1. 背景知識

1.1. 三維線性變換

在高中數學與大學的線性代數課程中，我們學過二維的線性變換，即旋轉、縮放、鏡射、推移等。我們可以將這些概念延伸到三維空間，並以程式實現，這樣的技術在電腦視覺、電腦圖學及物理模擬等領域都有廣泛的應用。

在三維空間中，一個物體的位置可以透過一個向量 (x, y, z) 來表示，而線性變換則是透過一個 3×3 矩陣來實現的，這個矩陣作用在一個向量上，可以產生一個新的向量，表示物體變換後的位置。以下是幾個常見的三維線性變換矩陣。

(1) 縮放（Scaling）

三維空間的縮放變換可以單獨的沿著 x、y、z 軸改變物體的尺寸。若想在各個軸分別縮放不同倍數，縮放矩陣 $\boldsymbol{S}$ 可表示為：

$$\boldsymbol{S}(k_x, k_y, k_z) = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix},\ k_x \cdot k_y \cdot k_z \neq 0$$

其中 k_x、k_y、k_z 分別是 x、y、z 軸的縮放因子，縮放因子應為非零實數。當縮放因子小於零，會在相應的軸上造成物體的鏡像翻轉效果，可以用於實現特定的圖形變換。

(2) 正射投影（Orthographic Projection）

在三維空間中的正射投影，是指在忽略掉一個維度的情況下，將三維物體的影像繪製到二維平面上。為了執行這種投影，正射投影矩陣將一個方向的縮放因子設為零，從而忽略該方向上的坐標。有投影到 yz 平面（忽略 x 坐標）的矩陣 P_{yz}、投影到 xz 平面（忽略 y 坐標）的矩陣 P_{xz}、投影到 xy 平面（忽略 z 坐標）的矩陣 P_{xy}，如下所示。

$$\boldsymbol{P}_{yz} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P_{xz} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P_{xy} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

(3) 旋轉（Rotation）

三維空間中的旋轉可以繞著三個主軸（ x 軸、y 軸和 z 軸）進行，因此有三種矩陣：繞 x 軸旋轉角度 θ 的矩陣 $\boldsymbol{R}_x(\theta)$、繞 y 軸旋轉角度 θ 的矩陣 $\boldsymbol{R}_y(\theta)$、繞 z 軸旋轉角度 θ 的矩陣 $\boldsymbol{R}_z(\theta)$，如下所示。旋轉方向會遵循「右手法則」: 將右手的大拇指指向旋轉軸的正向，四指的彎曲方向即代表正旋轉的方向。此外，我們可以發現矩陣中標示為紅色的部分，就是我們學過的二維旋轉矩陣。

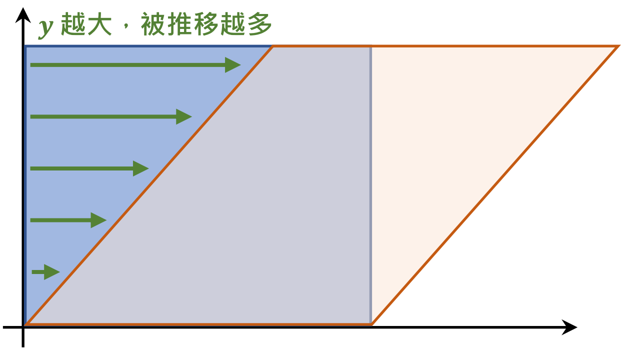
$$\boldsymbol{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\boldsymbol{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(4) 推移（Shearing）

推移是一種幾何變換，它會將一個物體在某方向上拉伸或壓縮，同時在垂直於該方向的方向上保持不變。二維平面上的推移如圖 1 所示，矩形在 x 軸方向被拉伸，同時在 y 軸方向上保持不變。



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0.8 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + 0.8y \\ y \end{bmatrix}$$

圖 1 推移變換 藍色的矩形沿 x 軸方向推移後，成為橘色的平行四邊形

在三維空間中，推移變換可以沿著任何軸向（ x 軸、y 軸和 z 軸）進行，因此有三種矩陣：沿著

x 軸的推移矩陣 $\boldsymbol{H}_x(s,t)$、沿著 y 軸的推移矩陣 $\boldsymbol{H}_y(s,t)$、沿著 z 軸的推移矩陣 $\boldsymbol{H}_z(s,t)$，如下所示。

$$\boldsymbol{H}_x(s,t)=\begin{bmatrix}1 & s & t\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$\boldsymbol{H}_y(s,t)=\begin{bmatrix}1 & 0 & 0\\ s & 1 & t\\ 0 & 0 & 1\end{bmatrix}$$

$$\boldsymbol{H}_z(s,t)=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ s & t & 1\end{bmatrix}$$

(5) 鏡射（Reflection）

在三維空間中，當我們想要確定一點相對於某一平面的鏡像位置時，可以使用鏡射矩陣來進行計算。如果一個平面的法線向量是單位向量（長度為 1），記作 $\boldsymbol{n}=(n_x,n_y,n_z)$，那麼相對於這個平面的鏡射變換矩陣 $\boldsymbol{M}$ 可以表達為：

$$\boldsymbol{M}=\boldsymbol{I}-2\cdot\boldsymbol{n}\cdot\boldsymbol{n}^{\mathrm{T}}$$

