Application of Line and Plane Intersection Geometry for Gamut Mapping in OKLab Vector Space

Steven Tan, 13524060[1, 2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13524060@std.stei.ac.id, [2]steventan6002@gmail.com

***Abstract*—The abstract is to be in fully-justified italicized text, at the top of the left-hand column as it is here, below the author information. The abstract is to be in 9-point, single-spaced type, and may be up to 8 cm long. Define all symbols used in the abstract. Do not cite references in the abstract. Do not delete the blank line immediately above the abstract; it sets the footnote at the bottom of this column. Leave two blank lines after the index terms, then begin the main text. All manuscripts must be in English.**

***Keywords*—About four key words or phrases in alphabetical order, separated by commas.**

I. Introduction

The evolution of digital display technology has created a significant discrepancy between content capture and content reproduction. While modern capture devices and media formats increasingly use wide-gamut color spaces such as Display P3 and Rec.2020, the vast majority of consumer displays remain constrained to the standard sRGB gamut [1]. Display P3, for instance, covers ~45% of the visible color spectrum, whereas sRGB covers only ~35%, creating a mismatch where valid source colors effectively do not exist within the destination's gamut.

This necessitates gamut mapping, the process of transforming vectors from a source gamut domain into a destination gamut domain. The fundamental challenge of gamut mapping is not merely data compression, but the preservation of perceptual attributes, specifically hue and lightness.

Conventional approaches often employ component-wise clipping, where individual basis coefficients (Red, Green, Blue) are truncated to the display's maximum values (e.g., if $R \geq 1$, then $R = 1$). While computationally inexpensive, this method is geometrically flawed. In a vector space, component-wise clipping alters the ratios between the basis components, effectively rotating the vector and changing its direction. Perceptually, this manifests as a hue shift, where a saturated color (e.g., bright blue) is distorted into a different hue (e.g., purple) as it is projected onto the gamut boundary [2].

To address this, colorimetric vectors must be mapped in a way that preserves their original direction. However, standard RGB spaces are perceptually non-uniform. A linear line in RGB space does not correspond to a linear line of perceived hue to the human eye. Therefore, geometric operations must be performed in a perceptually uniform vector space.

This paper proposes an application of line-plane intersection within the OKLab vector space to solve the gamut mapping problem. By modeling the destination sRGB gamut as a convex polytope, a mesh of triangular planes, and the input color as a parametric ray, we calculate the exact geometric intersection that minimizes chroma's error while preserving the vector's direction [3]. This approach leverages the hue linearity of OKLab [4] to ensure that the mathematical projection aligns with human visual perception.

II. Theoretical Framework

A. Color as Vector Spaces

Fundamentally, a digital color can be modeled as a vector v in a three dimensional vector space V. In the standard sRGB model, this vector space is spanned by three basis vectors corresponding to the display's additive primaries: Red, Green, and Blue. Any color vector v can be expressed as a linear combination of these basis vectors [1]:

$$\mathrm{v} = r\mathrm{e}_r + g\mathrm{e}_g + b\mathrm{e}_b$$

where the basis vectors are defined in the basis as

$$\mathrm{e}_r = [1,0,0]^T,$$

$$\mathrm{e}_g = [0,1,0]^T,$$

$$\mathrm{e}_b = [0,0,1]^T.$$

The scalars $r, g, b \in [0,1]$ represent the normalized intensities of the primaries. Consequently, the set of all reproducible colors in the RGB model forms a geometric unit cube, defined as the subset $C_{rgb} \subset \mathbb{R}^3$:

$$C_{rgb} = \{(r, g, b) \mid 0 \leq r, g, b \leq 1\}$$

However, raw RGB signals stored in digital files are inherently non-linear due to Gamma Correction, a non-linear transfer function $f(x)$ applied to match the human eye's non-linear sensitivity to luminance (Stevens' Power Law). Standard linear algebraic operations—such as vector addition, scalar multiplication, and basis transformations—are mathematically invalid in a non-linear space. Therefore, the input signals must first be linearized via the inverse transfer function (EOTF) before any geometric processing occurs [1]:

$$C_{linear} = \left(\frac{C_{srgb} + 0.055}{1.055}\right)^{2.4} \text{ for } C_{srgb} > 0.04045$$

Once linearized, the color space acts as a valid linear space, allowing for the application of matrix transformations to map coordinates between different bases (e.g., from the Display P3 basis to the device-independent CIE XYZ basis).

