基于 l3fparray 的轻量矩阵
Lightweight Matrix Based on l3fparray

Ms_yam*

2026-11-23

1 用户文档 [English]

1.1 宏包简介

lt3matrix 宏包提供一种轻量矩阵类型，主要用于 TEX 文档中的矩阵演示、数值示例和几何变换。使用者可以创建矩阵、设置矩阵元素、执行矩阵运算、遍历元素，并将矩阵排版到数学公式中。

本宏包中的矩阵具有确定的行数和列数。通常，矩阵创建后其维度不应修改，且本宏包不提供直接修改矩阵维度的接口；但当矩阵作为运算结果时，其维度会根据运算结果自动更新。矩阵元素则可以通过赋值命令修改。矩阵的创建和修改均为全局操作：在 TEX 分组内执行修改命令，分组结束后矩阵内容不会自动恢复。

空矩阵也具有明确的行数和列数，但不包含任何元素。行数和列数不能为负数。

在几何变换中，允许用户通过叠加基础变换的形式来构建完整的变换。虽然内部实现使用的是齐次变换矩阵，但用户从头到尾不需要任何相关知识。

应用变换时，通过变换矩阵动态定义变换函数，然后使用变换函数快速计算变换后的坐标。每次执行变换，都只是执行对应的四则运算（由 fp 模块执行），而不需要重复从矩阵中获取数据，从而大幅提高运算速度。

1.2 快速开始

下面的例子创建一个 2×3 矩阵，然后设置元素，并在数学模式中排版：

```
\ExplSyntaxOn
\matrix_new:Nnn \l_demo_matrix { 2 } { 3 }
\matrix_gset:Nn \l_demo_matrix { 1, 2, 3, 4, 5, 6 }
```

*E-mail: Ms_yam@163.com

```
\[
  \matrix_use:N \l_demo_matrix
\]
\ExplSyntaxOff
```

其效果为：

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

1.3 命令与参数约定

本宏包主要作为 expl3 的基础扩展宏包，其主要接口遵循 expl3 的命名规范，使用时应处于 `\ExplSyntaxOn` 状态。此外，为方便不熟悉 expl3 的 LaTeX 用户使用，本宏包也为几何变换等常用操作提供了相应的 LaTeX2e 风格命令。

expl3 风格命令名称中，冒号后的字母表示参数的形式：`N` 接收单个控制序列，`n` 接收按原样传入的一般参数（通常写在一对花括号中），`c` 则根据给出的字符名称构造控制序列。`e` 表示先对相应参数进行扩展，再将结果传入命令。因此，`N` 形式直接接收矩阵变量，例如 `\matrix_new:Nnn`；`c` 形式接收矩阵变量的名称（不带反斜杠），例如 `\matrix_new:cnn {l_demo_matrix} {2} {2}`。同一组命令的不同变体执行相同操作，区别仅在于相应参数的传递形式。

在本文档中，⟨*matrix*⟩ 表示矩阵变量；⟨*destination*⟩ 表示接收结果的目标矩阵；⟨*source*⟩ 表示作为输入的源矩阵。除创建命令外，作为参数传入的矩阵都应当已经创建。⟨*comma list*⟩ 表示逗号分隔的元素列表。

以 `g` 开头的矩阵修改命令采用全局赋值。矩阵的更改不会因普通 TeX 分组结束而恢复。

1.4 基础命令

1.4.1 创建矩阵

`\matrix_new:N` `\matrix_new:Nnn` `\matrix_new:cnn`

`\matrix_new:N` ⟨*matrix*⟩
`\matrix_new:Nnn` ⟨*matrix*⟩ {⟨*rows*⟩} {⟨*columns*⟩}

创建一个普通矩阵，所有元素均初始化为零。

Updated: 2026-11-18

不带维度参数时，创建一个 0×0 的空矩阵；指定 ⟨*rows*⟩ 和 ⟨*columns*⟩ 时，分别创建具有相应行数和列数的矩阵。两个维度参数均按整数表达式求值，且必须为非负数。维度允许为零。

0×0 的空矩阵通常用于存储矩阵运算的结果。如果矩阵变量已经定义，则产生错误。

\matrix_const:Nnnn
\matrix_const:cnnn

\matrix_const:Nnnn ⟨*matrix*⟩ {⟨*rows*⟩} {⟨*columns*⟩} {⟨*comma list*⟩}

创建一个常量矩阵，并使用逗号分隔列表初始化其元素。元素按照行优先顺序填入；列表中未指定的元素初始化为零。常量矩阵不能通过本宏包提供的矩阵修改命令进行修改。

⟨*matrix*⟩、⟨*rows*⟩ 和 ⟨*columns*⟩ 的含义与 \matrix_new:Nnn 相同。⟨*comma list*⟩ 是矩阵元素列表，每一项均按浮点数表达式求值。列表项数不得超过矩阵的元素总数；如果超过，则产生错误，且矩阵不会被创建。

1.4.2 设置和修改矩阵

以下命令用于修改已创建矩阵的内容。目标矩阵必须是普通矩阵；如果目标矩阵是常量矩阵，则会产生错误。所有修改操作均为全局操作。

\matrix_gzero:N
\matrix_gzero:c

\matrix_gzero:N ⟨*matrix*⟩

将 ⟨*matrix*⟩ 的所有元素设为零，同时保持矩阵的维度（行数和列数）不变。

\matrix_gset:Nn
\matrix_gset:(cn|Ne|ce)

Updated: 2026-11-19

\matrix_gset:Nn ⟨*matrix*⟩ {⟨*comma list*⟩}

使用逗号分隔列表重新设置 ⟨*matrix*⟩ 的所有元素，并按照行优先顺序写入。列表中的每一项均按浮点数表达式求值。列表项数不得超过矩阵的元素总数；如果列表项数不足，则剩余元素设为零。如果列表项数过多，则产生错误，且本次设置不会执行。矩阵的的维度（行数和列数）保持不变。

\matrix_gset_eq:NN
\matrix_gset_eq:(cN|Nc)

\matrix_gset_eq:NN ⟨*destination*⟩ ⟨*source*⟩

将 ⟨*source*⟩ 的维度和元素复制到 ⟨*destination*⟩。目标矩阵的维度会相应更新。复制完成后，两个矩阵的数据相互独立；后续修改其中一个矩阵不会影响另一个矩阵。

⟨*destination*⟩ 必须是已创建的普通矩阵；⟨*source*⟩ 必须是已创建的矩阵，可以是普通矩阵或常量矩阵。源矩阵不会被修改。

1.4.3 显示和检查矩阵

\matrix_use:N
\matrix_use:c

\matrix_use:N ⟨*matrix*⟩

在数学模式中排版 ⟨*matrix*⟩。非空矩阵以圆括号矩阵的形式显示；空矩阵则显示其维度信息，例如 2×0。本命令不会自动进入数学模式，因此必须在数学模式中调用。