其中 $\boldsymbol{I}$ 為單位矩陣，$\boldsymbol{n}^{\mathrm{T}}$ 為 $\boldsymbol{n}$ 的轉置矩陣。將上式展開後，可得：

$$\boldsymbol{M}(n_x,n_y,n_z)=\begin{bmatrix}1-2{n_x}^2 & -2n_xn_y & -2n_xn_z\\ -2n_xn_y & 1-2{n_y}^2 & -2n_yn_z\\ -2n_xn_z & -2n_yn_z & 1-2{n_z}^2\end{bmatrix}$$

(6) 複合線性變換

複合線性變換是將多個線性變換矩陣相乘，從而創造出一個新的線性變換，其效果等同於依照特定順序單獨應用這些變換。舉例來說，如果我們有多個線性變換矩陣 $\boldsymbol{T}_1,\boldsymbol{T}_2,\boldsymbol{T}_3,\dots,\boldsymbol{T}_k$，並且想先用矩陣 $\boldsymbol{T}_1$ 去變換向量 $\boldsymbol{v}$，接著用矩陣 $\boldsymbol{T}_2$ 變換 $\boldsymbol{T}_1$ 的結果，依此類推，直到 $\boldsymbol{T}_k$，則我們可以寫成：

$$\boldsymbol{v}'=\boldsymbol{T}_k\dots\boldsymbol{T}_3\boldsymbol{T}_2\boldsymbol{T}_1\boldsymbol{v}$$

由於矩陣乘法符合結合律，我們可以直接算出複合變換矩陣 $\boldsymbol{T}=\boldsymbol{T}_k\dots\boldsymbol{T}_3\boldsymbol{T}_2\boldsymbol{T}_1$ 並儲存。當需要對任意向量 $\boldsymbol{v}$ 應用這一系列變換時，我們只需要進行一次矩陣乘法 $\boldsymbol{v}'=\boldsymbol{T}\boldsymbol{v}$ 即可。這種方法在計算上非常有效率，特別是在三維圖形處理或物理模擬等場合，我們通常會將相同的變換序列反覆應用於多個不同的向量。

1.2. 仿射變換（Affine Transformation）

在三維空間中，我們可以透過 3×3 矩陣實現多種線性變換操作。然而，這種表示方法無法實現平移變換，我們必須單獨加上一個向量以實現平移。舉例來說，如果我們要使用 $\boldsymbol{T}_1$ 去變換向量 $\boldsymbol{v}$，然後平移 $\boldsymbol{t}_1$，接著再使用 $\boldsymbol{T}_2$ 進行變換，再平移 $\boldsymbol{t}_2$，我們會表達為以下：

$$\boldsymbol{v}' = \boldsymbol{T}_2(\boldsymbol{T}_1\boldsymbol{v} + \boldsymbol{t}_1) + \boldsymbol{t}_2$$

也就是說，我們無法把這種包含平移的變換，整合成一個複合變換矩陣 $\boldsymbol{T} = \boldsymbol{T}_k \dots \boldsymbol{T}_3\boldsymbol{T}_2\boldsymbol{T}_1$ 來加速計算。為了克服這一限制，我們要使用齊次坐標（Homogeneous Coordinates）來擴展線性變換。

齊次坐標是在常規三維坐標系 (x, y, z) 的基礎上增加一個額外的維度 w，我們通常設定 $w = 1$，讓點 (x, y, z) 擴展為 $(x, y, z, 1)$。利用齊次坐標系，我們可以將平移變換納入線性變換矩陣。這時我們的線性變換矩陣將從 3×3 變為 4×4，其中增加的第四行和第四列允許我們進行平移操作。平移變換和線性變換被統稱為「仿射變換」，此擴增矩陣則被稱為「仿射變換矩陣」。

具體的矩陣如下左式，$\boldsymbol{T}$ 為原本的線性變換矩陣，$\boldsymbol{t}$ 為平移向量，$\boldsymbol{v}$ 為變換前的向量，$\boldsymbol{v}'$ 為變換後的向量。我們簡單將矩陣乘開，就可以發現它等價於 $\boldsymbol{v}' = \boldsymbol{T}\boldsymbol{v} + \boldsymbol{t}$，也就是我們習慣的表達。

$$\begin{bmatrix} \boldsymbol{v}' \\ 1 \end{bmatrix} = \left[\begin{array}{c|c} \boldsymbol{T} & \boldsymbol{t} \\ 0 \dots 0 & 1 \end{array}\right]\begin{bmatrix} \boldsymbol{v} \\ 1 \end{bmatrix} \Leftrightarrow \boldsymbol{v}' = \boldsymbol{T}\boldsymbol{v} + \boldsymbol{t}$$

以 $\boldsymbol{T} = \boldsymbol{I}_3$，$\boldsymbol{t} = (5, 7, 8)$ 為例，我們可以直觀的看出它是如何實現平移效果的，如下式。

$$\underset{\text{保持 } \boldsymbol{w} = \mathbf{1}}{\begin{bmatrix} {v_x}' \\ {v_y}' \\ {v_z}' \\ \boxed{1} \end{bmatrix}} = \begin{array}{c} \begin{array}{cccc} & \boldsymbol{T} & & \boldsymbol{t} \end{array} \\ \left[\begin{array}{ccc|c} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 8 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x + 5 \\ v_y + 7 \\ v_z + 8 \\ 1 \end{bmatrix} \Leftrightarrow \boldsymbol{v}' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\boldsymbol{v} + \begin{bmatrix} 5 \\ 7 \\ 8 \end{bmatrix}$$