B. Perceptual Uniformity and OKLab Space

While CIE XYZ provides an absolute, device-independent reference frame, it is not perceptually uniform. The Euclidean metric is distorted in XYZ; the distance $|| \mathrm{v}_1 - \mathrm{v}_2 ||$ does not correlate with the perceived color ΔE. A geometric operation (like finding the shortest path) in XYZ does not correspond to the shortest perceptual path.

To resolve this, we uaw OKLab space [2]. Unlike the older CIELAB standard, OKLab performs the non-linear compression step in the LMS cone response domain rather than the XYZ domain. The transformation $T: XYZ \rightarrow OKLab$ involves a sequence of linear and non-linear maps:

1. Linear Basis Change (XYZ to LMS): A 3×3 matrix transformation M_1 maps the physical XYZ values to the biological response of the Long, Medium, and Short retinal cones.

2. Non-Linear Compression: A cube-root function is applied component-wise to approximate the eye's response to intensity.

3. Linear Basis Change (LMS to Lab): A second matrix M_2 rotates the axes to separate Achromatic Lightness (L) from the Chromatic opponent axes (a and b).

This derivation ensures Hue Linearity, straight lines radiating from the neutral axis in OKLab correspond to constant perceptual hue [2]. In a polar coordinate system, Hue (θ) and Chroma (C) are derived from the Cartesian coordinates (a, b) as:

$$C = \sqrt{a^2 + b^2}, \theta = \arctan(b/a)$$

Because OKLab preserves hue along radial vectors, we can use Linear Ray-Casting to compress Chroma without distorting the Hue angle, a property not present in standard sRGB or CIELAB.

C. Gamut as a Convex Polytope

A Gamut is defined as the subset of colors reproducible by a specific device. Geometrically, the sRGB gamut is the Convex Hull of the set of all valid linear combinations of the sRGB basis vectors.

In the linear RGB space, this gamut is a perfect cube with 8 vertices (Black, Red, Green, Blue, Cyan, Magenta, Yellow, White) and 12 triangular faces. However, when transformed into the perceptually uniform OKLab vector space, the non-linear compression warps this cube into a non-regular, curved volume. To perform intersection tests, we approximate this curved boundary as a Convex Polytope ($\mathcal{P}$) constructed using a discrete Triangular Mesh.

We generate this mesh by tessellating the six faces of the RGB cube and transforming the vertices into OKLab. The boundary is then defined as a collection of planar triangular faces. Each face f_i is defined by three vertices (V_0, V_1, V_2) and a surface normal vector $\vec{n}$.

The normal vector $\vec{n}$ is critical for determining the orientation of the plane (defining "inside" vs "outside"). It is computed via the Cross Product of two edge vectors. We adhere to a counter-clockwise winding order to ensure the normal points outward from the gamut volume:

$$\vec{n} = (V_1 - V_0) \times (V_2 - V_0)$$

This geometric construction reduces the complex, non-linear gamut boundary into a set of linear planes, enabling efficient algebraic intersection tests.

D. Line-Plane Intersection Geometry

The gamut mapping problem is reframed as a computational geometry problem, finding the intersection of a parametric ray and a plane.

We define the source color as a point P_{src} outside the gamut. To map it into the gamut, we cast a ray $R(t)$ towards an anchor point P_{anchor} located on the neutral axis (where $a = 0, b = 0$). Crucially, to preserve the original image detail, we define P_{anchor} to have the exact same Lightness (L) as the source:

$$P_{anchor} = (L_{src}, 0, 0).$$

This constraint ensures the mapping is purely chromatic compression. The parametric line equation is:

$$R(t) = P_{src} + t(P_{anchor} - P_{src})$$

where t is a scalar parameter.

At $t = 0$, we are at the source. At $t = 1$, we are at the anchor.

To find where this ray hits the gamut boundary, we employ the Möller-Trumbore intersection algorithm [4]. This algorithm solves for the intersection of the ray with a triangle without explicitly computing the plane equation coefficients, using a system of linear equations derived from Cramer's Rule.

