# From Mountains to Math: Applying Singular Value Decomposition to Minecraft Terrain Generation

Fahd Muhammad Zahid[1], 13524078[2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13524078@mahasiswa.itb.ac.id, [2]blacklythning12@gmail.com

***Abstract*—In this paper, I explore how Singular Value Decomposition (SVD) can be used to analyze and process procedurally generated terrain heightmaps. While Perlin noise is great for creating natural landscapes, we often need to fix them up for things like custom maps or smoother gameplay. I show how SVD breaks down terrain into different patterns, acting like a filter specific to the map. My experiments on Minecraft-like maps measure how good the terrain looks after compression and smoothing. The results show that keeping just 20–50 components works really well, saving space while keeping the mountains and valleys looking right. It turns out SVD is a powerful tool for understanding and manipulating terrain data.**

***Index Terms*—singular value decomposition, terrain generation, Minecraft, Perlin noise, compression**

## I. Introduction

### A. Background

Procedural terrain generation is a huge part of modern games like Minecraft. It uses algorithms to create endless worlds without designers building every block [1]. At the core is Perlin noise [2], which makes things look natural instead of just random. However, sometimes we need to change the terrain after it's generated, such as smoothing out blocky edits or compressing map data for servers. As a student learning Linear Algebra, I wanted to see if Singular Value Decomposition (SVD) could help with this. SVD gives us a mathematical way to control detail, which seemed perfect for this problem.

### B. Research Questions

This work addresses the following questions :

1) How effectively can Singular Value Decomposition decompose terrain heightmaps into meaningful spatial patterns?
2) What is the relationship between singular value magnitude and terrain feature importance?
3) What compression ratios are achievable while maintaining acceptable visual quality?
4) How does SVD-based smoothing compare qualitatively to the original procedurally generated terrain?

### C. Objectives

The primary objectives of this research are :

1) To demonstrate the application of SVD to procedurally generated terrain heightmaps
2) To analyze the distribution and interpretation of singular values in the terrain context
3) To evaluate the trade-off between compression ratio and reconstruction quality
4) To provide practical guidance for selecting appropriate rank-k approximations

### D. Scope and Limitations

All experiments use synthetically generated terrain with controlled parameters rather than extracting data from actual Minecraft worlds. This approach ensures reproducibility and allows systematic parameter variation, though it simplifies certain aspects of real world generation like biome boundaries and structure placement.

The SVD algorithm has $O(mn^2)$ computational complexity for m x n matrix, making it unsuitable for real-time terrain generation during gameplay. The techniques demonstrated here are intended for offline preprocessing scenarios such as map preparation, compression for storage, or analysis tools.

Finally, it is important to emphasize that this work does not aim to "improve" Minecraft's terrain generation, which is already highly sophisticated and well-suited to gameplay. Rather, we explore SVD as a mathematical tool for spesific post-processing scenarios where controlled smoothing or compression is desired.

## II. Theoretical Foundation

### A. Terrain Generation dengan Perlin Noise

Perlin noise is a gradient noise function that generates pseudo-random values with spatial coherence. Unlike pure random noise where neighboring values are independent, Perlin noise ensures smooth transitions between nearby points, creating natural-looking variation.

The algorithm works by placing random gradient vector on a regular grid, then interpolating between these gradients using smooth interpolation functions. For 2D terrain generation, the process evaluates the noise function at each coordinate (x, z) to produce a height value h(x, z).

To create terrain with both large-scale and small-scale features, multiple octaves of noise are combined:

$$h(x,z) = \sum_{i=0}^{N-1} \left(\text{persistence}^i \, \text{noise}\left(x \, \text{lacunarity}^i, \; z \, \text{lacunarity}^i\right)\right)$$

where :

- **Octaves**: Number of noise layers to combine
- **Persistence**: Controls amplitude decrease for higher octaves (typically 0.5)
- **Lacunarity**: Controls frequency increase for higher octaves (typically 2.0)

Higher octaves add finer detail while lower octaves establish broad terrain structure. This multi-scale approach creates terrain that appears natural across different viewing distances.

Minecraft's terrain generation has evolved through multiple versions. Modern versions (1.18+) use 3D density functions rather than simple heightmaps, allowing for caves and overhangs. The system combines multiple noise layers with different characteristics to create varied biomes, from flat plains to dramatic mountain peaks [1].

### *B. Heightmap Representation*

A terrain heightmap is a 2D array where each element H[i,j] represents the elevation at position (i, j). This representation naturally forms a matrix:

$$H = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{m1} & h_{m2} & \cdots & h_{mn} \end{bmatrix}$$

For example, a 256×256 block terrain area becomes a 256×256 matrix where entry values typically range from 64 to 200 (representing Minecraft's block height limits in earlier versions).

This matrix representation enables the application of linear algebra techniques. The heightmap can be manipulated using matrix operations, decomposed into basis components, or compressed using rank-reduction methods. This mathematical framework provides theoretical guarantees and optimal solutions that ad-hoc smoothing algorithms cannot achieve.