以 $\boldsymbol{T} = \boldsymbol{R}_x(\pi)$，$\boldsymbol{t} = (5, 7, 8)$ 為例，我們可以看出它實現了「先旋轉再平移」，如下式。

$$\underset{\text{保持 } \boldsymbol{w} = \mathbf{1}}{\begin{bmatrix} {v_x}' \\ {v_y}' \\ {v_z}' \\ \boxed{1} \end{bmatrix}} = \begin{array}{c} \begin{array}{cccc} & \boldsymbol{T} & & \boldsymbol{t} \end{array} \\ \left[\begin{array}{ccc|c} 1 & 0 & 0 & 5 \\ 0 & \cos\pi & -\sin\pi & 7 \\ 0 & \sin\pi & \cos\pi & 8 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x + 5 \\ v_y\cos\pi + v_z\sin\pi + 7 \\ -v_y\sin\pi + v_z\cos\pi + 8 \\ 1 \end{bmatrix}$$

$$\Leftrightarrow \boldsymbol{v}' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\pi & \sin\pi \\ 0 & -\sin\pi & \cos\pi \end{bmatrix} \boldsymbol{v} + \begin{bmatrix} 5 \\ 7 \\ 8 \end{bmatrix}$$

而我們也可以發現，變換後的向量在我們擴增的 w 維度上，始終保持 1，這讓我們可以連續進行變換。透過這種方式，我們就可以將平移表達在 $\boldsymbol{T} = \boldsymbol{T}_k \dots \boldsymbol{T}_3 \boldsymbol{T}_2 \boldsymbol{T}_1$ 的一連串矩陣乘法中。舉例來說，若我們要以 $(10,10,10)$ 為中心，對向量 $\boldsymbol{v}$ 進行 x 軸方向的 $60°$ 旋轉，然後進行 z 軸方向的 $45°$ 旋轉，它會相當於依序進行以下四個變換：平移 $(-10,-10,-10)$、 x 軸方向旋轉 $60°$、 z 軸方向旋轉 $45°$、平移 $(10,10,10)$，可以表達為以下：

$$\begin{bmatrix} v_x{}' \\ v_y{}' \\ v_z{}' \\ 1 \end{bmatrix} = \underbrace{\left[\begin{array}{ccc|c} \cos 45° & -\sin 45° & 0 & 10 \\ \sin 45° & \cos 45° & 0 & 10 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{array}\right]}_{\substack{\boldsymbol{z}\text{ 軸方向旋轉 }\mathbf{45°}\text{，} \\ \text{然後平移 }(\mathbf{10},\mathbf{10},\mathbf{10})}} \underbrace{\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos 60° & -\sin 60° & 0 \\ 0 & \sin 60° & \cos 60° & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]}_{\boldsymbol{x}\text{ 軸方向旋轉 }\mathbf{60°}} \underbrace{\left[\begin{array}{ccc|c} 1 & 0 & 0 & -10 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & -10 \\ 0 & 0 & 0 & 1 \end{array}\right]}_{\text{平移 }(\mathbf{-10},\mathbf{-10},\mathbf{-10})} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix}$$

1.3. 反向映射（Inverse Mapping）

如果我們想對一張影像進行仿射變換，最直觀的方法是將變換矩陣應用於原始坐標（原始點），以計算出它們變換後的位置（目標點）。例如，對一張 10×10 的影像進行以原點為中心的 $30°$ 旋轉，接著平移$(1,1)$，則原始點 $(2,3)$ 的變換可表示為：

$$\begin{bmatrix} \cos 30° & -\sin 30° & 1 \\ \sin 30° & \cos 30° & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 1.23 \\ 4.59 \\ 1 \end{bmatrix}$$

我們可以發現原始點 $(2,3)$ 被變換到目標點 $(1.23, 4.59)$，目標點的坐標並不是整數，也就是它無法直接對應到整數的像素位置上。若進一步對變換後的坐標進行四捨五入，將其直接作為像素的坐標，則可能會出現某些目標點沒有被對應到（即出現「空洞」），或多個原始點映射到同一個目標點（即出現「堆疊」）的情形，並不可行。

為了解決這些問題，我們可以採用反向映射（Inverse Mapping）的方法。在反向映射中，我們從目標影像的角度出發，對每一個目標點進行處理，反向計算它們對應到的原始點。若原始點坐標非整

數，則取周圍坐標進行插值。舉例來說，如果 $V(x,y)$ 表示原始點 (x,y) 的像素值，我們可以透過以下步驟求出目標點 $(4,3)$ 的值：

(1) 原始變換矩陣 $\boldsymbol{T} = \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 1 \\ \sin 30^\circ & \cos 30^\circ & 1 \\ 0 & 0 & 1 \end{bmatrix}$，可求出其逆矩陣 $\boldsymbol{T}^{-1} = \begin{bmatrix} 0.87 & 0.50 & -1.37 \\ 0.50 & 0.87 & -0.37 \\ 0 & 0 & 1 \end{bmatrix}$

