

Linear Algebra Behind the Lens: Camera Projection and Color Grading in Digital Images

Philipp Hamara - 13524101

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jalan Ganesha 10 Bandung

E-mail: philipphamara420@gmail.com, 13524101@std.stei.itb.ac.id

Abstract—Computer vision is a field that studies how computers perceive and interpret visual information from the real world. A fundamental aspect of the field is the projection of our three-dimensional world onto two-dimensional images through mathematical camera models. Another important aspect is color grading, where the appearance and mood of an image are adjusted by transforming its color representation. Both camera projection and color grading rely heavily on linear algebra. Camera models use vectors, matrices, and homogeneous coordinates to map 3D points in the world to 2D points on an image plane. Similarly, color grading often applies 3×3 matrices to RGB color vectors to modify or correct colors in a controlled, linear way. This paper explores how these two processes can be formulated and understood using linear algebra theory, and illustrates their implementation through simple numerical examples and code-based experiments.

Keywords—Computer Vision, Camera Projection, Color Grading, Digital Images

I. INTRODUCTION

Modern computers are increasingly used to capture, generate, and process visual data, from digital photography and cinema to video games and augmented reality. A key question in this context is how a three-dimensional world can be represented as two-dimensional images that still preserve geometric structure and visual meaning. This is the domain of camera modeling and projection, which describes how points in 3D space are mapped onto an image plane. Once an image has been formed, post-processing operations such as color grading play an equally important role, shaping the mood, contrast, and overall appearance of the image by adjusting its color characteristics.

Both camera projection and color grading can be formulated in a unified way using linear algebra. In camera modeling, points in three-dimensional space are represented as vectors and mapped to the image plane by multiplication with a projection matrix, typically expressed in homogeneous coordinates. This approach allows a sequence of operations: such as translation, rotation, and perspective projection to be combined compactly using matrix multiplication. On the color side, each pixel is treated as a vector in a three-dimensional RGB color space, and many color transformations, from technical color correction to creative looks,

can be described as linear transformations using 3×3 matrices applied to these color vectors.

This paper explores how fundamental concepts from linear algebra: vector spaces, linear transformations, matrices, and homogeneous coordinates underpin two central processes in digital imaging: camera projection and color grading. The main goal is to present clear mathematical formulations of these processes and to demonstrate their practical implementation through simple numerical examples and code-based experiments. In doing so, the paper aims to make the connection explicit between abstract linear algebra theory and concrete applications in computer vision and digital image processing.

More broadly, studying camera projection and color grading together highlights how a single mathematical framework can unify diverse tasks in computer vision and graphics. The same ideas that describe how a 3D scene is “seen” by a camera also describe how the colors of that scene are adjusted to achieve a particular visual style. Seeing these connections helps bridge the gap between abstract linear algebra learned in the classroom and the practical systems used in modern visual media, and shows why a solid understanding of vectors, matrices, and linear transformations is essential for anyone interested in imaging, graphics, or computer vision.

II. THEORETICAL BASIS

A. Vectors

A vector in linear algebra is an object that can be described by an ordered list of numbers (its components) and is often interpreted as having both length and direction. Vectors may live in spaces of different dimensions. For instance, in the plane a vector $\mathbf{v}$ can be written as

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix},$$

while in three-dimensional space a vector $\mathbf{v}$ is typically written as

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

Some basic operations and properties of vectors are:

- 1) **Addition:** Two vectors of the same dimension can be added by adding their components one by one. If

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix},$$

then

$$\mathbf{u} \pm \mathbf{v} = \begin{bmatrix} u_1 \pm v_1 \\ u_2 \pm v_2 \\ \vdots \\ u_n \pm v_n \end{bmatrix}.$$

Vector addition is both commutative and associative.