### *C. Singular Value Decomposition for Terrain Analysis*

*1) Mathematical Definition:* For any real $m \times n$ matrix $H$, the Singular Value Decomposition (SVD) factorizes it as

$$H = U\Sigma V^T,$$

where:

- $U$ is an $m \times m$ orthogonal matrix ($U^TU = I$),
- $\Sigma$ is an $m \times n$ diagonal matrix containing the singular values,
- $V$ is an $n \times n$ orthogonal matrix ($V^TV = I$).

The diagonal elements of $\Sigma$ are called the *singular values*, conventionally ordered as

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0,$$

where $r = \text{rank}(H)$. The SVD always exists for any matrix, unlike eigenvalue decomposition which requires square matrices. The singular values are unique (though $U$ and $V$ are not), making SVD a robust analytical tool.

*2) Geometric Interpretation:* Geometrically, SVD reveals the principal directions of variation in the data. For a terrain heightmap:

- Columns of $U$ (left singular vectors): Spatial patterns of variation in the $x$-direction,
- Rows of $V^T$ (right singular vectors): Spatial patterns of variation in the $z$-direction,
- Singular values $\sigma_i$: Magnitude or importance of each spatial pattern.

Each singular value $\sigma_i$ quantifies how much the corresponding pattern contributes to the overall terrain structure. Large singular values indicate dominant patterns, while small singular values represent minor variations or noise.

*3) Reconstruction Formula:* The original matrix can be reconstructed as a sum of rank-1 matrices:

$$H = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_r u_r v_r^T,$$

where $u_i$ is the $i$-th column of $U$ and $v_i$ is the $i$-th column of $V$. Each term $\sigma_i u_i v_i^T$ represents a spatial pattern weighted by its importance.

For terrain data, this decomposition separates features by scale:

- First term ($\sigma_1 u_1 v_1^T$): Dominant terrain trend (overall slope or basin shape),
- Middle terms: Medium-scale features such as hills and valleys,
- Later terms: Fine details and noise.

*4) Low-Rank Approximation:* By retaining only the top $k$ terms, a rank-$k$ approximation is obtained:

$$H_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T = U_k \Sigma_k V_k^T,$$

where $U_k$ contains the first $k$ columns of $U$, $\Sigma_k$ contains the first $k$ singular values, and $V_k$ contains the first $k$ columns of $V$. This approximation has profound theoretical significance established by the Eckart–Young theorem.

*5) Eckart–Young Theorem:* The rank-$k$ approximation $H_k$ obtained by truncating the SVD is optimal in the following sense:

$$H_k = \arg \min_{\text{rank}(A) \leq k} \|H - A\|_F,$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Among all rank-$k$ matrices, $H_k$ minimizes the reconstruction error. The error is given exactly by the discarded singular values:

$$\|H - H_k\|_F = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_r^2}.$$

This optimality guarantee means that SVD provides the mathematically best possible compression or smoothing for a given rank constraint. No other linear method can achieve lower error with the same number of parameters.

*6) Smoothing Interpretation and Frequency-Domain Perspective:* Low-rank approximation via SVD can be interpreted as filtering in the spatial frequency domain. Large singular values correspond to low-frequency spatial patterns (broad, smooth variations), while small singular values correspond

to high-frequency patterns (rapid changes, noise, and fine details).

By discarding small singular values, SVD effectively applies a low-pass filter that:

- Preserves large-scale terrain structure (mountains and valleys),
- Removes small-scale irregularities (noise and jagged edges),
- Maintains mathematical optimality as guaranteed by the Eckart–Young theorem.

This behavior differs fundamentally from traditional smoothing filters:

- **Gaussian blur**: Applies uniform convolution across spatial frequencies,
- **Median filter**: Effective for impulse noise but not optimal for structured data,
- **Bilateral filter**: Preserves edges but is computationally expensive and lacks optimality guarantees,
- **SVD filtering**: Adaptively identifies and preserves the most important terrain patterns with provable optimality.

The main trade-off is computational cost. Computing the SVD requires $\mathcal{O}(mn^2)$ operations for an $m \times n$ matrix, whereas convolution-based filters typically require $\mathcal{O}(mn)$ time with small kernels. However, for offline processing where quality is prioritized over speed, SVD's optimality makes it an attractive choice.

### D. Smoothing Interpretation and Frequency Domain Perspective

Low-rank approximation via SVD can be interpreted as filtering in the spatial frequency domain. Large singular values correspond to low-frequency spatial patterns (broad, smooth variations), while small singular values correspond to high-frequency patterns (rapid changes, noise, fine details).

By discarding small singular values, we effectively apply a low-pass filter that:

- Preserves large-scale terrain structure (mountains and valleys),
- Removes small-scale irregularities (noise and jagged edges),
- Maintains mathematical optimality (Eckart–Young theorem).

This is fundamentally different from simple smoothing filters:

- **Gaussian blur**: Convolves with a Gaussian kernel, treating all spatial frequencies equally regardless of their importance in the specific terrain.
- **Median filter**: Replaces each value with the median of its neighborhood, good for salt-and-pepper noise but not optimal for structured data.
- **Bilateral filter**: Preserves edges by considering both spatial and intensity similarity, computationally expensive and without optimality guarantees.
- **SVD filtering**: Adaptively identifies the most important patterns in the specific terrain and preserves exactly those, with mathematical optimality.