\matrix_log:N
\matrix_log:c

\matrix_log:N ⟨*matrix*⟩

将 ⟨*matrix*⟩ 的维度、存储信息和元素内容写入编译日志，用于检查矩阵状态和调试。本命令不会在页面正文中排版矩阵。

1.4.4 预定义矩阵

`\c_empty_matrix` 0×0 空矩阵。

`\c_identity_two_matrix` 分别为 2×2、3×3 和 4×4 单位矩阵。
`\c_identity_three_matrix`
`\c_identity_four_matrix`

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}、\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}、\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`\g_tmpa_matrix` 用于全局赋值的临时矩阵变量。本宏包的内部代码不会使用这些变量，因此可以安
`\g_tmpb_matrix` 全地与本宏包提供的命令一起使用。但是，它们可能会被其他代码覆盖，因此只能用于短期存储。

1.5 条件、映射与运算

1.5.1 矩阵条件句

`\matrix_if_empty_p:N` ⋆ `\matrix_if_empty:NTF` ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}
`\matrix_if_empty:NTF` ⋆ 判断 ⟨*matrix*⟩ 是否为空矩阵。如果矩阵不包含任何元素，则条件成立，与矩阵的行数和列数无关。因此，0×0、2×0 和 0×3 矩阵均为空矩阵。
New: 2026-11-19

`\matrix_if_square_p:N` ⋆ `\matrix_if_square:NTF` ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}
`\matrix_if_square:NTF` ⋆ 判断 ⟨*matrix*⟩ 是否为方阵，即行数和列数是否相等。空矩阵也参与维度判断，例如，0×0 矩阵是方阵，而 2×0 矩阵不是方阵。
New: 2026-11-19

`\matrix_if_same_dim_p:NN` ⋆ `\matrix_if_same_dim:NNTF` ⟨*matrix1*⟩ ⟨*matrix2*⟩ {⟨*true code*⟩} {⟨*false code*⟩}
`\matrix_if_same_dim:NNTF` ⋆ 判断两个矩阵的维度是否相同，即行数和列数是否分别相等。矩阵元素的内容不影响判断结果。
New: 2026-11-19

`\matrix_if_same_dim_p:Nnn` ⋆ `\matrix_if_same_dim:NnnTF` ⟨*matrix*⟩ {⟨*rows*⟩} {⟨*columns*⟩}
`\matrix_if_same_dim:NnnTF` ⋆ {⟨*true code*⟩} {⟨*false code*⟩}
New: 2026-11-19

判断 ⟨*matrix*⟩ 的维度是否与指定维度相同。⟨*rows*⟩ 和 ⟨*columns*⟩ 分别指定行数和列数，并按整数表达式求值。

\matrix_if_zero:N*TF* \matrix_if_zero:NTF ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

New: 2026-11-19 判断 ⟨*matrix*⟩ 是否为零矩阵，即矩阵中的所有元素是否均为零。浮点数比较采用 10^{-12} 的容差。空矩阵不视为零矩阵，因此该条件对空矩阵不成立。

TEX 黑客笔记：本命令会使用内部临时变量。如果在映射中调用此判定时出现异常，请尝试使用 TEX 分组隔离临时变量。

\matrix_if_identity:N*TF* \matrix_if_identity:NTF ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

New: 2026-11-19 判断 ⟨*matrix*⟩ 是否为单位矩阵。单位矩阵必须是方阵，主对角线上的元素均为 1，其余元素均为 0；浮点数比较采用 10^{-12} 的容差。空矩阵因为不包含元素 1，因此按非单位矩阵处理。

TEX 黑客笔记：本命令会使用内部临时变量。如果在映射中调用此判定时出现异常，请尝试使用 TEX 分组隔离临时变量。

1.5.2 逐元素运算

本节介绍对矩阵中的每个元素分别执行相同运算的命令。根据结果的存储方式，命令分为两种形式：一种直接将结果写回原矩阵，另一种将结果写入另一个目标矩阵。用于存储运算结果的矩阵都必须是已创建的普通矩阵。

直接写回原矩阵的形式保持矩阵的维度不变；写入目标矩阵的形式则会将目标矩阵的维度更新为与源矩阵相同。当目标矩阵与源矩阵是同一个矩阵时，运算结果仍会写回该矩阵，但相比直接写回原矩阵的形式，需要额外的临时存储分配和回收操作。因此，这种情况下建议使用直接写回原矩阵的形式。

\matrix_gnegate:N \matrix_gnegate:N ⟨*matrix*⟩
\matrix_gabs:N \matrix_gabs:N ⟨*matrix*⟩
\matrix_gsign:N \matrix_gsign:N ⟨*matrix*⟩

分别将 ⟨*matrix*⟩ 的每个元素替换为其相反数、绝对值和符号值，并将运算结果写回其自身。

\matrix_ground:Nn \matrix_ground:Nn ⟨*matrix*⟩ {⟨*digits*⟩}

对 ⟨*matrix*⟩ 的每个元素执行舍入，并将结果写回其自身。⟨*digits*⟩ 是传递给 l3fp 的 round 运算的精度参数。

\matrix_gnegate:NN \matrix_gnegate:NN ⟨*destination*⟩ ⟨*source*⟩
\matrix_gabs:NN \matrix_gabs:NN ⟨*destination*⟩ ⟨*source*⟩
\matrix_gsign:NN \matrix_gsign:NN ⟨*destination*⟩ ⟨*source*⟩

对 ⟨*source*⟩ 的每个元素执行相应运算，并将结果写入 ⟨*destination*⟩。目标矩阵的维度更新为与源矩阵相同。

\matrix_ground:NNn \matrix_ground:NNn ⟨*destination*⟩ ⟨*source*⟩ {⟨*digits*⟩}

对 ⟨*source*⟩ 的每个元素执行舍入，并将结果写入 ⟨*destination*⟩。⟨*digits*⟩ 的含义与 \matrix_ground:Nn 中相同。

\matrix_gadd:Nn \matrix_gadd:Nn ⟨*matrix*⟩ {⟨*scalar*⟩}
\matrix_gsub:Nn \matrix_gsub:Nn ⟨*matrix*⟩ {⟨*scalar*⟩}
\matrix_gmul:Nn \matrix_gmul:Nn ⟨*matrix*⟩ {⟨*scalar*⟩}
\matrix_gdiv:Nn \matrix_gdiv:Nn ⟨*matrix*⟩ {⟨*scalar*⟩}
\matrix_gpower:Nn \matrix_gpower:Nn ⟨*matrix*⟩ {⟨*scalar*⟩}