- 2) **Scalar Multiplication:** A vector can be multiplied by a real number (a scalar) by multiplying each component by that scalar. For a scalar c and a vector

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix},$$

the scaled vector is

$$c\mathbf{v} = \begin{bmatrix} cv_1 \\ cv_2 \\ \vdots \\ cv_n \end{bmatrix}.$$

Scalar multiplication satisfies familiar distributive and associative rules, such as

$$c(\mathbf{u} + \mathbf{v}) = c\mathbf{u} + c\mathbf{v}, \quad (c+d)\mathbf{u} = c\mathbf{u} + d\mathbf{u}, \quad (cd)\mathbf{u} = c(d\mathbf{u}).$$

- 3) **Magnitude (Norm):** The length of a vector $\mathbf{v}$, often called its norm, is defined by

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}.$$

This generalizes the usual Euclidean length formula from two and three dimensions to $\mathbb{R}^n$.

- 4) **Direction:** The direction of a nonzero vector describes how it is oriented in space. In many applications in geometry and physics, vectors are used specifically to encode directions of motion, forces, or viewing rays, where the magnitude represents intensity and the direction represents orientation.

B. Vector Spaces

A vector space (or linear space) over the real numbers is a set V whose elements are called vectors, together with two operations: vector addition and scalar multiplication, that satisfy certain axioms. The scalars are real numbers, and the vectors can be added to each other and multiplied by real numbers in a way that is compatible with these axioms.

Formally, a set V is a vector space over $\mathbb{R}$ if:

- For any $\mathbf{u}, \mathbf{v} \in V$, the sum $\mathbf{u} + \mathbf{v}$ is in V (closed under addition).

- For any $c \in \mathbb{R}$ and $\mathbf{v} \in V$, the product $c\mathbf{v}$ is in V (closed under scalar multiplication).

In addition, the following properties (axioms) must hold for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and all scalars $c, d \in \mathbb{R}$:

- 1) **Associativity of addition:**

$$(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w}).$$

- 2) **Commutativity of addition:**

$$\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}.$$

- 3) **Additive identity:** There exists a zero vector $\mathbf{0} \in V$ such that

$$\mathbf{v} + \mathbf{0} = \mathbf{v} \quad \text{for all } \mathbf{v} \in V.$$

- 4) **Additive inverse:** For every $\mathbf{v} \in V$ there exists a vector $-\mathbf{v} \in V$ such that

$$\mathbf{v} + (-\mathbf{v}) = \mathbf{0}.$$

- 5) **Compatibility of scalar multiplication with field multiplication:**

$$c(d\mathbf{v}) = (cd)\mathbf{v}.$$

- 6) **Identity element of scalar multiplication:**

$$1 \cdot \mathbf{v} = \mathbf{v} \quad \text{for all } \mathbf{v} \in V.$$

- 7) **Distributivity of scalar multiplication over vector addition:**

$$c(\mathbf{u} + \mathbf{v}) = c\mathbf{u} + c\mathbf{v}.$$

- 8) **Distributivity of scalar multiplication over scalar addition:**

$$(c + d)\mathbf{v} = c\mathbf{v} + d\mathbf{v}.$$

C. Matrices

A matrix is a rectangular table of numbers arranged in rows and columns. An $m \times n$ matrix has m rows and n columns. In linear algebra, matrices are used to describe linear transformations and can model operations such as rotations, scalings, and shears in spaces of various dimensions.

Some basic operations and properties of matrices are:

- 1) **Addition:** Two matrices of the same size can be added by adding their entries component-wise. If

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1n} \\ B_{21} & B_{22} & \cdots & B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n1} & B_{n2} & \cdots & B_{nn} \end{bmatrix},$$

then their sum is

$$A + B = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & \cdots & A_{1n} + B_{1n} \\ A_{21} + B_{21} & A_{22} + B_{22} & \cdots & A_{2n} + B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} + B_{n1} & A_{n2} + B_{n2} & \cdots & A_{nn} + B_{nn} \end{bmatrix}.$$

- 2) **Scalar Multiplication:** Multiplying a matrix by a scalar c means multiplying every entry of the matrix by c . For a scalar c and a matrix A ,

$$cA = c \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix},$$

so that

$$cA = \begin{bmatrix} cA_{11} & cA_{12} & \dots & cA_{1n} \\ cA_{21} & cA_{22} & \dots & cA_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ cA_{n1} & cA_{n2} & \dots & cA_{nn} \end{bmatrix}.$$

- 3) **Matrix Multiplication:** If A is an $m \times p$ matrix and B is a $p \times n$ matrix, then the product $C = AB$ is an $m \times n$ matrix. The entry in row i and column j of C is defined by

$$C_{ij} = \sum_{k=1}^p A_{ik} B_{kj}.$$

This operation corresponds to composing the linear transformations represented by A and B .