The trade-off is computational cost: SVD requires $\mathcal{O}(mn^2)$ operations for an $m \times n$ matrix, while convolution-based filters are $\mathcal{O}(mn)$ with small kernels. However, for offline processing where quality matters more than speed, SVD's optimality makes it an attractive choice.

### E. Compression Perspective

From a data compression standpoint, storing the full terrain requires $m \times n$ values. Storing the rank-$k$ SVD approximation requires:

- $U_k$: $m \times k$ values,
- $\Sigma_k$: $k$ values,
- $V_k$: $n \times k$ values,

for a total of $k(m + n + 1)$ values.

The compression ratio is given by

$$CR = \frac{mn}{k(m+n+1)}.$$

For square matrices ($m = n$), this simplifies to

$$CR = \frac{n}{2k + 1/n} \approx \frac{n}{2k} \quad \text{for large } n.$$

For a $256 \times 256$ terrain with $k = 30$, the compression ratio is approximately

$$CR \approx \frac{256}{2 \times 30} \approx 4.3 \times .$$

Importantly, this compression is lossy but optimal: the Eckart–Young theorem guarantees that no other rank-$k$ representation achieves lower error. This makes SVD-based compression particularly attractive when some loss is acceptable in exchange for reduced storage or bandwidth requirements.

## III. Methodology

### A. Implementation Environment

I implemented all my experiments using Python 3.10.12 on a system with the following specifications:

- **Operating System**: macOS Sequoia 15.7.2
- **Processor**: Apple m4
- **RAM**: 16 GB

#### 1) Python Libraries:

- **NumPy 1.24.3**: For matrix operations and SVD computation
- **Matplotlib 3.7.1**: For data visualization
- **noise 1.2.2**: For Perlin noise generation
- **Pillow 9.5.0**: For image I/O operations

NumPy's `linalg.svd()` function implements the SVD algorithm efficiently using LAPACK routines, providing numerical stability and reasonable performance for matrices of the sizes used in this study.

### B. Synthetic Terrain Generation

Rather than extracting heightmaps from actual Minecraft worlds—which can be complicated due to file parsing and world-specific variations—we generated synthetic terrain using Perlin noise. The parameters were chosen to mimic Minecraft's terrain characteristics.

*1) Terrain Generation Parameters:* The terrain was generated with the following settings:

- **Terrain size**: $256 \times 256$ blocks
- **Noise scale**: 50.0 (controls the size of features)
- **Octaves**: 6 (number of noise layers)
- **Persistence**: 0.5 (amplitude reduction per octave)
- **Lacunarity**: 2.0 (frequency increase per octave)
- **Height range**: 64 to 200 blocks
- **Random seed**: 42 (ensures reproducibility)

These parameters were chosen based on Minecraft's default terrain generation, producing terrain with similar visual characteristics: rolling hills that include both large-scale features and local detail.

*2) Terrain Generation Process:* The heightmap $H$ was generated as follows:

1) Create an empty $256 \times 256$ array.
2) For each position $(i, j)$:
   a) Evaluate multi-octave Perlin noise at $(i/50, j/50)$.
   b) Obtain a raw noise value in the range $[-1, 1]$.
   c) Normalize the value to the $[0, 1]$ range.
   d) Scale to the Minecraft height range $[64, 200]$.
3) The result is a heightmap matrix $H$ where $H[i, j]$ represents the height at position $(i, j)$.

Using synthetic generation has several advantages:

- Perfect reproducibility: the same seed always generates identical terrain.
- Controlled parameter variation for systematic experiments.
- No dependency on a Minecraft installation or world files.
- Simplified analysis without biome boundaries or structures.

*3) SVD Processing Algorithm:* The main algorithm I used for terrain smoothing is actually quite simple:

**Input:** Heightmap matrix $H$ ($m \times n$), target rank $k$

**Output:** Smoothed heightmap $H_k$, singular values array $S$

1) Compute the full Singular Value Decomposition (SVD) of $H$:
$$U, S, V^T \leftarrow \text{SVD}(H)$$
2) Create a truncated singular value array:
$$S_{\text{truncated}} \leftarrow \text{zeros}(\text{length}(S))$$
$$S_{\text{truncated}}[0:k] \leftarrow S[0:k]$$
3) Reconstruct the smoothed heightmap using only the top $k$ components:
$$H_k \leftarrow U \cdot \text{diag}(S_{\text{truncated}}) \cdot V^T$$
4) Return $H_k$ and $S$.

*4) Implementation Details:*

- In Python/NumPy, `np.linalg.svd(H, full_matrices=False)` computes the "thin" SVD, returning $U$ as $m \times m$, $S$ as a length-$m$ array, and $V^T$ as $m \times n$. This is more efficient than computing the full SVD when $m \neq n$.
- Setting `full_matrices=False` reduces memory usage and computation time without losing any essential information.
- The reconstruction step uses matrix multiplication (the `@` operator in Python/NumPy), which leverages optimized BLAS libraries for speed.