(2) $\boldsymbol{T}^{-1}\begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.87 & 0.50 & -1.37 \\ 0.50 & 0.87 & -0.37 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.60 \\ 0.23 \\ 1 \end{bmatrix}$，可知目標點 $(4,3)$ 對應到原始點 $(3.60, 0.23)$

(3) 假設我們採取雙線性插值（Bilinear Interpolation），如圖 2，那麼會寫成以下：

a. 找出鄰近像素點，這裡是 $(3,0)$、$(3,1)$、$(4,0)$、$(4,1)$

b. 先對 $y=0$ 和 $y=1$ 分別進行 x 方向的線性插值

$R_1 = V(3,0)\cdot(4-3.60) + V(4,0)\cdot(3.60-3) \cdots y = 0$

$R_2 = V(3,1)\cdot(4-3.60) + V(4,1)\cdot(3.60-3) \cdots y = 1$

c. 再對 R_1 和 R_2 進行 y 方向的線性插值，即完成插值

$V(3.60, 0.23) = R_1\cdot(1-0.23) + R_2\cdot(0.23-0)$

※ 所有不存在原圖的像素點，像素值定義為 0。例如要求原始點 $(3.60,-0.53)$ 的值，其鄰近像素點是 $(3,-1)$、$(3,0)$、$(4,-1)$、$(4,0)$，此時 $V(3,-1)$ 與 $V(3,-1)$ 皆為 0。

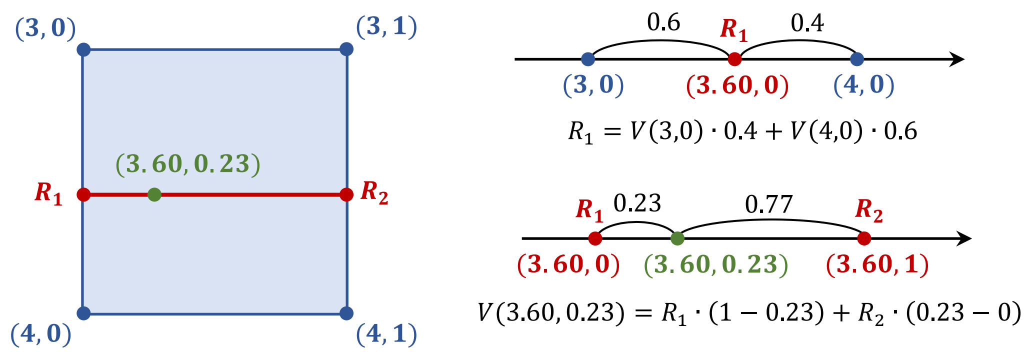


圖 2 雙線性插值

當我們使用 OpenCV 提供的 `cv::resize` 及 `cv::warpAffine` 等函式進行影像處理，實際上是在對所有目標點進行上述的三個步驟，並且這些函式預設採用雙線性插值。多數插值過程會改變圖像的像素分布，可能會導致細節和資訊的損失。這樣的損失通常是不可逆的，使用時需要特別注意。

2. 作業描述

本次作業共分為兩大部分，分別為「點的變換」與「立體影像的變換」，以下將分別說明。有兩個參數是以命令列引數的方式給定，其餘參數寫在 `input1.txt` 和 `input2.txt` 中，說明如表 1。請注意，路徑本身已經含有副檔名。

表 1 命列列引數

輸入方式	位置	本文件中的名字	解釋
命令列引數	`argv[1]`	`inputPath1`	`input1.txt` 的路徑，為第一部分的輸入。
	`argv[2]`	`outputPath1`	`output1.txt` 的路徑，為第一部分的輸出。
	`argv[3]`	`inputPath2`	`input1.txt` 的路徑，為第二部分的輸入。
	`argv[4]`	`outputPath2`	`output1.txt` 的路徑，為第二部分的輸出。

2.1. 第一部分：點的變換（共 50 分）

給定 n 個仿射變換 $\boldsymbol{T}_1, \boldsymbol{T}_2, \boldsymbol{T}_3, \ldots, \boldsymbol{T}_n$，每個變換都可以用一個 4×4 的矩陣來表示。這些變換依序作用於向量 $\boldsymbol{v} = (v_x, v_y, v_z, 1)$，產生新的向量 $\boldsymbol{u} = (u_x, u_y, u_z, 1)$。若將複合變換矩陣 $\boldsymbol{T}$ 定義為所有變換矩陣的乘積，即 $\boldsymbol{T} = \boldsymbol{T}_n \ldots \boldsymbol{T}_3 \boldsymbol{T}_2 \boldsymbol{T}_1$，則變換關係可以表示為：

$$\boldsymbol{u} = \boldsymbol{T}_n \ldots \boldsymbol{T}_3 \boldsymbol{T}_2 \boldsymbol{T}_1 \boldsymbol{v} = \boldsymbol{T}\boldsymbol{v}$$

請你撰寫一個 C++ 程式，完成以下四小題並依序輸出結果：

(a) 計算複合變換矩陣 $\boldsymbol{T}$。

(b) 根據給定的四個向量 $\boldsymbol{v}$，計算出相對應的新向量 $\boldsymbol{u}$。

(c) 承接上一小題，如果四個向量 $\boldsymbol{v}$ 分別代表一個四面體的四個頂點，且構成的四面體體積為 A（並保證 A 為正值）。而四個新向量 $\boldsymbol{u}$ 構成的四面體體積為 B。請計算體積比 $r = B/A$ 與矩陣 $\boldsymbol{T}$ 的行列式值 $\det(\boldsymbol{T})$，並觀察 r 與 $\det(\boldsymbol{T})$ 是否相等或互為相反數。