- 4) **Transpose:** The transpose of a matrix A is obtained by reflecting it across its main diagonal, so that rows become columns and columns become rows. The transpose is denoted by A^T . If

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix},$$

then

$$A^T = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix}.$$

D. Camera Projection

Camera projection describes how points in three-dimensional space are mapped onto a two-dimensional image plane using a camera model such as the pinhole camera. In the pinhole camera model, all rays of light from the scene pass through a single point (the camera center) before intersecting the image plane, which produces a perspective projection in which objects farther from the camera appear smaller. This model can be expressed in a compact way using homogeneous coordinates and matrix-vector multiplication, and is the standard formulation in modern computer vision.

In a linear algebra framework, a 3D point in world coordinates is written in homogeneous form as

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

and the camera is represented by a 3×4 projection matrix P that combines intrinsic parameters (focal length, principal point, pixel aspect ratio) and extrinsic parameters (rotation and translation describing the camera pose). The corresponding image point, also in homogeneous coordinates, is then given by

$$\mathbf{x} \sim P\mathbf{X},$$

where $\sim$ denotes equality up to a nonzero scalar factor, reflecting that homogeneous coordinates represent points only up to scaling.

Writing

$$\mathbf{x} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix} = P\mathbf{X},$$

the inhomogeneous image coordinates are obtained by normalizing with respect to the third component,

$$x = \frac{\tilde{x}}{\tilde{w}}, \quad y = \frac{\tilde{y}}{\tilde{w}},$$

so that the final homogeneous image point is

$$\hat{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

This division by $\tilde{w}$ implements the perspective effect of the pinhole model, since $\tilde{w}$ depends on the depth of the 3D point relative to the camera.

The 3×4 projection matrix is often factorized as

$$P = K[R | \mathbf{t}],$$

where K is the 3×3 intrinsic calibration matrix, R is a 3×3 rotation matrix, and $\mathbf{t}$ is a 3×1 translation vector. This factorization makes explicit that camera projection under the pinhole model is a linear transformation in homogeneous coordinates, allowing rotations, translations, and perspective effects to be combined into a single matrix multiplication followed by a homogeneous normalization step.

E. Color Grading

Color grading adjusts the visual appearance of an image by modifying its colors to achieve a desired look or correct color imbalances. In terms of linear algebra, each pixel color can be represented as a vector in the RGB color space,

$$\mathbf{c}_{\text{in}} = \begin{bmatrix} R \\ G \\ B \end{bmatrix},$$

where R , G , and B denote the intensities of the red, green, and blue channels, respectively. The RGB color space can be viewed as a three-dimensional vector space, so any linear color operation on a pixel can be modeled as a linear transformation on this three-dimensional vector.

A global, linear color transform can therefore be modeled by a 3×3 matrix M that maps one RGB vector to another:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}, \quad \mathbf{c}_{\text{out}} = M\mathbf{c}_{\text{in}}.$$

Written component-wise, this gives

$$\begin{bmatrix} R_{\text{out}} \\ G_{\text{out}} \\ B_{\text{out}} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix},$$

so each output channel is a linear combination of the three input channels:

$$\begin{aligned} R_{\text{out}} &= m_{11}R + m_{12}G + m_{13}B, \\ G_{\text{out}} &= m_{21}R + m_{22}G + m_{23}B, \\ B_{\text{out}} &= m_{31}R + m_{32}G + m_{33}B. \end{aligned}$$

There are three input components (R, G, B) and three output components, and each output component is allowed to mix all three inputs with different weights. By choosing the entries of M appropriately, one can implement operations such as channel mixing, white balance adjustments, approximate luminance-based black-and-white conversion, or simple creative looks (for example, cool, warm, or green-tinted tones). Applying the same matrix M to every pixel implements a consistent, linear color grading operation across the entire image.