将矩阵 ⟨*matrix*⟩ 的每个元素分别与 ⟨*scalar*⟩ 执行加、减、乘、除或乘方运算，并将结果写回其自身。对于矩阵 A 和标量 c，对应的逐元素运算依次为 $A+c$、$A-c$、$A\times c$、A/c 和 A^c。⟨*scalar*⟩ 按 l3fp 浮点表达式求值。

\matrix_gadd:NNn \matrix_gadd:NNn ⟨*destination*⟩ ⟨*source*⟩ {⟨*scalar*⟩}
\matrix_gsub:NNn \matrix_gsub:NNn ⟨*destination*⟩ ⟨*source*⟩ {⟨*scalar*⟩}
\matrix_gmul:NNn \matrix_gmul:NNn ⟨*destination*⟩ ⟨*source*⟩ {⟨*scalar*⟩}
\matrix_gdiv:NNn \matrix_gdiv:NNn ⟨*destination*⟩ ⟨*source*⟩ {⟨*scalar*⟩}
\matrix_gpower:NNn \matrix_gpower:NNn ⟨*destination*⟩ ⟨*source*⟩ {⟨*scalar*⟩}

将 ⟨*source*⟩ 的每个元素分别与 ⟨*scalar*⟩ 执行加、减、乘、除或乘方运算，并将结果写入 ⟨*destination*⟩。对于源矩阵 A 和标量 c，对应的逐元素运算依次为 $A+c$、$A-c$、$A\times c$、A/c 和 A^c。⟨*scalar*⟩ 按 l3fp 浮点表达式求值。

1.5.3 矩阵间的运算

本节介绍两个矩阵之间的逐元素加法、逐元素减法和矩阵乘法。用于存储结果的目标矩阵必须是已创建的普通矩阵；作为输入的操作数必须是已创建的矩阵，可以是普通矩阵或常量矩阵。

\matrix_gadd:NNN \matrix_gadd:NNN ⟨*destination*⟩ ⟨*matrix A*⟩ ⟨*matrix B*⟩
\matrix_gsub:NNN \matrix_gsub:NNN ⟨*destination*⟩ ⟨*matrix A*⟩ ⟨*matrix B*⟩

分别计算两个矩阵对应元素的和或差，并将结果写入 ⟨*destination*⟩。两个操作数的行数和列数必须分别相等，否则产生维度不匹配错误。目标矩阵的维度更新为与操作数相同。

\matrix_gadd:NN \matrix_gadd:NN ⟨*matrix A*⟩ ⟨*matrix B*⟩
\matrix_gsub:NN \matrix_gsub:NN ⟨*matrix A*⟩ ⟨*matrix B*⟩

分别执行原地运算 $A\leftarrow A+B$ 和 $A\leftarrow A-B$。两个矩阵的行数和列数必须分别相等，否则产生维度不匹配错误。运算结果写回第一个矩阵，其维度保持不变；第二个矩阵仅作为输入，不会被修改。

\matrix_gmul:NNN \matrix_gmul:NNN ⟨*destination*⟩ ⟨*matrix A*⟩ ⟨*matrix B*⟩

计算两个矩阵的乘积 AB。⟨*matrix A*⟩ 的列数必须等于 ⟨*matrix B*⟩ 的行数，否则产生维度不匹配错误。将结果写入 ⟨*destination*⟩，其维度更新为 ⟨*matrix A*⟩ 的行数乘以 ⟨*matrix B*⟩ 的列数。

\matrix_gmul:NN \matrix_gmul:NN ⟨*matrix A*⟩ ⟨*matrix B*⟩

计算两个矩阵的乘积 AB，并将结果写回第一个矩阵，即执行 $A \leftarrow AB$。乘积的维度由操作数决定，因此第一个矩阵的行数和列数可能发生变化。第二个矩阵仅作为输入，不会被修改。

1.5.4 基于矩阵的映射

映射按行优先顺序访问矩阵中的元素，并在每次访问时将当前元素值传递给回调。映射机制只负责读取和传递元素值，不会自动将回调结果写回矩阵。如果矩阵为空，则不会调用回调。映射可以嵌套使用；内层映射结束后，会恢复外层映射的处理状态。

\matrix_map_function:NN \matrix_map_function:NN ⟨*matrix*⟩ ⟨*function*⟩

对 ⟨*matrix*⟩ 的每个元素调用一次 ⟨*function*⟩。⟨*function*⟩ 必须接收一个参数，该参数为当前元素值。

\matrix_map_inline:Nn \matrix_map_inline:Nn ⟨*matrix*⟩ {⟨*inline code*⟩}

对 ⟨*matrix*⟩ 的每个元素执行一次 ⟨*inline code*⟩。在内联代码中，`##1` 表示当前元素值。例如，下面的代码依次将矩阵元素写入终端：

```
\matrix_map_inline:Nn \l_demo_matrix
  { \iow_term:n { ##1 } }
```

\matrix_map_break: \matrix_map_break:

\matrix_map_break:n \matrix_map_break:n {⟨*code*⟩}

在映射回调中调用 `\matrix_map_break:` 可以提前结束当前映射。`\matrix_map_break:n` 除了结束当前映射，还会在退出映射时执行 ⟨*code*⟩。

1.6 几何变换

本宏包提供基于齐次坐标矩阵的二维和三维几何变换。二维变换使用 3×3 矩阵，三维变换使用 4×4 矩阵。

变换的构建与应用是两个独立阶段：构建阶段负责组合和保存变换矩阵，应用阶段则通过变换函数计算变换后的坐标。两个阶段不会自动同步，即变换矩阵不是立即生效的，而是需要显式调用刷新命令。

变换矩阵采用列向量约定，变换矩阵默认初始化单位变换矩阵（即无变换）。每次添加变换时，当前矩阵在右侧乘以新的变换矩阵。因此，若依次添加 $T_1, T_2, \ldots, T_n$，最终矩阵为 $M = T_1 T_2 \cdots T_n$；位移量、缩放系数、错切系数和角度均按 l3fp 浮点表达式求值。变换矩阵的更新均采用全局状态，不会因普通 TEX 分组结束而恢复。

1.6.1 构建变换

以下命令用于初始化变换矩阵以及添加各种几何变换。所有变换均基于当前变换矩阵进行组合。注意，构建二维变换时，不能使用三维版函数；反之亦然。

\transform_init:n `\transform_init:n {⟨dimension⟩}`

New: 2026-11-21

初始化当前变换矩阵。⟨*dimension*⟩ 指定变换空间的维数，只能为 2 或 3。初始化后，当前变换矩阵为相应维数的单位矩阵，表示尚未添加任何几何变换。

\transform_save:n `\transform_save:n {⟨name⟩}`
\transform_restore:n `\transform_restore:n {⟨name⟩}`