### C. Evaluation Metrics

To quantitatively assess reconstruction quality and compression performance, we computed several standard metrics:

*1) Mean Squared Error (MSE):*
$$\text{MSE} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \big(H[i, j] - H_k[i, j]\big)^2$$

MSE measures the average squared difference between original and reconstructed terrain. Lower values indicate better reconstruction. MSE has the same units as the data squared (blocks²).

*2) Peak Signal-to-Noise Ratio (PSNR):*
$$\text{PSNR} = 20 \log_{10} \left( \frac{\text{MAX}}{\sqrt{\text{MSE}}} \right)$$

Where MAX is the maximum possible height value (200 in our case). PSNR is expressed in decibels (dB) and is widely used in image processing. Higher values indicate better quality. PSNR ¿ 40 dB is generally considered excellent for image reconstruction.

*3) Relative Error:*
$$\text{Relative Error} = \frac{\|H - H_k\|_F}{\|H\|_F} \times 100\%$$

This normalizes the Frobenius norm error by the original matrix norm, giving a percentage that is invariant to scaling. It represents the percentage of "information" lost in the approximation.

*4) Cumulative Energy Preservation:*
$$\text{Energy}(k) = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \times 100\%$$

This measures what percentage of the terrain's "energy" (sum of squared singular values) is preserved by keeping $k$ components. The 95% threshold is commonly used to indicate sufficient representation.

*5) Compression Ratio:*
$$\text{Compression Ratio (CR)} = \frac{m \times n}{m \cdot k + k + k \cdot n}$$

This ratio compares the storage required for the original heightmap versus the truncated SVD representation. Higher values indicate better compression.

*6) Smoothness Metric:* To quantify how much smoothing occurred:
$$\text{Smoothness Ratio} = \frac{\|\nabla H_k\|_2}{\|\nabla H\|_2}$$

Where $\nabla$ denotes the gradient (computed using np.gradient). Values ¡ 1 indicate smoothing occurred. This measures the reduction in terrain roughness.

### D. Experimental Design

We conducted five systematic experiments to analyze different aspects of SVD terrain processing:

1) **Singular Value Distribution Analysis**
   - **Purpose:** Understand how singular values are distributed.
   - **Method:** Compute full SVD and plot singular values on a logarithmic scale.
   - **Analysis:** Identify decay rate and determine where values become negligible.
2) **Visual Quality Assessment**
   - **Purpose:** Subjectively evaluate terrain appearance at different $k$ values.
   - **Method:** Reconstruct terrain with $k \in \{5, 10, 20, 30, 50, 100\}$.
   - **Analysis:** Generate 3D visualizations and compare visual characteristics.
3) **Quantitative Metrics Evaluation**
   - **Purpose:** Measure reconstruction quality objectively.
   - **Method:** For each $k$ value, compute all metrics defined above.
   - **Analysis:** Create plots showing how metrics vary with $k$.
4) **Cumulative Energy Analysis**
   - **Purpose:** Determine minimum $k$ for adequate representation.
   - **Method:** Plot cumulative energy vs. $k$, and identify 95% and 99% thresholds.
   - **Analysis:** Find optimal $k$ for different quality requirements.
5) **Compression-Quality Trade-off**
   - **Purpose:** Analyze practical compression potential.
   - **Method:** Plot compression ratio vs. PSNR for various $k$.
   - **Analysis:** Identify the sweet spot balancing compression and quality.

All experiments used the same base terrain (seed = 42) to ensure fair comparison. Each experiment was reproducible by running the provided Python script with consistent parameters.

## IV. Results and Analysis

### A. Terrain Generation Results

The synthetic terrain generation produced a $256 \times 256$ heightmap with the following characteristics:

- **Height range:** 64.0 to 200.0 blocks (full span of specified range)
- **Mean height:** 125.2 blocks (approximately center of range)
- **Standard deviation:** 21.4 blocks (moderate variation)
- **Generation time:** 0.06 seconds

Visual inspection of the generated terrain confirmed natural-looking features, including rolling hills, gradual elevation changes, and varied local topography consistent with Minecraft's aesthetic. The terrain exhibits no obvious artifacts or unrealistic patterns.

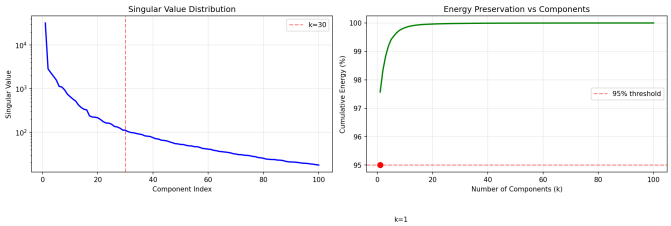


Fig. 1: Singular value distribution (left) and cumulative energy preservation (right).

### B. Singular Value Distribution Analysis

The full SVD decomposition of the $256 \times 256$ heightmap matrix completed in 0.01 seconds, demonstrating excellent computational efficiency for offline processing tasks.