(d) 根據給定的一個向量 $\boldsymbol{u}$，反過來求向量 $\boldsymbol{v}$。若 $\boldsymbol{v}$ 不存在或不唯一，請輸出字串 `"NaN"`。

請先讀取存放在路徑 `inputPath1` 下的 `input1.txt` 文件，這是第一部分的輸入資料。輸入資料

的詳細格式和範例說明請參考表 2 和圖 3。輸入的第 1 ~ 4 行為 (b) 小題要變換的四個向量 $\boldsymbol{v}$，第 5 行為 (d) 小題要變換的向量 $\boldsymbol{v}$，第 6 行的整數 n 代表總共有幾個變換，第 7 行起會有 n 個變換按照固定格式輸入，每個變換的首行會以字元 `'#'` 開頭，說明這是一個變換的開始。變換的輸入格式請參考表 3。

在運算完成後，請依照以下格式，在路徑 `outputPath1` 的 `output1.txt` 輸出四小題的答案。若答案包含小數，請四捨五入至小數點後二位。在輸出數據時，即使小數點後的數字為零，仍需保留至小數點後第二位。舉例來說，如果計算結果為整數 `1`，輸出時應格式化為 `1.00`。輸出範例如圖 4。

(a) 輸出 a_1、a_2、a_3、a_4，彼此以空格分隔，然後換行，接著繼續輸出 a_5、a_6、a_7、a_8，彼此以空格分隔，然後換行，以此類推。

(b) 共有四行，每行代表一個向量 $\boldsymbol{u}$。對於每個向量 $\boldsymbol{u}$ ，輸出求出的 u_x、u_y、u_z，彼此以空格分隔，然後換行。

(c) 請依序輸出「體積比 r」與「行列式值 $\det(\boldsymbol{T})$」，彼此以空格分隔，然後換行。接著輸出字串 `"r==det(T)"`、`"r==-det(T)"`、`"zeros"`、`"others"` 其中一者，然後換行（`"r==det(T)"` 表示兩者相等且非零；`"r==-det(T)"` 表示兩者互為相反數且非零；`zeros` 表示兩者皆為零；`others` 表示不是前面三個情況）。

(d) 共有一行，代表一個向量 $\boldsymbol{v}$。若向量 $\boldsymbol{v}$ 存在且唯一，則輸出求出的 v_x、v_y、v_z，彼此以空格分隔，然後換行；否則輸出字串 `"NaN"`，然後換行。

表 2 input1.txt 格式

位置	本文件中的名字	解釋
第 1 行	$\boldsymbol{v}$	每行包含三個整數，以空白分隔，分別代表 v_x、v_y、v_z，組成向量 $\boldsymbol{v} = (v_x, v_y, v_z)$。與 (b) 小題有關。
第 2 行		
第 3 行		
第 4 行		

第 5 行	$\boldsymbol{u}$	每行包含三個整數，以空白分隔，分別代表 u_x、u_y、u_z，組成向量 $\boldsymbol{u}=(u_x,u_y,u_z)$。與 (d) 小題有關。
第 6 行	n	一個整數 n，表示共有幾個變換矩陣（$1 \le n \le 20$）。
第 7 行起	$\boldsymbol{T}_1,\boldsymbol{T}_2,\dots,\boldsymbol{T}_n$	符合固定格式的 n 個變換矩陣。輸入的參數可能為浮點數。

表 3 變換的格式說明 輸入的參數可能為浮點數（如 t,c,k,θ,s,t,a_{ij}）

名稱	輸入格式	解釋
平移	#T t_x t_y t_z	表示平移 (t_x,t_y,t_z)。
投影	#Pyz	表示正射投影到 yz 平面。同理，也可能出現 Pxz 和 Pxy。
縮放	#S c_x c_y c_z k_x k_y k_z	表示以點 $\boldsymbol{C}=(c_x,c_y,c_z)$ 為中心，分別沿 x、y、z 軸進行縮放變換，其中 k_x、k_y、k_z 為三軸的縮放因子。
旋轉	#Rx c_x c_y c_z θ	表示以點 $\boldsymbol{C}=(c_x,c_y,c_z)$ 為中心，繞 x 軸方向旋轉角度 $\theta°$。同理，也可能出現繞 y 軸（Ry）和 z 軸（Rz）的旋轉。
推移	#Hx c_x c_y c_z s t	表示以點 $\boldsymbol{C}=(c_x,c_y,c_z)$ 為中心，沿 x 軸進行推移，其中 s 和 t 為推移矩陣的兩個參數，具體定義請參考本文件第三頁。同理，也可能出現沿 y 軸（Hy）和 z 軸（Hz）的推移。
自訂	#M a_{11} a_{12} a_{13} a_{14} a_{21} a_{22} a_{23} a_{24} a_{31} a_{32} a_{33} a_{34} 0 0 0 1	表示使用自定義的 4×4 變換矩陣進行變換。該矩陣如下所示，最後一列固定為 $[0\quad 0\quad 0\quad 1]$。 $T=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

```
1 2 3
4 6 8
3 9 0
0 2 5
0 0 0
4
#T 5 4 3
#Rx 5 4 3 120
#M
1 2 3 4
5 6 5 4
3 2 1 0
0 0 0 1
#Hx 5 4 3 -1 1
```