New: 2026-11-21

保存或恢复当前变换矩阵。

\transform_save:n 将当前变换矩阵保存到指定名称对应的记录中。如果该名称已经存在，则使用当前矩阵更新原有记录。

\transform_restore:n 从指定名称的记录中恢复变换矩阵。如果对应的记录不存在，则不执行任何操作。

\transform_shift:nn `\transform_shift:nn {⟨dx⟩} {⟨dy⟩}`
\transform_shift:nnn `\transform_shift:nnn {⟨dx⟩} {⟨dy⟩} {⟨dz⟩}`

New: 2026-11-21

添加平移变换，分别用于二维和三维空间。偏移量应为纯数字，不能带单位。

\transform_scale:nn `\transform_scale:nn {⟨sx⟩} {⟨sy⟩}`
\transform_scale:nnn `\transform_scale:nnn {⟨sx⟩} {⟨sy⟩} {⟨sz⟩}`
\transform_scale:nnnn `\transform_scale:nnnn {⟨x⟩} {⟨y⟩} {⟨sx⟩} {⟨sy⟩}`
\transform_scale:nnnnnn `\transform_scale:nnnnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨sx⟩} {⟨sy⟩} {⟨sz⟩}`

New: 2026-11-21

添加基于原点或指定点的缩放变换，分别用于二维和三维空间。如需镜像，则将相关的缩放系数设为 -1。

\transform_shear:nn
\transform_shear:nnnn
\transform_shear:nnnnnn
\transform_shear:nnnnnnnnn

New: 2026-11-21

```
\transform_shear:nn {⟨xy coefficient⟩} {⟨yx coefficient⟩}
\transform_shear:nnnn {⟨x⟩} {⟨y⟩} {⟨xy coefficient⟩} {⟨yx coefficient⟩}
\transform_shear:nnnnnn
  {⟨xy coefficient⟩} {⟨xz coefficient⟩}
  {⟨yx coefficient⟩} {⟨yz coefficient⟩}
  {⟨zx coefficient⟩} {⟨zy coefficient⟩}
\transform_shear:nnnnnnnnn {⟨x⟩} {⟨y⟩} {⟨z⟩}
  {⟨xy coefficient⟩} {⟨xz coefficient⟩}
  {⟨yx coefficient⟩} {⟨yz coefficient⟩}
  {⟨zx coefficient⟩} {⟨zy coefficient⟩}
```

添加基于原点或指定点的错切变换，分别用于二维和三维空间。

\transform_rotate:n
\transform_rotated:n
\transform_rotate:nnn
\transform_rotated:nnn

New: 2026-11-21

```
\transform_rotate:n  {⟨angle in radians⟩}
\transform_rotated:n {⟨angle in degrees⟩}
\transform_rotate:nnn  {⟨x⟩} {⟨y⟩} {⟨angle in radians⟩}
\transform_rotated:nnn {⟨x⟩} {⟨y⟩} {⟨angle in degrees⟩}
```

添加二维旋转变换，以坐标原点或指定点为旋转中心。正角度为逆时针方向。

\transform_rotate_x:n
\transform_rotate_y:n
\transform_rotate_z:n
\transform_rotated_x:n
\transform_rotated_y:n
\transform_rotated_z:n
\transform_rotate_x:nnnn
\transform_rotate_y:nnnn
\transform_rotate_z:nnnn
\transform_rotated_x:nnnn
\transform_rotated_y:nnnn
\transform_rotated_z:nnnn

New: 2026-11-21

```
\transform_rotate_x:n  {⟨angle in radians⟩}
\transform_rotate_y:n  {⟨angle in radians⟩}
\transform_rotate_z:n  {⟨angle in radians⟩}
\transform_rotated_x:n {⟨angle in degrees⟩}
\transform_rotated_y:n {⟨angle in degrees⟩}
\transform_rotated_z:n {⟨angle in degrees⟩}
\transform_rotate_x:nnnn  {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in radians⟩}
\transform_rotate_y:nnnn  {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in radians⟩}
\transform_rotate_z:nnnn  {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in radians⟩}
\transform_rotated_x:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in degrees⟩}
\transform_rotated_y:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in degrees⟩}
\transform_rotated_z:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in degrees⟩}
```

添加三维旋转变换，分别绕 x、y 和 z 轴旋转，旋转中心为坐标原点或指定点。正角度的旋转方向遵循右手定则。

1.6.2 刷新变换

构建变换时，添加变换的命令只更新当前变换矩阵，不会自动更新用于计算坐标的变换函数。因此，在构建完成后，需要显式刷新变换函数。

刷新采用局部函数定义。如果在 TEX 分组内执行刷新，分组结束后变换函数会恢复为分组开始前的定义。

\transform_refresh: \transform_refresh:
\transform_refresh:N \transform_refresh:N ⟨*matrix*⟩

New: 2026-11-21

根据指定的变换矩阵刷新对应的变换函数。指定矩阵必须为合法的齐次变换矩阵：二维变换使用 3×3 矩阵，三维变换使用 4×4 矩阵，且矩阵右下角的元素必须为 1。

1.6.3 应用变换

与变换矩阵不一样，本模块始终保存两套变换函数：分别用于二维变换与三维变换。即，使用都可以同时进行二维与三维变换。

\transform_apply:nnn ⋆ \transform_apply:nnn {⟨*x*⟩} {⟨*y*⟩} {⟨*suffix*⟩}
\transform_apply:nnnn ⋆ \transform_apply:nnnn {⟨*x*⟩} {⟨*y*⟩} {⟨*z*⟩} {⟨*suffix*⟩}

New: 2026-11-21

对给定坐标应用对应的变换，并输出对应结果。坐标应为不带单位的纯数字。各结果分量依次输出并以逗号分隔，且不含齐次分量。⟨*suffix*⟩ 会追加到每个变换后的坐标分量之后，如用于添加单位。如果不需要追加内容，可以将其留空。

\transform_apply:nn ⋆ \transform_apply:nn {⟨*coordinates*⟩} {⟨*suffix*⟩}

New: 2026-11-21

对给定坐标应用对应的变换，并输出对应结果。⟨*coordinates*⟩ 是由逗号分隔的坐标列表，包含两个或三个坐标分量，分别用于二维和三维变换。坐标应为不带单位的纯数字。各结果分量依次输出并以逗号分隔，且不含齐次分量。⟨*suffix*⟩ 会追加到每个变换后的坐标分量之后，如用于添加单位。如果不需要追加内容，可以将其留空。