The intersection occurs where the ray point equals a point on the triangle defined by barycentric coordinates (u, v):

$$P_{src} + t\vec{D} = (1 - u - v)V_0 + uV_1 + vV_2$$

Rearranging this gives a linear system Ax = b, which allows us to solve for t. The explicit solution for the distance parameter t is:

$$t = \frac{\vec{n} \cdot (V_0 - P_{src})}{\vec{n} \cdot (P_{anchor} - P_{src})}$$

If $0 \leq t \leq 1$, the intersection point lies on the infinite plane defined by the triangle. However, the gamut is composed of finite triangles. To validate that the intersection point lies strictly within the bounded triangular face, we must verify the Barycentric Conditions:

$$u \geq 0, v \geq 0, u + v \leq 1$$

By iterating this test over the gamut mesh, the algorithm identifies the exact geometric boundary along the hue vector, providing the mapped coordinate.

III. Implementation and Methodology

A. Computational Environment and Data Structure

The proposed algorithm was implemented using the Python programming language, using the NumPy library for high-performance vector algebra and SciPy for computational geometry tasks. Color space conversions were handled via the Colour-Science library to ensure accurate matrix transformations between sRGB (IEC 61966-2-1) [1] and the Display P3 source space.

B. Geometric Construction of the Gamut Polytope

To perform geometric intersection, the volume of the sRGB gamut must be split into a computable mesh.

1. Grid Generation: We generated a point cloud representing the surface of the sRGB cube by subdividing its six planar faces (R = 0, R = 1, G = 0, ...) into a grid of normalized coordinates.

```python
def generate_srgb_hull(resolution=15):
    # Create evenly spaced values from 0 to 1
    values = np.linspace(0, 1, resolution)

    # Generate rgb points on the 6 faces of the RGB cube
    rgb_points = []

    # Face 1 & 2: Fix Red Value (0 and 1)
    for g in values:
        for b in values:
            rgb_points.append([0.0, g, b])
            rgb_points.append([1.0, g, b])

    # Face 3 & 4: Fix Green Value (0 and 1)
    for r in values:
        for b in values:
            rgb_points.append([r, 0.0, b])
            rgb_points.append([r, 1.0, b])

    # Face 5 & 6: Fix Blue Value (0 and 1)
    for r in values:
        for g in values:
            rgb_points.append([r, g, 0.0])
            rgb_points.append([r, g, 1.0])

    rgb_array = np.array(rgb_points)
```

Fig. 1. Python Code Part of Grid Generation

2. Basis Transformation: These surface points were linearized (gamma decoding) and transformed from the RGB basis to the OKLab vector space [2].

```python
# Convert points from sRGB -> XYZ -> OKLab
# sRGB_to_XYZ function handles gamma decoding
xyz_array = colour.sRGB_to_XYZ(rgb_array)
oklab_array = colour.XYZ_to_Oklab(xyz_array)
```

Fig. 2. Python Code Part of Basis Transformation

3. Convex Hull Construction: The QuickHull algorithm [3] was applied to this point cloud to generate a Triangular Mesh. This results in a set of vertices V and a connectivity list of triangle indices, defining the sRGB gamut as a closed Convex Polytope.

```python
# Compute the 3D Convex Hull
hull = ConvexHull(oklab_array)
```

Fig. 3. Python Code Part of Convex Hull Construction

C. The Intersection Algorithm

The core mapping logic is formulated as a ray-casting problem. For an out of gamut source vector

$$\mathrm{v_{src}} = [\mathrm{L}, \mathrm{a}, \mathrm{b}],$$

we define a radial projection ray R(t) directed toward an anchor $\mathrm{v_{anchor}}$. To strictly preserve Lightness and Hue, the anchor is defined on the neutral axis at the same lightness level:

$$\mathrm{v_{anchor}} = [\mathrm{L}, 0, 0]^{\mathrm{T}}$$

The ray is defined parametrically as:

$$R(t) = v_{src} + t(v_{anchor} - v_{src})$$

```
# Anchor is neutral gray with same lightness
anchor = np.array([source_L, 0.0, 0.0])
ray_direction = anchor - source_oklab
```

```
if closest_t is not None:
    mapped_color = source_oklab + closest_t * ray_direction
    return mapped_color
```

Fig. 4. Python Part of Calculating the Resulting Ray

We iterate through the triangular faces of the gamut mesh. For each face defined by vertices $(V0, V1, V2)$, we calculate the intersection parameter t using the Möller–Trumbore algorithm [4]. This method solves the linear system required to find the barycentric coordinates (u, v) and the ray distance t.

```
# Calculate determinant to check if ray is parallel to triangle
h_vector = np.cross(ray_direction, edge2)
determinant = np.dot(edge1, h_vector)
```

```
 # Precompute inverse determinant
 inverse_determinant = 1.0 / determinant
```

