatkan *color bleeding* atau distorsi warna)
    - Memory Management (tekstur yang terkompresi sekaligus terlalu banyak)

VI. Conclusion

Metode dekomposisi nilai singular (Singular Value Decomposition/SVD) dapat menjadi langkah efektif sebagai metode kompresi tekstur dalam pengembangan game, dengan hasil yang menunjukkan keseimbangan optimal antara rasio kompresi dan kualitas visual melalui parameter k yang dapat disesuaikan. Tidak hanya untuk *game* secara internal, namun beberapa aplikasi ketiga juga dapat dibuat untuk membantu meningkatkan performa permainan dengan metode ini. Eksperimen menunjukkan bahwa metode ini mampu mengurangi ukuran file yang signifikan sambil mempertahankan kualitas visual yang masih dapat diterima oleh para pengguna. Untuk selanjutkan, karena akan berfokus pada pengembangan permainan, kode implementasi SVD dapat diatur agar dapat melakukan kompresi/dekompresi secara *real-time*, pengembangan metode adaptif untuk pemilihan nilai parameter k, dan integrasi yang lebih baik dengan game engine modern.

VII. Acknowledgment

Pertama, saya ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas tuntunan-Nyalah saya dapat menyelesaikan masalah ini. Kedua, saya ingin mengucapkan terima kasih untuk seluruh dosen mata kuliah IF2123 Aljabar Linear dan Geometri, khususnya untuk Dr. Judhi Santoso, M.Sc. dan Arrival Dwi Sentosa, S.Kom., M.T. selaku dosen K-03 Jatinangor yang sudah menuntun saya menjelajahi dunia aljabar linear dan geometri ini. Dan saya juga berterima kasih untuk semua pihak lain yang secara tidak langsung sudah membantu saya dalam menyelesaikan makalah ini.

References

[1] Munir, R. (2023). Singular Value Decomposition (SVD) (Bagian 1). Diakses pada 29 Desember 2024, dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf

[2] Munir, R. (2023). Singular Value Decomposition (SVD) (Bagian 2). Diakses pada 29 Desember 2024, dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-22-Singular-value-decomposition-Bagian2-2023.pdf

[3] M. Brady and D. Mathews, "Image Compression using Singular Value Decomposition (SVD)," Department of Mathematics, University of Utah, 2015. [Online]. Available: http://www.math.utah.edu/~goller/F15_M2270/BradyMathews_SVDImage.pdf

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.

Jatinangor, 2 Januari 2025

Ivant Samuel Silaban - 13523129