- b 小題：給定 $\boldsymbol{v}=(1,2,3)$，求 $\boldsymbol{u}$
- b 小題：給定 $\boldsymbol{v}=(4,6,8)$，求 $\boldsymbol{u}$
- b 小題：給定 $\boldsymbol{v}=(3,9,0)$，求 $\boldsymbol{u}$
- b 小題：給定 $\boldsymbol{v}=(2,0,5)$，求 $\boldsymbol{u}$
- d 小題：給定 $\boldsymbol{u}=(0,0,0)$，求 $\boldsymbol{v}$
- $\boldsymbol{n}=4$，共有 4 個變換矩陣
- $\boldsymbol{T}_1$：平移 $(5,4,3)$
- $\boldsymbol{T}_2$：以 $(5,4,3)$ 為中心，繞 $\boldsymbol{x}$ 軸方向旋轉 120°
- $\boldsymbol{T}_3$：用自訂義的矩陣進行變換，矩陣如下

$$\boldsymbol{T}_3=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 5 & 4 \\ 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $\boldsymbol{T}_4$：以 $(5,4,3)$ 為中心，沿 $\boldsymbol{x}$ 軸推移 $\boldsymbol{H}_x(-1,1)$

圖 3 `input1.txt` 範例

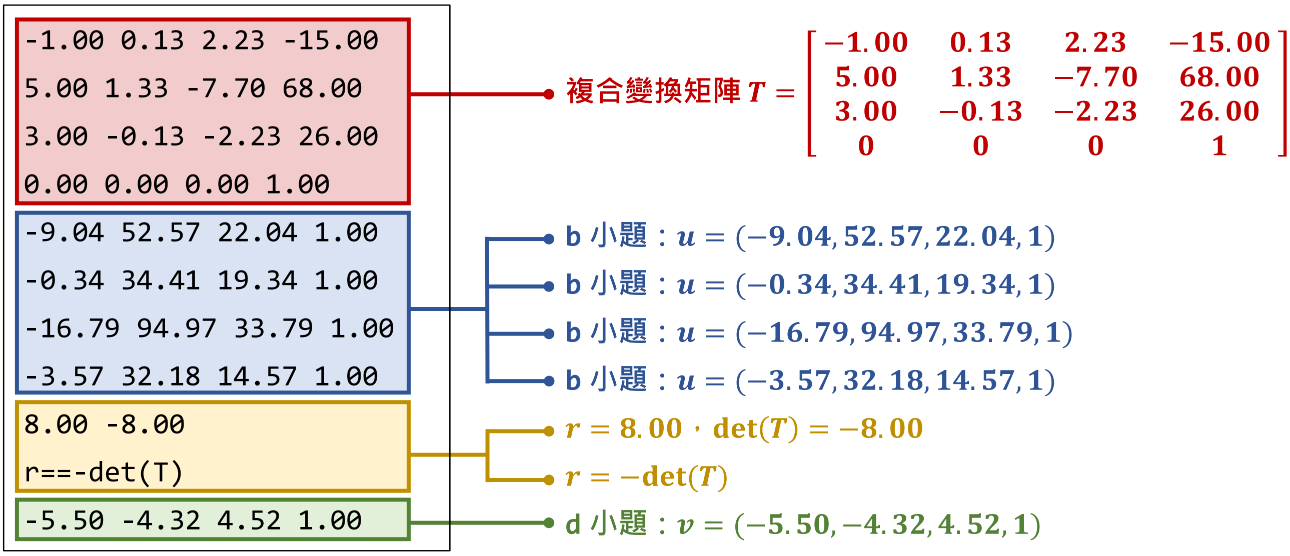


圖 4 `output1.txt` 範例

2.2. 第二部分：立體影像的變換（共 50 分）

醫學影像研究經常面臨數據不足的問題，主要原因是病人隱私保護和醫院患者數量有限，導致資料難以獲得。對於需要大量數據的深度學習來說，這個問題往往會成為最大的瓶頸。然而，在影像深度學習模型中，大多數都採用 CNN（Convolutional Neural Networks）架構。CNN 的主要特點是依據「像素的鄰近排列」和「特徵捕捉」來進行學習及識別圖案。也就是說，即使只是對原始圖像進行一點小變化，如圖 5，CNN 也會認為它們是不同影像，甚至無法識別，因為像素排列或特徵大小已經與原圖不同。(如果對 CNN 的原理有興趣，可以參考影片：youtube.com/watch?v=OP5HcXJg2Aw)

圖 5 影響 CNN 的變化 (a) 原圖；(b) 旋轉 30 度；(c) 縮小至 80%

基於 CNN 的特點，我們可以透過資料增強（Data Augmentation）來應對資料不足。資料增強指的是對原始資料進行一系列的變換，如旋轉、縮放或位移，來創造出更多數據，增加資料量和多樣性。