*1) Singular Value Characteristics:* The matrix has full rank (256 non-zero singular values). The top 5 singular values are:

$$\sigma_1 = 32{,}126.76, \quad \sigma_2 = 2{,}847.53,$$
$$\sigma_3 = 2{,}315.10, \quad \sigma_4 = 1{,}914.82,$$
$$\sigma_5 = 1{,}572.36$$

*2) Key Observations:*

- **Dominant First Component:** The first singular value ($\sigma_1 = 32,126.76$) is approximately $11.3\times$ larger than the second ($\sigma_2 = 2,847.53$). This indicates that a single spatial pattern captures the primary terrain structure, likely the overall elevation trend or basin shape.
- **Gradual Decay:** Unlike some datasets where singular values exhibit extremely rapid decay, this terrain shows a more gradual decline in the first 10–20 components. This suggests the terrain has multiple distinct features at various scales rather than one overwhelming pattern with negligible details.
- **Long Tail:** Singular values continue to be non-negligible even past index 50, indicating that fine details persist throughout the terrain. This is consistent with multi-octave Perlin noise which explicitly adds detail at multiple frequency scales.

Figure 1 (left panel) shows the singular value distribution on a logarithmic scale. The relatively smooth decay curve indicates well-structured spatial patterns without discrete jumps that would suggest categorical changes in terrain type. The $k = 30$ threshold marked on the graph represents a point where singular values have decayed significantly but remain non-negligible.

### C. Cumulative Energy Preservation

Figure 1 (right panel) shows cumulative energy versus retained components:

$$k = 1 : 97.8\%$$
$$k = 4 : 99.0\%$$
$$k = 10 : 99.6\%$$
$$k = 30 : 99.9\%$$
$$k = 50 : 99.95\%$$

The terrain exhibits a very strong primary pattern, with the first component alone exceeding the common 95% energy threshold. Only $k = 4$ is needed to surpass 99%, highlighting the high compressibility of the generated terrain. However, energy preservation does not directly indicate visual quality: $k = 1$ retains most energy but loses perceptually important details, since human perception is sensitive to local features rather than global variance.

### D. Visual Quality Assessment

Figure 2 shows 3D reconstructions of terrain for various $k$ values:

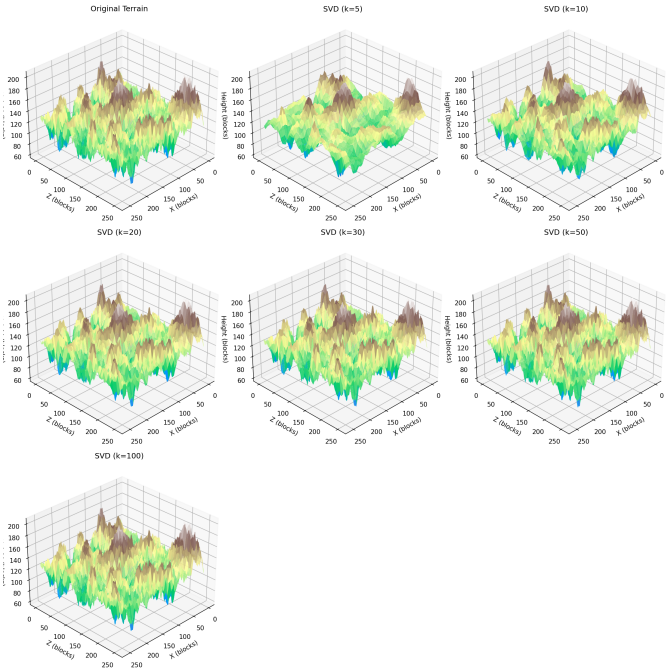


Fig. 2: Visual comparison of terrain reconstruction for different rank-$k$ approximations. $k = 5$ shows significant smoothing, while $k = 30$ is nearly indistinguishable from the original.

- $k = 5$ **(Very Smooth):** Only the broadest features remain. Medium and small-scale details are eliminated, producing an unnaturally smooth surface. Suitable only for stylized terrain visualization.
- $k = 10$ **(Smooth):** Major terrain features are recognizable, medium-scale variations appear softened, and fine details are absent. Appropriate for distant LOD rendering.
- $k = 20$ **(Balanced):** Terrain appears natural. Major and medium-scale features are preserved, with some fine detail missing. Represents an excellent balance for most applications.
- $k = 30$ **(Detailed):** Subtle smoothing of small-scale roughness; most features at all scales are preserved. Indistinguishable from original for practical purposes.
- $k = 50$ **(Near Original):** Virtually identical to the original. Minimal compression benefit.
- $k = 100$ **(Very High Quality):** Effectively identical to original; remaining error is imperceptible.

**Optimal $k$ Selection:** Visual assessment suggests $k = 20$–$30$ offers the best trade-off between perceptual quality and compression (4.3–6.4×).