Fig. 5. Python Code Part of setting up for solving the linear system $Ax = b$

```
# Calculate vector from vertex 0 to ray origin
s_vector = ray_origin - vertex0

# Calculate barycentric coordinate u
u = inverse_determinant * np.dot(s_vector, h_vector)
 # Calculate barycentric coordinate v
 q_vector = np.cross(s_vector, edge1)
 v = inverse_determinant * np.dot(ray_direction, q_vector)
```

```
# Calculate t (distance along ray)
t = inverse_determinant * np.dot(edge2, q_vector)
```

Fig. 6. Python Code Part of Using Cramer's Rule to solve the linear system $Ax = b$

A valid intersection occurs when:

$$0 \leq u, v \leq 1 \text{ and } u + v \leq 1 \text{ and } 0 \leq t \leq 1$$

The algorithm finds the first point where our line hits the boundary of the sRGB gamut, while guaranteeing that we stayed on a straight path that didn't change the color (Hue).

```
# Check if intersection lies outside triangle (u must be between 0 and 1)
if u < 0.0 or u > 1.0:
    return None
```

```
# Check if intersection lies outside triangle (v < 0 or u+v > 1)
if v < 0.0 or u + v > 1.0:
    return None

# Return t only if intersection is forward (t > e)
if t > e:
    return t
```

```
if t is not None:
    # We want the smallest positive t (closest intersection)
    # t must be <= 1.0 to be within the segment to the anchor
    if t <= 1.0:
        if closest_t is None or t < closest_t:
            closest_t = t
```

Fig. 7. Python Code Part of Validity Checking

D. Comparative Analysis Strategy

To evaluate the efficacy of the geometric approach, we compared it against the industry-standard Naive Clipping method (where RGB components > 1.0 are truncated). We utilized a test set consisting of the six primary and secondary vectors of the Display P3 gamut (Red, Green, Blue, Cyan, Magenta, Yellow). Evaluation metrics included:

- Δh(Hue Shift): Angular deviation in degrees.
- ΔL (Lightness Error): Scalar difference in luminance.
- ΔE_{00}(CIEDE2000): Total perceptual color difference.

A quick mention to author's hardware is needed as it only supports sRGB and thus can't reliably show whether the color produced by clipping is significantly different than either the source color or the color mapped geometrically. This became the reason behind the author's inability to give results visualized with color patches or gradients, even though it is more intuitive.

IV. Results and Discussion

A. Visual Verification of Geometric Trajectories

To verify the geometric behavior of the algorithm, we visualized the mapping trajectories in two-dimensional subspaces.

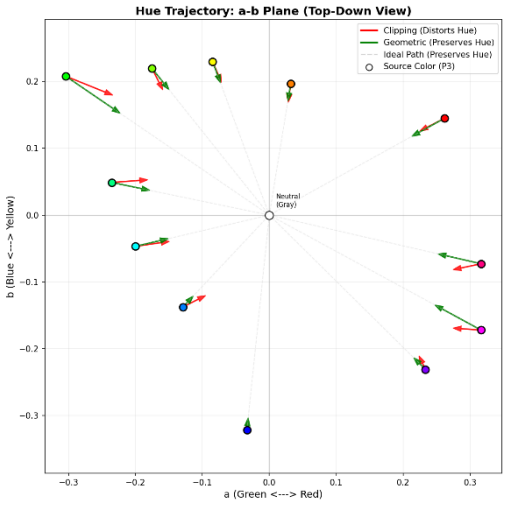

Fig. 8. 2D Vector field radial projection on the a-b chromaticity plane

Green arrows (Geometric) demonstrate radial projection toward the origin, strictly preserving the vector angle (Hue). Red arrows (Clipping) exhibit shifts, particularly evident in the Green and Magenta regions.

As shown in Fig. 1, the geometric method produces trajectories that align perfectly with the hue angle. In contrast, Naive Clipping produces vector that rotate the source vectors, altering their color identity.

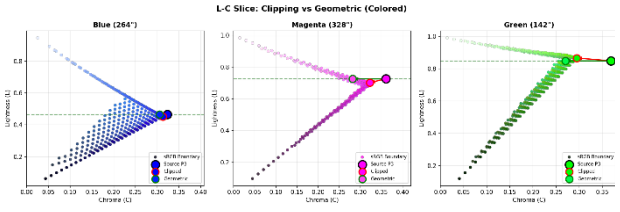

Fig. 9. L-C Cross-Section at Hue 264°(Blue).