III. IMPLEMENTATION

This section presents the implementation of two small Python programs that demonstrate how linear algebra is used in camera projection and color grading. Both implementations use NumPy for matrix and vector operations, with Matplotlib for visualization and OpenCV for image I/O.

A. Camera Projection

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d.art3d import Poly3DCollection
4
5 # Pyramid geometry (world space)
6
7 base_size = 2.0
8 half = base_size / 2.0
9 apex_height = 2.0
10
11 V0 = np.array([-half, -half, 0.0])
12 V1 = np.array([ half, -half, 0.0])
13 V2 = np.array([ half,  half, 0.0])
14 V3 = np.array([-half,  half, 0.0])
15 V4 = np.array([ 0.0,  0.0, apex_height])
16
17 vertices = np.stack([V0, V1, V2, V3, V4], axis=0) # (5, 3)
18
19 faces = [
20     [0, 1, 2, 3], # base
21     [0, 1, 4],   # side 1
22     [1, 2, 4],   # side 2
23     [2, 3, 4],   # side 3
24     [3, 0, 4],   # side 4
25 ]
26
27 face_colors = [
28     "#aaaaaa", # base
29     "#ff5555", # side 1
30     "#55ff55", # side 2
31     "#5555ff", # side 3
32     "#ffaa00", # side 4
33 ]
34
35 pyr_world_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])

```

Fig. 1. Python code defining the pyramid geometry in world coordinates, including vertices, faces, per-face colors, and homogeneous coordinates.

In the implementation stage, a colored pyramid is used as a simple 3D object to demonstrate camera projection. The pyramid is defined by five vertices in world coordinates: four form a square base on the plane $z=0$, and one is an apex elevated along the z -axis. These vertices are grouped into one quadrilateral face for the base and four triangular faces for the sides, and each face is assigned a distinct color so that it remains visually distinguishable after projection. The vertex array is then extended to homogeneous coordinates $[x, y, z, 1]$, which allows all subsequent transformations (view and projection) to be expressed as matrix–vector multiplications in homogeneous space, consistent with standard projective camera models in computer vision.

```

1 # Camera (view) and projection
2
3 T = np.array([
4     [1, 0, 0, 0],
5     [0, 1, 0, -5],
6     [0, 0, 1, -8],
7     [0, 0, 0, 1],
8 ], dtype=float)
9
10 fovy = np.deg2rad(60.0)
11 aspect = 1.0
12 near, far = 1.0, 50.0
13 f = 1.0 / np.tan(fovy / 2.0)
14
15 P_persp = np.array([
16     [f / aspect, 0, 0, 0],
17     [0, f, 0, 0],
18     [0, 0, (far+near)/(near-far), (2*far*near)/(near-far)],
19     [0, 0, -1, 0],
20 ], dtype=float)
21
22 def project_points_homogeneous(points_h, T_view, P):
23     pts_cam = (T_view @ points_h.T).T
24     pts_clip = (P @ pts_cam.T).T
25     w = pts_clip[:, 3:4]
26     pts_ndc = pts_clip[:, :3] / w
27     x_img = pts_ndc[:, 0]
28     y_img = pts_ndc[:, 1]
29     return pts_cam[:, :3], np.stack([x_img, y_img], axis=1)
30
31 pts_cam3d, pts_img2d = project_points_homogeneous(pyr_world_h, T, P_persp)

```

Fig. 2. Python code for the camera view matrix, perspective projection matrix, and the projection function from homogeneous world coordinates to 2D image coordinates.

The camera model is represented by a 4×4 view matrix and a 4×4 perspective projection matrix, as shown in Figure 2. The view matrix translates the world by $(0, -5, -8)$, which corresponds to placing the camera at approximately $(0, 5, 8)$ in world coordinates and orienting it toward the origin, so that the pyramid lies inside the camera's field of view. The projection matrix is parameterized by a vertical field of view of 60° , an aspect ratio of 1, and near–far clipping planes at 1 and 50 units, yielding the standard pinhole camera matrix that maps camera-space coordinates into clip space. The projection function then applies the view and projection matrices to the homogeneous world vertices, performs the homogeneous normalization by dividing by the fourth component w , and returns both the camera-space coordinates and the 2D image-plane coordinates $(x_{\text{img}}, y_{\text{img}})$.