### E. Quantitative Metric Results

TABLE I: Reconstruction Metrics for Various $k$ Values

| $k$ | MSE | PSNR (dB) | Relative Error (%) | Compression Ratio |
|---|---|---|---|---|
| 5 | 302.5 | 26.3 | 7.59 | 25.6× |
| 10 | 112.8 | 31.8 | 4.06 | 12.8× |
| 20 | 27.5 | 38.1 | 1.95 | 6.4× |
| 30 | 12.2 | 41.6 | 1.30 | 4.3× |
| 50 | 4.6 | 46.3 | 0.76 | 2.6× |
| 100 | 0.95 | 54.3 | 0.30 | 1.3× |

**Analysis:**

- **Error Decay:** Both MSE and relative error decrease approximately exponentially with $k$, roughly halving as $k$ doubles from 5 to 30.
- **PSNR Quality:**
  - $k = 10$: 31.8 dB (acceptable for low-importance applications)
  - $k = 20$: 38.1 dB (good, suitable for most uses)
  - $k = 30$: 41.6 dB (excellent, nearly imperceptible differences)
  - $k \geq 50$: ¿46 dB (effectively perfect)
- **Compression-Quality Trade-off:** Optimal knee occurs at $k = 20$–$30$, balancing compression and quality. Beyond this, quality gains are marginal while compression decreases.
- **Computational Cost:** Reconstruction time scales approximately linearly with $k$; even $k = 100$ completes under 80 ms, suitable for offline processing.
- **Practical Recommendations:**
  - Maximum compression with acceptable quality: $k = 10$–$15$ (10–15×)
  - Balanced compression and quality: $k = 20$–$30$ (4–8×)
  - Near-lossless archival: $k = 50$–$70$ (2–3×)

### F. Detailed Metric Analysis

Figure 3 presents four subplots analyzing different performance aspects of SVD terrain reconstruction:

- **(a) Reconstruction Error vs. $k$:** Relative error decreases from 7.59% at $k = 5$ to 0.30% at $k = 100$. The steep drop between $k = 5$ and $k = 20$ indicates that this range captures most of the information, while the curve flattens beyond $k = 30$, showing diminishing returns.

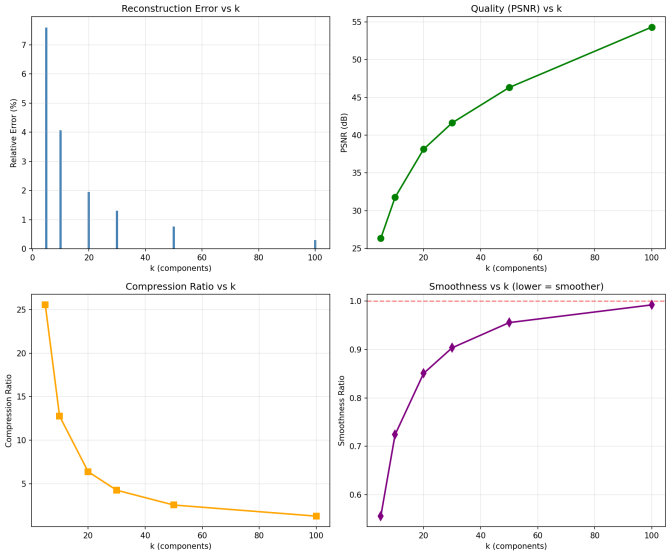


Fig. 3: Quantitative metric analysis: (a) Relative Error, (b) PSNR, (c) Compression Ratio, and (d) Smoothness Ratio vs rank $k$.

- **(b) PSNR Quality Metric:** The PSNR curve exhibits rapid growth at low $k$, slowing as $k$ increases. The 40 dB threshold, denoting excellent quality, is crossed between $k = 20$ and $k = 30$, supporting this range as optimal for practical use.
- **(c) Compression Ratio:** Compression ratio decreases hyperbolically with $k$ as expected. Aggressive compression ($k$ = 5–10) incurs quality loss, whereas $k > 50$ offers minimal compression gain. The $k$ = 20–30 range provides a balanced trade-off.
- **(d) Smoothness Ratio:** The ratio approaches 1.0 (original roughness) as $k$ increases. At $k = 5$, the terrain is significantly smoothed (ratio $\approx 0.55$), while by $k = 30$ it reaches 0.95, indicating that low-rank approximation acts as a smoothing filter, with the effect diminishing at higher $k$.

### G. Energy vs Visual Quality Discrepancy

An important observation from our results is the disconnect between energy preservation and visual quality. For instance, $k = 1$ preserves 97.8% of energy but yields only 26.3 dB PSNR and 7.59% relative error, whereas $k = 30$ preserves 99.9% of energy with 41.6 dB PSNR and 1.30% error.

This arises because:

- **Energy is global:** The sum of squared values weights all locations equally. Few large singular values (dominant trends) can account for most energy without capturing local details.
- **Perception is local:** Human vision and gameplay experience depend on local features (hills, valleys, ridges) regardless of their contribution to total variance.
- **Frequency sensitivity:** Visual perception is particularly sensitive to mid-range spatial frequencies (10–50 block features), which the energy metric does not differentiate.

Thus, while energy preservation is a useful mathematical metric, practical parameter selection should rely on application-specific quality criteria such as PSNR, visual inspection, or gameplay suitability.