\transform_apply_inline:nnn \transform_apply_inline:nnn {⟨*x*⟩} {⟨*y*⟩} {⟨*code*⟩}
\transform_apply_inline:nnnn \transform_apply_inline:nnnn {⟨*x*⟩} {⟨*y*⟩} {⟨*z*⟩} {⟨*code*⟩}

New: 2026-11-23

对给定坐标应用当前变换，并把变换后的坐标分量传给 ⟨*code*⟩ 执行。在 ⟨*code*⟩ 中，可用 `#1`、`#2` 以及（仅三维版）`#3` 引用变换后的坐标分量。坐标应为不带单位的纯数字。本函数不能递归调用（即不得在其自身的 ⟨*code*⟩ 参数中再次调用本函数）。

1.7 LaTeX2e 命令

本小节介绍采用 LaTeX2e 风格的接口命令。

1.7.1 初始化与保存恢复

\TransformInit \TransformInit [⟨*dimension*⟩]

New: 2026-11-21

初始化当前变换矩阵为单位矩阵。⟨*dimension*⟩ 指定变换维度，可取 `2` 或 `3`，省略时默认为 `2`。二维变换使用 3×3 单位矩阵，三维变换使用 4×4 单位矩阵。每次开始一组新的变换时，应先调用本命令初始化变换状态。

\TransformSave \TransformSave {⟨name⟩}
\TransformRestore \TransformRestore {⟨name⟩}
New: 2026-11-21
\TransformSave 将当前变换矩阵以 ⟨name⟩ 为名称保存；\TransformRestore 则恢复此前以相同名称保存的变换矩阵。名称用于区分不同的保存记录。若指定名称尚无保存记录，\TransformRestore 不执行恢复操作。

1.7.2 平移、缩放与错切

\TransformShift \TransformShift {⟨dx, dy⟩}
\TransformShift {⟨dx, dy, dz⟩}
New: 2026-11-21
添加平移变换。二维形式使用水平和垂直方向的位移量 dx、dy；三维形式还包含深度方向的位移量 dz。参数数量必须与当前变换维度相对应。

\TransformScale \TransformScale [⟨center⟩] {⟨scale⟩}
\TransformScale [⟨center⟩] {⟨sx, sy⟩}
\TransformScale [⟨center⟩] {⟨sx, sy, sz⟩}
New: 2026-11-21
添加缩放变换。一个缩放系数表示各方向使用相同的缩放比例；两个系数分别指定二维坐标轴的缩放比例，三个系数分别指定三维坐标轴的缩放比例。可选参数 ⟨center⟩ 用于指定缩放中心，二维形式为 x,y，三维形式为 x,y,z；省略时以原点为中心。参数数量应与当前变换维度相匹配。

\TransformShear \TransformShear [⟨center⟩] {⟨a, b⟩}
\TransformShear [⟨center⟩] {⟨a, b, c, d, e, f⟩}
New: 2026-11-21
添加错切变换。二维形式使用两个系数，对应变换矩阵 $\begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$；三维形式使用六个系数，对应变换矩阵 $\begin{bmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$。可选参数 ⟨center⟩ 用于指定错切中心，格式与 \TransformScale 中相同；省略时以原点为中心。参数数量应与当前变换维度相匹配。

1.7.3 旋转变换

\TransformRotate \TransformRotate [⟨center⟩] {⟨angle in radians⟩}
\TransformRotated \TransformRotated [⟨center⟩] {⟨angle in degrees⟩}
New: 2026-11-21
添加二维平面旋转，要求当前变换为二维变换。可选参数 ⟨center⟩ 指定旋转中心，格式为 x, y；省略时绕原点旋转。

\TransformRotateX \TransformRotateX [⟨center⟩] {⟨angle in radians⟩}
\TransformRotateY \TransformRotateY [⟨center⟩] {⟨angle in radians⟩}
\TransformRotateZ \TransformRotateZ [⟨center⟩] {⟨angle in radians⟩}
\TransformRotatedX \TransformRotatedX [⟨center⟩] {⟨angle in degrees⟩}
\TransformRotatedY \TransformRotatedY [⟨center⟩] {⟨angle in degrees⟩}
\TransformRotatedZ \TransformRotatedZ [⟨center⟩] {⟨angle in degrees⟩}

New: 2026-11-21

添加三维旋转，分别绕 x、y、z 轴旋转，要求当前变换为三维变换。可选参数 ⟨*center*⟩ 指定旋转中心，格式为 `x, y, z`；省略时绕原点旋转。

1.7.4 刷新与应用变换

变换命令只更新当前变换矩阵。要将矩阵系数转换为可调用的坐标变换函数，需要执行刷新操作；此后再调用应用命令计算坐标。

\TransformRefresh \TransformRefresh
\TransformRefresh [⟨matrix⟩]

New: 2026-11-21

根据变换矩阵刷新内部坐标变换函数。省略可选参数时，使用当前变换矩阵；指定 ⟨*matrix*⟩ 时，则从该矩阵提取变换函数。二维变换矩阵必须为 3×3，三维变换矩阵必须为 4×4；两种形式的右下角元素均须为 1，否则报告无效变换矩阵错误。

\TransformApply \TransformApply [⟨suffix⟩] {⟨coordinates⟩}

New: 2026-11-21

对指定坐标应用对应的变换，并输出对应结果。⟨*coordinates*⟩ 是由逗号分隔的坐标列表，包含两个或三个坐标分量，分别用于二维和三维变换。坐标分量应为纯数字，不能带单位。各结果分量依次输出并以逗号分隔，且不含齐次分量。⟨*suffix*⟩ 会追加到每个变换后的坐标分量之后，如用于添加单位。如果不需要追加内容，可以将其留空。

1.8 错误条件

当前实现会在以下情况下报告错误：

- 创建矩阵时，行数或列数为负数；
- 创建矩阵时，矩阵变量已经定义；
- 将常量矩阵用作修改操作的目标矩阵；
- 创建或设置矩阵时，元素列表中的项数超过矩阵的元素总数；
- 执行矩阵间的逐元素二元运算时，两个操作数的维度不一致；
- 执行矩阵乘法时，左操作数的列数与右操作数的行数不一致；

- 几何变换命令的参数数量与二维或三维形式不匹配；
- 刷新变换时，输入矩阵不是符合要求的 3×3 或 4×4 变换矩阵，或其右下角元素不为 1。

2 User Guide [中文]

2.1 Package Overview

The lt3matrix package provides a lightweight matrix type, intended mainly for matrix demonstrations, numerical examples, and geometric transformations in TEX documents. Users can create matrices, set matrix entries, perform matrix operations, iterate over entries, and typeset matrices in mathematical formulas.