```

1 # Visualization
2
3 fig = plt.figure(figsize=(10, 5))
4
5 ax3d = fig.add_subplot(1, 2, 1, projection='3d')
6
7 poly3d = []
8 for face in faces:
9     poly3d.append([vertices[i] for i in face])
10
11 mesh = Poly3DCollection(poly3d, facecolors=face_colors, edgecolors='k', alpha=0.9)
12 ax3d.add_collection3d(mesh)
13
14 # Camera position and viewing direction (toward origin)
15 cam_pos = np.array([0.0, 5.0, 8.0])
16 look_at = np.array([0.0, 0.0, 0.0])
17 cam_dir = look_at - cam_pos # direction vector
18
19 ax3d.scatter(*cam_pos, color='black', s=40)
20 ax3d.text(*cam_pos, "Camera", color='black')
21
22 # Arrow showing where the camera is looking
23 ax3d.quiver(
24     cam_pos[0], cam_pos[1], cam_pos[2],
25     cam_dir[0], cam_dir[1], cam_dir[2],
26     length=3.0, normalize=True, color='black'
27 )
28
29 ax3d.set_title("Colored pyramid in 3D (with camera direction)")
30 ax3d.set_xlabel("X")
31 ax3d.set_ylabel("Y")
32 ax3d.set_zlabel("Z")
33 ax3d.set_xlim(-3, 3)
34 ax3d.set_ylim(-1, 7)
35 ax3d.set_zlim(-1, 5)
36 ax3d.view_init(elev=20, azim=-60)
37 ax3d.set_box_aspect([1,1,1])
38
39 # 2D projection
40 ax2d = fig.add_subplot(1, 2, 2)
41 for face, color in zip(faces, face_colors):
42     pts = pts_img2d[face]
43     xs = pts[:, 0]
44     ys = pts[:, 1]
45     ax2d.fill(xs, ys, facecolor=color, edgecolor='k', alpha=0.9)
46
47 for i, (x, y) in enumerate(pts_img2d):
48     ax2d.text(x, y, str(i), fontsize=8, ha='center', va='center')
49
50 ax2d.set_title("Perspective projection on image plane (colored faces)")
51 ax2d.set_xlabel("x")
52 ax2d.set_ylabel("y")
53 ax2d.axhline(0, color='black', linewidth=0.5)
54 ax2d.axvline(0, color='black', linewidth=0.5)
55 ax2d.set_aspect('equal', adjustable='box')
56 ax2d.grid(True)
57
58 plt.tight_layout()
59 plt.show()

```

Fig. 3. Python code for rendering the colored pyramid and camera in 3D (left subplot) and its perspective projection as colored faces on the 2D image plane (right subplot).

Finally, the visualization code in Figure 3 renders both the 3D configuration and the 2D projection. In the left subplot, a `Poly3DCollection` is constructed from the face index lists and colored using the predefined face colors, producing a colored pyramid in 3D world space. The camera is drawn as a point at (0,5,8) with a text label, and an arrow created with `ax3d.quiver` indicates the viewing direction toward the origin. In the right subplot, the projected 2D points are grouped according to the same face indices and filled as colored polygons using `ax2d.fill`, so that each 3D face maps to a similarly colored region in the image plane; vertex indices are annotated to emphasize the one-to-one correspondence between 3D vertices and their 2D projections, providing a concrete visual illustration of the perspective camera model.

B. Color Grading

Figure 4 shows the Python code used to load the input image and convert it to an RGB representation with floating-point intensities in the range [0,1]. The image is first read from disk using `cv2.imread`, then converted from BGR (OpenCV's default) to RGB, and finally normalized to ensure numerically stable linear operations.

```

1 import numpy as np
2 import cv2
3
4 # Load image
5 img_bgr = cv2.imread("input.jpg").astype(np.float32) / 255.0
6 img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
7
8 h, w, c = img_rgb.shape

```

Fig. 4. Python code for loading and normalizing the input RGB image.