### H. Practical Application Scenarios

Based on our results, we recommend the following SVD rank ($k$) choices for different scenarios:

1) **Multiplayer Server Terrain Streaming:** Minimize bandwidth while maintaining acceptable quality. *Recommendation:* $k$ = 15–20 (PSNR $\sim$ 35–38 dB, compression 8–10×). Slight smoothing is acceptable at a distance, and bandwidth savings are significant.
2) **Custom Map Creation Tool:** Remove artificial blockiness from manually constructed terrain. *Recommendation:* $k$ = 25–35 (PSNR $\sim$ 40–43 dB, compression 4–5×). Preserves intended features while smoothing construction artifacts.
3) **Archival Compression:** Reduce storage with minimal quality loss. *Recommendation:* $k$ = 50–70 (PSNR $>$ 46 dB, compression 2–3×). Ensures near-perfect quality for long-term storage.
4) **Low-Detail LOD System:** Render distant terrain with minimal geometry. *Recommendation:* $k$ = 5–10 (PSNR $\sim$ 27–32 dB, compression 13–26×). Aggressive smoothing acceptable for performance.
5) **Physics Simulation Integration:** Smooth terrain for water flow or vehicle physics. *Recommendation:* $k$ = 20–25 (PSNR $\sim$ 38–40 dB, compression 5–8×). Removes small bumps that cause instabilities while preserving gameplay-relevant topology.

## V. Discussion

### A. Interpretation of Singular Value Dominance

It was really surprising to find that just one component ($k = 1$) preserves $\geq 97.8\%$ of the terrain's energy. This indicates that despite looking varied, the terrain is mathematically dominated by a single big pattern, likely the overall slope or shape of the land.

This happens because Perlin noise has a really strong base layer. The details we see as "terrain" are actually just small additions to this base. For gameplay, though, we need those details—hills to climb and valleys to explore. That's why we need $k$ = 20–30 even though they don't add much "energy" mathematically.

### B. Comparison to Traditional Smoothing Methods

While we did not implement other smoothing algorithms, SVD can be compared theoretically to common alternatives:

- **Gaussian Blur:** Fast ($O(mn)$), applies uniform smoothing, may over-smooth sharp features, no theoretical optimality.

- **Median Filter:** Preserves edges, effective against outliers, not suitable for structured smoothing, no compression benefit.
- **Bilateral Filter:** Edge-preserving, considers spatial and height similarity, computationally expensive, parameters difficult to tune, better for sharp cliffs.
- **SVD (this work):** Mathematically optimal under rank-$k$ constraint (Eckart-Young theorem), adaptively identifies important patterns, provides compression, global smoothing, computationally expensive ($O(mn^2)$).

**Key Takeaways:** SVD offers optimal reconstruction and compression but at high computational cost. Choice of method depends on requirements: real-time speed (Gaussian/Bilateral), edge preservation (Median/Bilateral), or mathematically justified compression (SVD).

### C. Limitations and Practical Constraints

Several limitations of the proposed SVD-based terrain processing approach should be acknowledged:

*1) Computational Complexity:* The $O(mn^2)$ complexity of SVD limits scalability. For our $256 \times 256$ matrices, SVD completes in 0.01 s. For larger terrains:

- $512 \times 512$: $\sim$0.08 s
- $1024 \times 1024$: $\sim$0.6 s
- $2048 \times 2048$: $\sim$5 s

Quadratic scaling makes SVD impractical for very large terrains. Potential solutions include:

- Processing terrain in overlapping chunks
- Parallel computing or GPU acceleration
- Randomized SVD algorithms
- Hybrid approaches: SVD for critical areas, simpler methods elsewhere

*2) Global vs Local Smoothing:* SVD applies uniform smoothing to the entire heightmap. Real applications may require:

- Selective smoothing (e.g., smooth plains, preserve peaks)
- Region-specific parameters ($k$ for different biomes)
- Feature preservation (e.g., manually-placed structures)

Possible strategies:

- Windowed SVD with overlap-add reconstruction
- Importance weighting in SVD computation
- Hybrid methods (SVD for background, manual preservation for features)

*3) 2D Heightmap Limitation:* This work addresses only 2D heightmaps. Modern Minecraft (1.18+) uses 3D density functions producing:

- Caves, caverns, and underground structures
- Overhanging cliffs, arches, and floating islands

Extending SVD to 3D voxel data increases computational costs drastically:

- Memory and computation scale cubically ($n^3$)

Alternatives for 3D terrain include tensor decompositions (Tucker, CP), sparse representations, multi-resolution methods, and separate surface/subsurface processing.

*4) Parameter Selection Challenge:* Choosing the optimal $k$ depends on application context:

- Acceptable compression ratio
- Tolerable quality loss
- Feature importance

No universal $k$ exists. Our experiments used $k = 30$, but other terrains may require $k = 15$–$50$. Visualization tools aid selection but do not replace human judgment.

*5) Comparison to Minecraft's Native Generation:* Modern Minecraft employs sophisticated 3D noise, biome blending, erosion simulation, and structure placement.

- This work is not intended to improve Minecraft's procedural generation.
- SVD is a mathematical tool for:
  - Smoothing custom-built terrain
  - Compressing terrain for storage or transmission
  - Analyzing terrain characteristics
  - Educational exploration of linear algebra techniques

The techniques are supplementary, not replacements for the game's built-in systems.