Matrices in this package have a fixed number of rows and columns. Normally, the dimensions of a matrix should not be changed after it is created, and the package does not provide an interface for directly modifying matrix dimensions; however, when a matrix is the result of an operation, its dimensions are updated automatically according to the result. Matrix entries can be modified via assignment commands. Both the creation and modification of matrices are global operations: if a modification command is executed inside a TEX group, the matrix contents will not be restored automatically after the group ends.

An empty matrix also has a definite number of rows and columns, but contains no entries. The number of rows and columns cannot be negative.

For geometric transformations, users may build a complete transformation by composing basic transformations. Although the internal implementation uses homogeneous transformation matrices, users need no knowledge of them whatsoever.

When applying a transformation, a transformation function is defined dynamically from the transformation matrix, and this function is then used to compute transformed coordinates quickly. Each transformation execution only performs the corresponding arithmetic operations (carried out by the fp module), without repeatedly retrieving data from the matrix, thereby greatly improving computational speed.

2.2 Quick Start

The following example creates a 2×3 matrix, assigns values to its elements, and typesets the matrix in mathematical mode:

```
\ExplSyntaxOn
\matrix_new:Nnn \l_demo_matrix { 2 } { 3 }
\matrix_gset:Nn \l_demo_matrix { 1, 2, 3, 4, 5, 6 }
\[
  \matrix_use:N \l_demo_matrix
```

```
\]
\ExplSyntaxOff
```

The output is:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

2.3 Command and Argument Conventions

This package is primarily intended as a basic extension to expl3. Its main interfaces follow the expl3 naming conventions and should be used with `\ExplSyntaxOn` enabled. In addition, the package provides corresponding LaTeX2e-style commands for commonly used operations, such as geometric transformations, making them accessible to LaTeX users who are not familiar with expl3.

In expl3-style command names, the letters following the colon indicate the forms of the arguments: `N` accepts a single control sequence, `n` accepts a general argument passed as is (usually enclosed in braces), `c` constructs a control sequence from a character-based name, and `e` expands the corresponding argument before passing the result to the command. Thus, the `N` form takes a matrix variable directly, as in `\matrix_new:Nnn`, whereas the `c` form takes the name of a matrix variable (without the leading backslash), as in `\matrix_new:cnn {l_demo_matrix} {2} {2}`. Different variants of the same command perform the same operation; they differ only in how the corresponding arguments are passed.

In this document, ⟨*matrix*⟩ denotes a matrix variable; ⟨*destination*⟩ denotes the target matrix that receives a result; and ⟨*source*⟩ denotes an input matrix. Except for creation commands, any matrix passed as an argument must have been created beforehand. ⟨*comma list*⟩ denotes a comma-separated list of elements.

Matrix modification commands beginning with `g` use global assignments. Changes to a matrix are not reverted when an ordinary TeX group ends.

2.4 Basic Commands

2.4.1 Creating Matrices

\matrix_new:N
\matrix_new:Nnn
\matrix_new:cnn

\matrix_new:N ⟨*matrix*⟩
\matrix_new:Nnn ⟨*matrix*⟩ {⟨*rows*⟩} {⟨*columns*⟩}
Creates a regular matrix with all elements initialized to zero.

Updated: 2026-11-18

If no dimensions are specified, an empty 0×0 matrix is created. Otherwise, ⟨*rows*⟩ and ⟨*columns*⟩ specify the number of rows and columns, respectively. Both dimension arguments are evaluated as integer expressions and must be non-negative. Zero dimensions are allowed.

An empty 0×0 matrix is typically used to store the result of a matrix operation. An error is raised if the matrix variable is already defined.

\matrix_const:Nnnn
\matrix_const:cnnn

\matrix_const:Nnnn ⟨*matrix*⟩ {⟨*rows*⟩} {⟨*columns*⟩} {⟨*comma list*⟩}
Creates a constant matrix and initializes its elements from a comma-separated list. Elements are filled in row-major order; any elements not specified in the list are initialized to zero. A constant matrix cannot be modified using the matrix modification commands provided by this package.

⟨*matrix*⟩, ⟨*rows*⟩, and ⟨*columns*⟩ have the same meanings as for `\matrix_new:Nnn`. ⟨*comma list*⟩ is the list of matrix elements, with each item evaluated as a floating-point expression. The number of items must not exceed the total number of elements in the matrix. If it does, an error is raised and the matrix is not created.

2.4.2 Setting and Modifying Matrices

The following commands modify the contents of an existing matrix. The destination matrix must be a regular matrix; an error is raised if it is a constant matrix. All modifications are global.

\matrix_gzero:N
\matrix_gzero:c

\matrix_gzero:N ⟨*matrix*⟩
Sets all elements of ⟨*matrix*⟩ to zero, while preserving its dimensions (number of rows and columns).

\matrix_gset:Nn
\matrix_gset:(cn|Ne|ce)
Updated: 2026-11-19

\matrix_gset:Nn ⟨*matrix*⟩ {⟨*comma list*⟩}

Sets all elements of ⟨***matrix***⟩ using a comma-separated list, with the elements filled in row-major order. Each item in the list is evaluated as a floating-point expression. The number of items must not exceed the total number of elements in the matrix. If fewer items are provided, the remaining elements are set to zero. If too many items are provided, an error is raised and the assignment is not performed. The dimensions of the matrix remain unchanged.

\matrix_gset_eq:NN
\matrix_gset_eq:(cN|Nc)

\matrix_gset_eq:NN ⟨*destination*⟩ ⟨*source*⟩

Copies the dimensions and elements of ⟨***source***⟩ to ⟨***destination***⟩. The dimensions of the destination matrix are updated accordingly. After copying, the two matrices have independent data; subsequent modifications to either matrix do not affect the other.

⟨***destination***⟩ must be an existing regular matrix; ⟨***source***⟩ must be an existing matrix and may be either a regular matrix or a constant matrix. The source matrix is not modified.

2.4.3 Displaying and Inspecting Matrices

\matrix_use:N
\matrix_use:c

\matrix_use:N ⟨*matrix*⟩

Typesets ⟨***matrix***⟩ in mathematical mode. Non-empty matrices are displayed as parenthesized matrices, while empty matrices display their dimensions, such as 2×0. This command does not enter mathematical mode automatically and must therefore be used in mathematical mode.

\matrix_log:N
\matrix_log:c

\matrix_log:N ⟨*matrix*⟩

Writes the dimensions, storage information, and element contents of ⟨***matrix***⟩ to the compilation log for inspection and debugging. This command does not typeset the matrix in the document body.