```

1 # Define 3x3 color matrices
2 M_cool = np.array([
3     [1.05, -0.05, 0.00],
4     [-0.05, 1.10, -0.05],
5     [0.00, -0.10, 1.20],
6 ], dtype=np.float32)
7
8 M_warm = np.array([
9     [1.15, -0.05, -0.05],
10    [0.00, 1.05, -0.05],
11    [-0.05, -0.05, 1.00],
12 ], dtype=np.float32)
13
14 M_contrast = np.array([
15    [1.20, -0.05, -0.05],
16    [-0.05, 1.20, -0.05],
17    [-0.05, -0.05, 1.20],
18 ], dtype=np.float32)
19
20 M_bw = np.array([
21    [0.299, 0.587, 0.114],
22    [0.299, 0.587, 0.114],
23    [0.299, 0.587, 0.114],
24 ], dtype=np.float32)
25
26 M_green = np.array([
27    [0.90, 0.10, 0.00],
28    [0.10, 1.00, 0.10],
29    [0.00, 0.10, 0.90],
30 ], dtype=np.float32)

```

Fig. 5. Definition of the 3×3 color matrices for cool, warm, contrast, grayscale, and green-enhanced effects.

Several 3×3 color matrices are then defined, as illustrated in Figure 5. The matrices `M_cool`, `M_warm`, and `M_contrast` adjust the balance between the red, green, and blue channels in a coupled way: diagonal entries slightly larger than one strengthen the main channel, while small negative off-diagonal terms redistribute part of the intensity

into other channels, thus modifying the global color tone and contrast. Mathematically, each output color $\mathbf{c}' = [R', G', B']^\top$ is obtained from an input color $\mathbf{c} = [R, G, B]^\top$ via

$$\mathbf{c}' = M\mathbf{c},$$

where M is one of the color matrices shown in the figure.

The matrix M_{bw} in Figure 5 implements a weighted grayscale conversion using the coefficients (0.299, 0.587, 0.114) for the R, G, and B components, which are widely used to approximate luminance according to human visual perception. Because all three rows are identical, each output channel receives the same intensity, producing a visually grayscale image. The matrix M_{green} is designed to emphasize the green channel while still incorporating contributions from red and blue.



Fig. 6. Function for applying a 3×3 color matrix to all pixels of the RGB image.

Figure 6 presents the function `apply_color_matrix`, which applies a chosen color matrix to all pixels of the image. The three-dimensional image array of size $(h, w, 3)$ is reshaped into a two-dimensional array of size $(h \times w, 3)$, so that each row corresponds to one RGB color vector. A single batched matrix multiplication with the transpose of M applies the linear color transform to all pixels simultaneously, and the result is clamped to the interval $[0, 1]$ to keep the intensities within a valid range. This design is modular: adding a new color style only requires defining another 3×3 matrix and calling the same function.



Fig. 7. Saving the transformed images after applying the different color matrices.

Finally, the code fragment in Figure 7 converts each processed image back to 8-bit format and saves it to disk using OpenCV. The images are rescaled from $[0, 1]$ to $[0, 255]$, cast to `uint8`, and converted from RGB to BGR to match OpenCV's default channel ordering for file output [web:184][web:193]. Each result is written with a filename that encodes the corresponding color transformation (cool, warm, contrast, grayscale, or green), which facilitates visual comparison of the different matrix-based effects.

IV. RESULTS

A. Camera Projection

In the results, the 3D and 2D visualizations clearly show how the camera projects the colored pyramid from world space onto the image plane. In the 3D view on the left, the five pyramid vertices are located at fixed coordinates in the world frame: four vertices form a square base on the plane $z = 0$ at positions $(\pm \frac{\text{base_size}}{2}, \pm \frac{\text{base_size}}{2}, 0)$, while the apex lies at $(0, 0, \text{apex_height})$. Each face of the pyramid is drawn with a distinct color (a gray base and four differently colored sides), so that it is easy to see how each planar 3D face is oriented relative to the camera. The camera itself is placed at approximately $(0, 5, 8)$ in world coordinates, and an arrow from this point toward the origin indicates the viewing direction along the principal axis, i.e., the line from the camera center to the scene that defines the center of the field of view in the pinhole model.

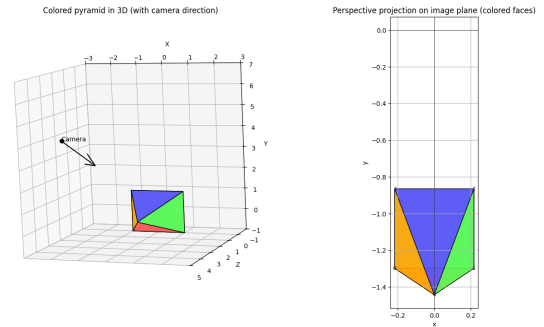