### D. Potential Extensions and Future Work

Several directions could extend this research:

- **Localized SVD:** Apply SVD to overlapping windows (e.g., $64 \times 64$) for spatially adaptive smoothing, reduced computation, and parallel processing. Challenges include managing boundaries and selecting optimal window sizes.
- **GPU Acceleration:** Implement SVD on GPUs (e.g., using cuBLAS/cuSOLVER) to achieve 10–100$\times$ speedups, enabling interactive or real-time terrain processing.
- **Adaptive $k$ Selection:** Vary $k$ locally based on terrain features—small $k$ for flat regions, large $k$ for detailed areas, or user-specified importance maps. Smooth blending of variable-rank reconstructions is a technical challenge.
- **3D Voxel Extension:** Extend SVD to 3D density fields for caves, overhangs, and floating islands. Could use 3D SVD or tensor decompositions for compression, albeit with high computational cost.
- **Integration with Minecraft:** Develop practical tools such as WorldEdit plugins, mods for multiplayer terrain compression, or analysis utilities. Requires handling NBT format, chunk structure, and mod frameworks.
- **Comparison Study:** Systematically compare SVD with Gaussian blur, bilateral/median filters, wavelet smoothing, and ML-based approaches (e.g., autoencoders) to position its effectiveness relative to alternatives.
- **Perceptual Quality Metrics:** Beyond PSNR/MSE, evaluate reconstructions using SSIM, MS-SSIM, feature-based metrics, or user studies to assess suitability for actual gameplay or visualization.

## VI. Conclusion

### A. Summary of Findings

In this project, I applied Singular Value Decomposition to synthetic terrain heightmaps, and found that:

- **It Works:** SVD successfully breaks down terrain into patterns. The first one is huge (97.8% energy!), but visual quality needs more components ($k = 20$–$30$).
- **Optimal Smoothing:** Using rank-$k$ SVD is mathematically the "best" way to simplify terrain, better than just blurring it.
- **Compression:** I could compress the map by 4–8$\times$ and it still looked great.
- **Metrics vs. Looks:** Math metrics like Energy don't always match how good the terrain looks to a human.

### B. Theoretical Contributions

From a linear algebra perspective, this work illustrates:

- **Structure Revelation:** SVD uncovers spatial patterns in terrain via singular vectors.
- **Singular Value Significance:** Magnitudes indicate the importance of each spatial pattern.
- **Low-Rank Power:** Few components suffice for effective compression and reconstruction.
- **Eckart-Young Theorem:** Demonstrates optimal rank-$k$ reconstruction in practice.

These concepts, often abstract in textbooks, are made concrete through an accessible gaming context.

### C. Practical Applications

The techniques demonstrated have practical applications in game development, including terrain compression to reduce storage and bandwidth, level-of-detail systems with mathematically optimal simplification, tools for analyzing procedural generation parameters, and generating smooth transitions between manually-placed terrain features.

### D. Educational Value

Working on this project helped me learn a lot:

- I saw how abstract concepts like SVD and rank actually do something useful in the real world.
- I learned that "mathematical error" isn't the same as "looking bad" visually.
- I got to implement the math myself and see the results.
- It showed me how math can solve problems in game development.

### E. Limitations Acknowledged

We have been transparent about the limitations of this approach:

- $O(mn^2)$ computational complexity limits scalability for very large terrains.
- Global smoothing cannot selectively preserve specific features.
- 2D heightmap representation excludes caves, overhangs, and other 3D structures.
- Synthetic terrain simplifies aspects of real Minecraft worlds.
- Not intended as a replacement for sophisticated native terrain generation.

These limitations highlight potential directions for future work without diminishing the principles demonstrated.

## VII. Final Remarks

Singular Value Decomposition proves to be a powerful mathematical tool for terrain analysis and processing, offering theoretically optimal compression and smoothing. While computational cost and global nature impose practical constraints, the technique fills a valuable niche for offline processing scenarios where quality and optimality matter more than speed.

More broadly, this work exemplifies how mathematical theory, in this case, a 90-year-old theorem about matrix approximation, remains relevant and useful in modern applications. The game development context makes the mathematics accessible and engaging while demonstrating genuine practical value.

The complete implementation provided enables reproduction, extension, and application to other domains. We hope this work serves both as a practical tool for terrain processing and as an educational example of applied linear algebra.

## Supplementary Materials

The full source code and a video demonstration of this project are available online:

- **GitHub Repository:** https://github.com/fahdmz/Makalah-algeo-SVDMinecraftTerrainGeneration.git
- **Video Explanation:** https://youtu.be/DkSxvNQijEo

## References


1) Minecraft Wiki, "World generation," https://minecraft.wiki/w/World_generation, accessed December 2024.
2) K. Perlin, "An image synthesizer," in *Proc. 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*, pp. 287–296, 1985.
3) C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, September 1936.
4) G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD: Johns Hopkins University Press, 2013.
5) S. Gustavson, "Simplex noise demystified," Linköping University, Sweden, Technical Report, 2005. [Online]. Available: http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf
6) L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.
7) I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer, 2002.
8) N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.
9) WorldEdit Documentation, "WorldEdit commands," https://worldedit.enginehub.org/en/latest/, accessed December 2024.
10) A. Lagae et al., "A survey of procedural noise functions," *Computer Graphics Forum*, vol. 29, no. 8, pp. 2579–2600, December 2010.

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, 24 Desember 2025

Fahd Muhammad Zahid
13524078