2.4.4 Predefined Matrices

\c_empty_matrix

The empty 0×0 matrix.

\c_identity_two_matrix
\c_identity_three_matrix
\c_identity_four_matrix

The 2×2, 3×3, and 4×4 identity matrices, respectively.

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

\g_tmpa_matrix
\g_tmpb_matrix

Scratch matrix variables for global assignments. These variables are not used by the package's internal code and can therefore be safely used with commands provided by this package. However, they may be overwritten by other code and should only be used for short-term storage.

2.5 Conditionals, Mapping, and Operations

2.5.1 Matrix Conditionals

\matrix_if_empty_p:N ⋆ \matrix_if_empty:NTF ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}
\matrix_if_empty:N*TF* ⋆

New: 2026-11-19

Tests whether ⟨*matrix*⟩ is empty. The condition is true if the matrix contains no elements, regardless of its number of rows and columns. Thus, matrices of dimensions 0×0, 2×0, and 0×3 are all considered empty.

\matrix_if_square_p:N ⋆ \matrix_if_square:NTF ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}
\matrix_if_square:N*TF* ⋆

New: 2026-11-19

Tests whether ⟨*matrix*⟩ is square, that is, whether its number of rows equals its number of columns. Empty matrices are also checked according to their dimensions. For example, a 0×0 matrix is square, whereas a 2×0 matrix is not.

\matrix_if_same_dim_p:NN ⋆ \matrix_if_same_dim:NNTF ⟨*matrix1*⟩ ⟨*matrix2*⟩
\matrix_if_same_dim:NN*TF* ⋆ {⟨*true code*⟩} {⟨*false code*⟩}

New: 2026-11-19

Tests whether the two matrices have the same dimensions, that is, whether their respective numbers of rows and columns are equal. The matrix elements do not affect the result.

\matrix_if_same_dim_p:Nnn ⋆ \matrix_if_same_dim:NnnTF ⟨*matrix*⟩ {⟨*rows*⟩} {⟨*columns*⟩}
\matrix_if_same_dim:Nnn*TF* ⋆ {⟨*true code*⟩} {⟨*false code*⟩}

New: 2026-11-19

Tests whether ⟨*matrix*⟩ has the specified dimensions. ⟨*rows*⟩ and ⟨*columns*⟩ specify the number of rows and columns, respectively, and are evaluated as integer expressions.

\matrix_if_zero:N*TF* `\matrix_if_zero:NTF` ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

New: 2026-11-19 Tests whether ⟨*matrix*⟩ is a zero matrix, that is, whether all its elements are zero. Floating-point comparisons use a tolerance of 10^{-12}. An empty matrix is not considered a zero matrix, so this condition is false for an empty matrix.

TEX hackers note: This command uses internal scratch variables. If unexpected behavior occurs when calling this test within a mapping, try enclosing the call in a TEX group to isolate the scratch variables.

\matrix_if_identity:N*TF* `\matrix_if_identity:NTF` ⟨*matrix*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

New: 2026-11-19 Tests whether ⟨*matrix*⟩ is an identity matrix. An identity matrix must be square, with all elements on the main diagonal equal to 1 and all other elements equal to 0. Floating-point comparisons use a tolerance of 10^{-12}. An empty matrix is not considered an identity matrix, since it contains no element equal to 1.

TEX hackers note: This command uses internal scratch variables. If unexpected behavior occurs when calling this test within a mapping, try enclosing the call in a TEX group to isolate the scratch variables.

2.5.2 Element-wise Operations

This section describes commands that perform the same operation on each element of a matrix. Depending on how the results are stored, the commands are provided in two forms: one writes the results back to the original matrix, while the other writes them to a separate destination matrix. The matrix used to store the results must be an existing regular matrix.

The form that writes the results back to the original matrix preserves its dimensions, whereas the form that writes the results to a destination matrix updates the destination dimensions to match those of the source matrix. If the destination and source are the same matrix, the results are still written back to that matrix. However, compared with the form that writes directly to the original matrix, this requires additional temporary storage allocation and deallocation. Therefore, the in-place form is recommended in this case.

\matrix_gnegate:N `\matrix_gnegate:N` ⟨*matrix*⟩
\matrix_gabs:N `\matrix_gabs:N` ⟨*matrix*⟩
\matrix_gsign:N `\matrix_gsign:N` ⟨*matrix*⟩

Replaces each element of ⟨*matrix*⟩ with its negation, absolute value, or sign, respectively, and writes the results back to the same matrix.

\matrix_ground:Nn `\matrix_ground:Nn ⟨matrix⟩ {⟨digits⟩}`

Rounds each element of ⟨*matrix*⟩ and writes the results back to the same matrix. ⟨*digits*⟩ is the precision argument passed to the `round` operation of l3fp.

\matrix_gnegate:NN `\matrix_gnegate:NN ⟨destination⟩ ⟨source⟩`
\matrix_gabs:NN `\matrix_gabs:NN ⟨destination⟩ ⟨source⟩`
\matrix_gsign:NN `\matrix_gsign:NN ⟨destination⟩ ⟨source⟩`

Applies the corresponding operation to each element of ⟨*source*⟩ and writes the results to ⟨*destination*⟩. The dimensions of the destination matrix are updated to match those of the source matrix.

\matrix_ground:NNn `\matrix_ground:NNn ⟨destination⟩ ⟨source⟩ {⟨digits⟩}`

Rounds each element of ⟨*source*⟩ and writes the results to ⟨*destination*⟩. The meaning of ⟨*digits*⟩ is the same as for `\matrix_ground:Nn`.

\matrix_gadd:Nn `\matrix_gadd:Nn ⟨matrix⟩ {⟨scalar⟩}`
\matrix_gsub:Nn `\matrix_gsub:Nn ⟨matrix⟩ {⟨scalar⟩}`
\matrix_gmul:Nn `\matrix_gmul:Nn ⟨matrix⟩ {⟨scalar⟩}`
\matrix_gdiv:Nn `\matrix_gdiv:Nn ⟨matrix⟩ {⟨scalar⟩}`
\matrix_gpower:Nn `\matrix_gpower:Nn ⟨matrix⟩ {⟨scalar⟩}`

Performs addition, subtraction, multiplication, division, or exponentiation, respectively, between each element of ⟨*matrix*⟩ and ⟨*scalar*⟩, and writes the results back to the same matrix. For a matrix A and a scalar c, the corresponding element-wise operations are $A + c$, $A - c$, $A \times c$, A/c, and A^c. ⟨*scalar*⟩ is evaluated as a floating-point expression by l3fp.