Fig. 8. Visualization of camera projection. Left: colored pyramid in 3D world coordinates with camera position and viewing direction. Right: perspective projection of the same pyramid onto the 2D image plane.

In the 2D view on the right, the same vertices have been transformed by the view and projection matrices and then normalized by their homogeneous component w to obtain image-plane coordinates $(x_{\text{img}}, y_{\text{img}})$. Each 3D face uses the corresponding projected vertices to draw a filled polygon with the same color as in the 3D plot, so the gray base and each of the four colored side faces appear as separate regions in the image. Vertex indices are labeled consistently in both views, making explicit which 3D point (X, Y, Z) maps to which 2D point $(x_{\text{img}}, y_{\text{img}})$. This figure therefore illustrates the complete mapping from world coordinates (left) through the camera model to image-plane coordinates

(right), providing a concrete example of the perspective projection described by the pinhole camera model.

B. Color Grading

The following figures show the original image and the results of applying several different 3×3 color grading matrices.



Fig. 9. Original image (ITB's Plaza Widya)
Source:
Author's archive



Fig. 10. Color grading result: Cool tone
Source:
Author's archive



Fig. 11. Color grading result: Warm tone
Source:
Author's archive



Fig. 12. Color grading result: Black and white
Source:
Author's archive



Fig. 13. Color grading result: Higher contrast
Source:
Author's archive



Fig. 14. Color grading result: Green tone
Source:
Author's archive

V. CONCLUSION

The work in this paper shows that two seemingly different tasks in digital imaging, projecting a 3D scene onto a 2D image plane and manipulating the colors of that image, are both naturally and efficiently described using linear algebra. Camera projection was modeled using homogeneous

coordinates and a projection matrix that maps 3D points to 2D image coordinates via matrix–vector multiplication, illustrating how translation, rotation, and perspective can be combined into a single linear transformation. Color grading was formulated as applying 3×3 matrices to RGB color vectors, demonstrating how global adjustments such as cool and warm looks, increased contrast, grayscale conversion, and tinted styles can all be implemented as linear transformations in a three-dimensional color space.

Through simple Python implementations, these concepts were made concrete. The camera projection experiment projected the corners of a virtual cube using a view and projection matrix, and the results were visualized as 2D point configurations that match the geometric intuition of a pinhole camera model. The color grading experiment applied several different 3×3 matrices to a real image, producing side-by-side comparisons that directly reveal how changes in matrix entries translate into changes in image appearance. These examples highlight that many core operations in computer vision and graphics reduce to carefully chosen matrix operations on vectors representing geometry and color.

Overall, the paper emphasizes that vector spaces, matrices, and linear transformations are not just abstract mathematical objects but practical tools that underpin real imaging pipelines. Understanding how camera matrices and color matrices operate makes it easier to reason about, design, and debug visual systems, from rendering engines to post-production workflows. By connecting course-level linear algebra theory with tangible visual results, this work illustrates why a solid grasp of linear algebra is essential for further study and development in computer vision, computer graphics, and digital content creation.

VI. SUGGESTION

For future work building on this paper, several directions can be explored to deepen both the mathematical and practical understanding of camera projection and color grading as applications of linear algebra:

- **More Realistic Camera Models:** This work uses a simplified pinhole camera and manually defined projection matrix. Future projects could incorporate real camera calibration data, including intrinsic and extrinsic parameters as well as lens distortion, to compare idealized linear models with more realistic imaging pipelines.