本次作業的第二部分，我們將實作立體醫學影像的資料增強。有 h 張大小為 $l \times w$ 的切片，這些切片共同構成一組立體影像。定義 h 張切片沿著 z 軸排列，其坐標範圍從 0 到 $h-1$。同時，每張切片的長度 l 將對應到 x 軸的坐標範圍，從 0 到 $l-1$，而寬度 w 則對應到 y 軸的坐標範圍，從 0 到 $w-1$。如此一來，我們可以將每一個點在立體影像中的位置表示為三維坐標 (x, y, z)，如圖 6。

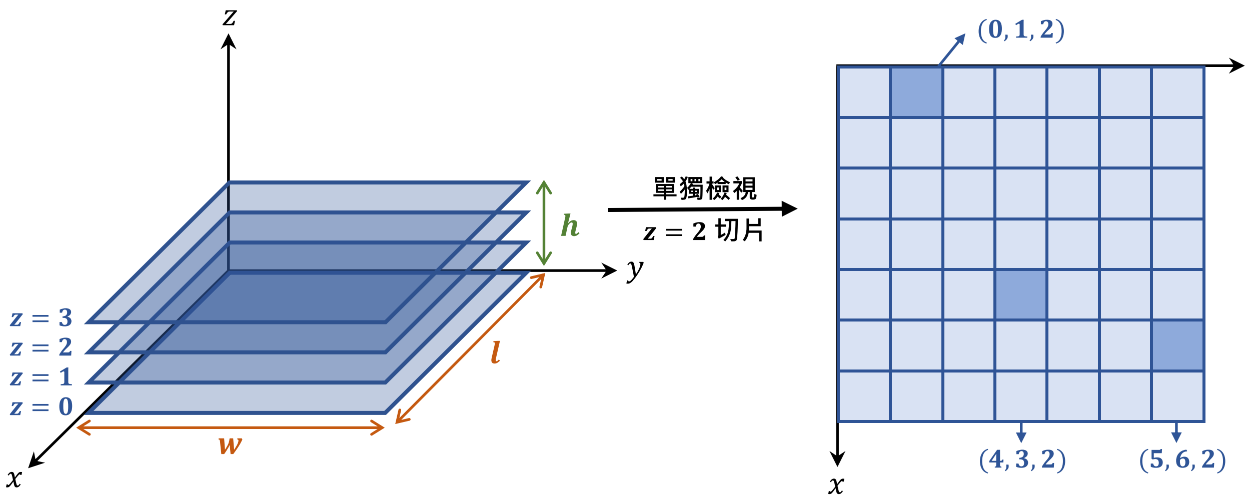
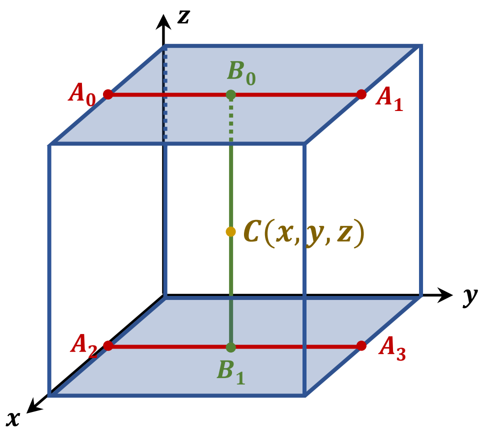


圖 6 立體影像坐標的定義

給定 n 個仿射變換 $\boldsymbol{T}_1, \boldsymbol{T}_2, \boldsymbol{T}_3, \dots, \boldsymbol{T}_n$，每個變換都可以用一個 4×4 的矩陣來表示，並且**保證可逆**。請撰寫一個 C++ 程式，將這些變換依序作用於原始的立體影像上，求出變換後的立體影像。請使用「反向映射」方法並採用三線性插值（Trilinear Interpolation）計算。三線性插值是雙線性插值在三維空間中的擴展（請先理解第六頁的雙線性插值），具體步驟請參考圖 7。所有不存在原影像的像素點，像素值都定義為 0。插值後請截斷像素值，以整數型態儲存和輸出。



使用三線性插值求出 C 點的像素值（x, y, z 非整數）

(1) 找出 C 點的 8 個鄰近像素點

(2) 沿 x 方向插值：從 8 個鄰近像素點插值出 A_0, A_1, A_2, A_3

(3) 沿 y 方向插值：從 A_0, A_1, A_2, A_3 插值出 B_0, B_1

(4) 沿 z 方向插值：從 B_0, B_1 插值出 C

圖 7 三線性插值

請先讀取存放在路徑 `inputPath2` 下的 `input2.txt` 文件，這是第二部分的輸入資料。輸入資料的具體格式和範例說明請參考表 4 和圖 8。輸入第 1 行的三個整數 l, w, h 表示立體影像的大小，第 2 行起為立體影像的像素值，每張切片以 l 行描述，共有 h 張，總計 $l \times h$ 行，描述完像素值的下一行有一個整數 n，代表總共有幾個變換，再下一行起會有 n 個變換按照固定格式輸入。變換的輸入格式請參考表 3。

在運算完成後，請依照指定格式，在路徑 `outputPath2` 的 `output2.txt` 輸出變換後的立體影像。立體影像的輸出格式與輸入相同，也就是先輸出第一張切片，再輸出第二張，一直到第 h 張。每張切片以 l 行描述，每行有w 個整數，以空白分隔。輸出範例如圖 9。