Caption: Fig. 2. L-C Cross-Section at Hue 264°(Blue).

The Geometric method (Green) projects horizontally, maintaining constant Lightness (L). Naive Clipping (Red) projects diagonally downward, resulting in significant luminance loss.

Fig. 2 confirms the handling of the Lightness dimension. The geometric intersection slides the coordinate along the constant L plane, whereas clipping forces the color downward, darkening the image to find a valid solution.

B. Quantitative Error Analysis

The numerical performance of the algorithm was evaluated against the Display P3 primaries. The results are summarized in Table I.

GAMUT MAPPING METRICS COMPARISON (P3 -> sRGB)

Color	Method	dE00	dH (deg)	dL	dC
Red	Clip	6.8225	0.2704	0.0206	0.0417
	Geom	9.3764	0.0000	0.0000	0.0561
Green	Clip	5.3253	3.1459	-0.0176	0.0737
	Geom	7.8133	0.0000	0.0000	0.0985
Blue	Clip	1.3670	0.0144	0.0143	0.0100
	Geom	1.8734	0.0000	0.0000	0.0182
Cyan	Clip	4.7756	1.6372	-0.0125	0.0511
	Geom	4.5037	0.0000	0.0000	0.0508
Magenta	Clip	2.9129	3.0959	0.0233	0.0376
	Geom	4.6458	0.0000	0.0000	0.0785
Yellow	Clip	5.3746	0.4594	-0.0032	0.0340
	Geom	5.5215	0.0000	0.0000	0.0341

KEY: dE00 = Perceptual | dH = Hue Shift | dL = Lightness Loss | dC = Chroma Loss

Discussion of Metrics:

1. Hue Preservation (Δh): The primary objective of this study was achieved. The Geometric method yielded a hue shift of 0° across all test vectors. In contrast, Naive Clipping introduced significant angular errors, most notably in Green (3.15°) and Magenta (3.10°). A shift of 3° is visually perceptible, often causing green foliage to appear yellowish or blue skies to shift toward violet.
2. Lightness Preservation (ΔL): The Geometric method preserves the luminance($\Delta L = 0$). Clipping introduced luminance errors up to 0.0233 (Magenta), confirming the "darkening" effect observed in visual tests.
3. The Chroma Trade-off (ΔC vs ΔE_{00}): It is observed that the Geometric method often results in a higher Total Perceptual Error (ΔE_{00}) than clipping (e.g., Red: 9.37 vs 6.82). This is a mathematical necessity of constrained optimization. Because the Geometric method is forbidden from reducing Lightness or rotating Hue to find a "closer" point, it must compress the Chroma (ΔC) significantly more than clipping (e.g., Red ΔC : 0.0561 vs 0.0417).

C. Discussion

The results highlight a fundamental geometric trade-off in signal processing. Naive Clipping minimizes the Euclidean distance to the gamut boundary (lower ΔE) but ignores the Vector Direction (Hue). While this retains more saturation, it changes the color identity.

The proposed Line-Plane Intersection method prioritizes Vector Direction over distance. By treating the gamut mapping problem as a geometric intersection along a fixed ray, we ensure that the resulting color inherit the color identity of the source, differing only in magnitude (Chroma) but identical in orientation (Hue) and elevation (Lightness). While this results in a perceptually "desaturated" or "pastel" output for high-luminance inputs (due to the convex shape of the sRGB polytope at high L), it is a mathematically correct method for preserving the artistic intent of the source signal.

V. Conclusion

This study successfully demonstrated the application of computational geometry to solve the gamut mismatch problem between Display P3 and sRGB. By modeling the destination gamut as a Convex Polytope within the perceptually uniform OKLab vector space, we replaced traditional component-wise clipping with a rigorous Line-Plane Intersection algorithm.

The results validate that treating color as a geometric vector allows for precise control over perceptual attributes. Specifically:

1. Geometric Accuracy: The implementation of the Möller–Trumbore algorithm achieved a hue shift (Δh) of 0° across all primary test vectors. This confirms that the radial projection strategy strictly preserves the vector direction, eliminating the hue

shifting effects (Abney effect) observed in naive clipping. Unlike clipping, which alters the vector ratios in the perceptually non-uniform RGB space, our method respects the hue linearity of OKLab.