- **Extension to Non-Linear Color Pipelines:** The current implementation focuses on linear 3×3 color matrices in RGB space. Further research could investigate how these linear blocks fit into full color pipelines that include non-linear operations such as gamma correction, tone mapping, and film-style curves, and analyze which parts remain purely linear.
- **Comparative Study of Color Spaces:** Another direction is to extend the analysis beyond RGB to other color spaces, such as XYZ, YUV, or Lab, and study how linear transformations behave in each space. This could include implementing and comparing color grading ma-

trices in multiple spaces and evaluating their visual and numerical effects.

- **Integration with Graphics or Vision Frameworks:**

The simple Python implementations presented here could be integrated into a small rendering demo or a basic computer vision project (for example, using OpenGL or a vision library). This would highlight how the same matrix formulations are used in real-time graphics and image processing systems.

By pursuing these directions, students and developers can gain a deeper appreciation of how vector spaces, matrices, and linear transformations are embedded in real imaging pipelines, and how theoretical linear algebra concepts translate into concrete design decisions in computer vision, graphics, and digital content creation.

APPENDIX

The GitHub repository for this paper can be accessed at <https://github.com/philippqiwu/Makalah-IF2123-CameraProjection-ColorGrading>. A video explanation of the contents of this paper is also present at <https://youtu.be/wozE5LjCdWU>

ACKNOWLEDGMENT

The author expresses his heartfelt gratitude to Dr. Ir. Rila Mandala, M.Eng., Ph.D., his lecturer for Linear Algebra and Geometry at Bandung Institute of Technology, for his comprehensive teaching which laid down the groundwork for understanding the concepts applied in this paper.

The author would also like to thank his friends and family for all the support they have shown throughout his journey as a student at ITB.

Lastly, the author would like to show his appreciation to the Hong Kong film director Wong Kar-Wai for his masterful work in films like *Chungking Express* and *Fallen Angels* which inspired and opened the author's eyes to the magical wonders of filmmaking and color grading.

REFERENCES

- [1] Munir, Rinaldi. 2025. "Vektor di Ruang Euclidean (Bagian 1)". *Department of Informatics, Institut Teknologi Bandung*, 2025-2026, [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-11-Vektor-di-Ruang-Euclidean-Bag1-2025.pdf>. [Accessed: Dec. 23, 2025].
- [2] Munir, Rinaldi. 2025. "Ruang Vektor Umum (Bagian 1)". *Department of Informatics, Institut Teknologi Bandung*, 2025-2026, [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-15-Ruang-vektor-umum-Bagian1-2025.pdf>. [Accessed: Dec. 23, 2025].
- [3] Munir, Rinaldi. 2025. "Ruang Vektor Umum (Bagian 4) dan Transformasi Linier". *Department of Informatics, Institut Teknologi Bandung*, 2025-2026, [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-18-Ruang-vektor-umum-Bagian4-2025.pdf>. [Accessed: Dec. 23, 2025].
- [4] Munir, Rinaldi. 2025. "Review Matriks". *Department of Informatics, Institut Teknologi Bandung*, 2025-2026, [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/Algeo-01-Review-Matriks-2025.pdf>. [Accessed: Dec. 23, 2025].

- [5] Fusiello, Andrea. 2003. "Camera Modeling and Calibration." *Vision Book: Imaging Geometry*, [Online]. Available: https://visionbook.mit.edu/imaging_geometry.html. [Accessed: Dec. 23, 2025].
- [6] Kris Kitani. 2017. "11.1 Camera Matrix." 16-385 *Computer Vision*, Carnegie Mellon University, [Online]. Available: https://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf. [Accessed: Dec. 23, 2025].
- [7] Cornell CS. 2013. "Color Science as Linear Algebra." *CS 417 Course Notes*, Cornell University, [Online]. Available: <https://www.cs.cornell.edu/courses/cs417/2003sp/Lectures/Lecture38/38color.pdf>. [Accessed: Dec. 23, 2025].
- [8] Jadhav, A. et al. 2022. "Linear Algebra in Image Processing." *JETIR*, [Online]. Available: <https://www.jetir.org/papers/JETIR2204008.pdf>. [Accessed: Dec. 24, 2025].

STATEMENT

Hereby, I declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not a product of plagiarism.

Bandung, 23 December 2025

A handwritten signature in black ink, consisting of a large, stylized 'P' followed by a series of loops and a long horizontal stroke extending to the right.

Philipp Hamara
13524101