表 4 `input2.txt` 格式

位置	本文件中的名字	解釋
第 1 行	l, w, h	包含三個整數，以空白分隔，代表立體影像的 l, w, h。
第 2 行起	立體影像	前 $1 \sim l$ 行描述第一張切片（$z = 0$），每行有 w 個整數，整數的值介於 0 到 255 之間，以空白分隔，代表各點的像素值。第 $(l+1) \sim 2 \times l$ 行描述第二張切片（$z = 1$），依此類推。
再下一行	n	一個整數 n，表示共有幾個變換矩陣（$1 \leq n \leq 20$）。
下二行起	$\boldsymbol{T}_1, \boldsymbol{T}_2, \dots, \boldsymbol{T}_n$	符合固定格式的 n 個變換矩陣。輸入的參數可能為浮點數。

```
3 5 2
10 30 50 70 90
70 50 30 10 30
50 70 90 70 50
110 130 150 170 190
170 150 130 110 130
150 170 190 170 150
2
#T 0 -1 0
#Rx 1 2 0.5 10
```

立體影像的 $l = 3, w = 5, h = 2$

第一張切片（大小是 3×5）各點的像素值（$z = 0$）

第二張切片（大小是 3×5）各點的像素值（$z = 1$）

$n = 2$，共有 2 個變換矩陣

$\boldsymbol{T}_1$：平移 $(0, -1, 0)$

$\boldsymbol{T}_2$：以 $(1, 2, 0.5)$ 為中心，繞 x 軸方向旋轉 $10°$

圖 8 `input2.txt` 範例

```
64 66 69 73 6
86 49 12 23 2
104 106 72 43 3
87 126 170 159 0
97 106 110 103 0
113 156 167 122 0
```

變換後的第一張切片，各點的像素值（$z = 0$）

變換後的第二張切片，各點的像素值（$z = 1$）

圖 9 `output2.txt` 範例

3. 輸入與輸出

程式執行一次只要處理一筆測資，需要接收兩個命令列引數。在自己電腦嘗試時，請將 `input1.txt` 和 `input2.txt` 中的絕對路徑**改成你自己的路徑**，並確保路徑中沒有中文，此外讀寫圖片請使用 png 檔，以避免一些常見錯誤。請注意，**每行輸出結尾都沒有空格**，若多加空格會不通過。

4. 評分標準

本次作業共有 8 筆測資，皆為隱藏測資，每筆測資佔 10 分，共 80 分。測資部分使用線上批改系統自動批改，採分段給分，兩部分獨立計分互不影響：

(1) 「點的變換」的輸出正確，佔 5 分。

(2) 「立體影像的變換」的輸出正確，佔 5 分。

若不使用 OpenCV 在批改系統中達到 70 分(也就是必須撰寫出一個程式，沒有使用到 OpenCV，且能在批改系統中達到 70 分)，並完成以下兩條件，可再獲得 40 分的分數：

(1) 將「矩陣乘法」、「求行列式值」、「求逆矩陣」三個運算在 C++ 中分別寫成 function。

(2) 撰寫文件說明如何實作這三個 function，並解釋其背後原理，上傳至 Portal 作業區。文件中請盡量證明自己確實理解程式碼及其背後的線性代數運算，而不是單純複製網路上的程式碼。

（若說明不夠詳細或方法不合理，會酌量扣分）

【BONUS】前 10% 繳交作業（包含文件）且分數達 90 分者，最後會再額外加 10 分。

【BONUS】程式平均執行時間前 10% 且分數達 90 分者，最後會再額外加 10 分。

【BONUS】提供測試資料(不可為公開測資)且有滿分的同學程式執行錯誤，最後會再額外加 10 分。

本次作業的攻擊方式不包括使用巨大立體影像讓程式超時。

作業程式碼將進行相似度比對，**對於較為相似的程式，我們會現場 Demo 確保不是抄襲**。對於抄襲或作弊的程式，一律視為 0 分。

5. 線上批改系統與環境

請將程式碼上傳至老師的線上批改系統 http://dslab.csie.org/course/1121LA/。執行環境如下表。本作業限定使用 C++ 撰寫，且不提供 OpenCV 以外的第三方函式庫。

作業系統	**Ubuntu 22.04**
編譯器	**g++ 11.4.0**
OpenCV 版本	**opencv 4.5.3**

6. 繳交期限

2023/12/4 23:59。

7. 提示

程式必須考慮浮點數誤差 (詳情可以參考 https://ppt.cc/foIp4x)。如果作業一有其中一個測資沒有通過，很可能是遺漏這個部分。浮點數誤差範例如下：

```
int main() {
    float a = 0.4;
    float b = 0.3;
    float c = 10 * (a - b);
    cout << fixed << setprecision(10) << "output:\n";
    cout << "c: " << c << endl;
    cout << "floor(c): " << int(c) << endl;

    float d = 0.71 * 10;
    if (d == 7.1)
        cout << "d == 7.1" << endl;
    else
        cout << "d != 7.1" << endl;
    cout << "d: " << d << endl;
}
```

```
output:
c: 0.9999999404
floor(c): 0
d != 7.1
d: 7.0999999046
```

(terminal)

8. 附註

若有成績的相關疑慮，請回報至 yzu1607a@gmail.com（標題：[線性代數] sXXXXXXX 作業二問題）或於 Discord 伺服器問答區發問。為了保持公平性，我們不協助 Debug，只會確認是否為批改系統系統錯誤。