2. Luminance Preservation: By constraining the intersection ray to the constant-lightness plane, the algorithm achieved zero luminance error ($\Delta L = 0$), contrasting with the darkening effects of standard RGB clipping.
3. The Fidelity Trade-off: As noted in the survey by Morovic and Luo [5], gamut mapping algorithms must balance the preservation of colorimetric accuracy against the preservation of cognitive texture. Our results show that while geometric intersection yields a higher total error (ΔE_{00}) due to significant chroma compression, it preserves the identity of the color.

We conclude that while naive clipping offers a computationally cheap approximation, the Line-Plane Intersection method provides the mathematical rigor required for high-fidelity color reproduction, transforming the gamut mapping problem from an arbitrary signal truncation into a solved geometric projection.

VI. Appendix

Github repository: https://github.com/StevenT-1/Makalah-Aljabar-Linier-dan-Geometri-Gamut-Mapping.git

VII. Acknowledgment

The author is deeply thankful to God Almighty for providing strength, determination, and opportunity to bring this paper to completion. The author also expresses appreciation to Ir. Rila Mandala, M.Eng., Ph.D., the lecturer of the IF2123 Linear Algebra and Geometry course, for his dedicated guidance and support.

References

[1] IEC, "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB," IEC 61966-2-1:1999, Oct. 1999. [Online]. Available: https://webstore.iec.ch/publication/10912. [Accessed: Dec. 24, 2025].

[2] B. Ottosson, "A perceptual color space for image processing," bottosson.github.io, Dec. 23, 2020. [Online]. Available: https://bottosson.github.io/posts/oklab/. [Accessed: Dec. 20, 2025].

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," ACM Transactions on Mathematical Software, vol. 22, no. 4, pp. 469–483, Dec. 1996. [Online]. Available: https://dl.acm.org/doi/10.1145/235815.235821. [Accessed: Dec. 20, 2025].

[4] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," Journal of Graphics Tools, vol. 2, no. 1, pp. 21–28, 1997. [Accessed: Dec. 20, 2025].

[5] J. Morovic and M. R. Luo, "The fundamentals of gamut mapping: A survey," Journal of Imaging Science and Technology, vol. 45, no. 3, pp. 283–290, May 2001. [Accessed: Dec. 20, 2025].

[6] C. Lilley and L. Verou, "CSS Color Module Level 4," W3C Candidate Recommendation Snapshot, Jul. 05, 2022. [Online]. Available: https://www.w3.org/TR/css-color-4/. [Accessed: Dec. 24, 2025].

[7] I. Muse, "Gamut Mapping," *ColorAide Documentation*, 2024. [Online]. Available: https://facelessuser.github.io/coloraide/gamut/#gamut-mapping-in-any-perceptual-space. [Accessed: Dec. 24, 2025].

[8] R. Munir, "Algeo 11: Vektor di Ruang Euclidean (Bagian 1)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-11-Vektor-di-Ruang-Euclidean-Bag1-2025.pdf. [Accessed: Dec. 24, 2025].

[9] R. Munir, "Algeo 12: Vektor di Ruang Euclidean (Bagian 2)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-12-Vektor-di-Ruang-Euclidean-Bag2-2025.pdf. [Accessed: Dec. 24, 2025].

[10] R. Munir, "Algeo 13: Vektor di Ruang Euclidean (Bagian 3)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-13-Vektor-di-Ruang-Euclidean-Bag3-2025.pdf. [Accessed: Dec. 24, 2025].

[11] R. Munir, "Algeo 15: Ruang vektor umum (Bagian 1)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-15-Ruang-vektor-umum-Bagian1-2025.pdf. [Accessed: Dec. 24, 2025].

[12] R. Munir, "Algeo 16: Ruang vektor umum (Bagian 2)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-16-Ruang-vektor-umum-Bagian2-2025.pdf. [Accessed: Dec. 24, 2025].

[13] R. Munir, "Algeo 17: Ruang vektor umum (Bagian 3)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-17-Ruang-vektor-umum-Bagian3-2025.pdf. [Accessed: Dec. 24, 2025].

[14] R. Munir, "Algeo 18: Ruang vektor umum (Bagian 4)," Aljabar Linier dan Geometri Course Notes, Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-18-Ruang-vektor-umum-Bagian4-2025.pdf. [Accessed: Dec. 24, 2025].

PERNYATAAN

I declare that this paper is my own original work, not an adaptation or translation of another person's paper, and is not plagiarism.

Bandung, 24 Desember 2025

Steven Tan, 13524060