\matrix_gadd:NNn `\matrix_gadd:NNn ⟨destination⟩ ⟨source⟩ {⟨scalar⟩}`
\matrix_gsub:NNn `\matrix_gsub:NNn ⟨destination⟩ ⟨source⟩ {⟨scalar⟩}`
\matrix_gmul:NNn `\matrix_gmul:NNn ⟨destination⟩ ⟨source⟩ {⟨scalar⟩}`
\matrix_gdiv:NNn `\matrix_gdiv:NNn ⟨destination⟩ ⟨source⟩ {⟨scalar⟩}`
\matrix_gpower:NNn `\matrix_gpower:NNn ⟨destination⟩ ⟨source⟩ {⟨scalar⟩}`

Performs addition, subtraction, multiplication, division, or exponentiation, respectively, between each element of ⟨*source*⟩ and ⟨*scalar*⟩, and writes the results to ⟨*destination*⟩. For a source matrix A and a scalar c, the corresponding element-wise operations are $A + c$, $A - c$, $A \times c$, A/c, and A^c. ⟨*scalar*⟩ is evaluated as a floating-point expression by l3fp.

2.5.3 Operations Between Matrices

This section describes element-wise addition, element-wise subtraction, and matrix multiplication between two matrices. The destination matrix used to store the result must be an existing regular matrix; the input operands must be existing matrices and may be either regular or constant matrices.

`\matrix_gadd:NNN` `\matrix_gadd:NNN` ⟨*destination*⟩ ⟨*matrix A*⟩ ⟨*matrix B*⟩
`\matrix_gsub:NNN` `\matrix_gsub:NNN` ⟨*destination*⟩ ⟨*matrix A*⟩ ⟨*matrix B*⟩

Computes the element-wise sum or difference of two matrices, respectively, and writes the result to ⟨*destination*⟩. The two operands must have the same number of rows and the same number of columns; otherwise, a dimension mismatch error is raised. The dimensions of the destination matrix are updated to match those of the operands.

`\matrix_gadd:NN` `\matrix_gadd:NN` ⟨*matrix A*⟩ ⟨*matrix B*⟩
`\matrix_gsub:NN` `\matrix_gsub:NN` ⟨*matrix A*⟩ ⟨*matrix B*⟩

Performs the in-place operation $A \leftarrow A + B$ or $A \leftarrow A - B$, respectively. The two matrices must have the same number of rows and the same number of columns; otherwise, a dimension mismatch error is raised. The result is written back to the first matrix, whose dimensions remain unchanged. The second matrix is used only as input and is not modified.

`\matrix_gmul:NNN` `\matrix_gmul:NNN` ⟨*destination*⟩ ⟨*matrix A*⟩ ⟨*matrix B*⟩

Computes the matrix product AB. The number of columns in ⟨*matrix A*⟩ must equal the number of rows in ⟨*matrix B*⟩; otherwise, a dimension mismatch error is raised. The result is written to ⟨*destination*⟩, whose dimensions are updated to the number of rows in ⟨*matrix A*⟩ by the number of columns in ⟨*matrix B*⟩.

`\matrix_gmul:NN` `\matrix_gmul:NN` ⟨*matrix A*⟩ ⟨*matrix B*⟩

Computes the matrix product AB and writes the result back to the first matrix, performing the operation $A \leftarrow AB$. The dimensions of the first matrix may therefore change according to the product. The second matrix is used only as input and is not modified.

2.5.4 Mapping over Matrices

Mapping visits the elements of a matrix in row-major order and passes the value of the current element to the callback on each iteration. The mapping mechanism

only reads and passes element values; it does not automatically write callback results back to the matrix. If the matrix is empty, the callback is not invoked. Mapping can be nested; when an inner mapping ends, the processing state of the outer mapping is restored.

\matrix_map_function:NN `\matrix_map_function:NN ⟨matrix⟩ ⟨function⟩`

Calls ⟨*function*⟩ once for each element of ⟨*matrix*⟩. ⟨*function*⟩ must accept one argument, which is the value of the current element.

\matrix_map_inline:Nn `\matrix_map_inline:Nn ⟨matrix⟩ {⟨inline code⟩}`

Executes ⟨*inline code*⟩ once for each element of ⟨*matrix*⟩. Within the inline code, `##1` represents the value of the current element. For example, the following code writes the matrix elements to the terminal in sequence:

```
\matrix_map_inline:Nn \l_demo_matrix
  { \iow_term:n { ##1 } }
```

\matrix_map_break: `\matrix_map_break:`
\matrix_map_break:n `\matrix_map_break:n {⟨code⟩}`

Calling `\matrix_map_break:` within a mapping callback terminates the current mapping early. In addition to terminating the current mapping, `\matrix_map_break:n` executes ⟨*code*⟩ when the mapping exits.

2.6 Geometric Transformations

This package provides two-dimensional and three-dimensional geometric transformations based on homogeneous transformation matrices. Two-dimensional transformations use 3×3 matrices, while three-dimensional transformations use 4×4 matrices.

The construction and application of transformations are two independent stages. The construction stage builds and combines transformation matrices, while the application stage uses transformation functions to calculate transformed coordinates. These two stages are not synchronized automatically: changes to the transformation matrix do not take effect until the transformation functions are explicitly refreshed.

Transformation matrices use the column-vector convention. The transformation matrix is initialized to the identity matrix, representing no transformation. Each newly added transformation matrix is multiplied on the left of the current matrix.

Thus, if transformations $T_1, T_2, \ldots, T_n$ are added in that order, the resulting matrix is

$$M = T_n T_{n-1} \cdots T_2 T_1.$$

When this composite matrix is applied to a coordinate column vector, the transformations are performed in the order in which they were added. Translation offsets, scaling factors, shear coefficients, and angles are all evaluated as floating-point expressions by l3fp. Updates to transformation matrices are global and are not reverted when an ordinary TEX group ends.

2.6.1 Building Transformations

The following commands initialize the transformation matrix and add various geometric transformations. Each transformation is composed with the current transformation matrix. The two-dimensional and three-dimensional versions of a command must be used with the corresponding transformation dimension.

`\transform_init:n` `\transform_init:n {⟨dimension⟩}`

New: 2026-11-21

Initializes the current transformation matrix. ⟨*dimension*⟩ specifies the dimension of the transformation space and must be either 2 or 3. After initialization, the current transformation matrix is set to the identity matrix of the corresponding dimension, indicating that no geometric transformation has been added.

`\transform_save:n` `\transform_save:n {⟨name⟩}`
`\transform_restore:n` `\transform_restore:n {⟨name⟩}`

New: 2026-11-21

Saves or restores the current transformation matrix.

`\transform_save:n` saves the current transformation matrix under the specified name. If a record with that name already exists, it is updated with the current matrix.

`\transform_restore:n` restores the transformation matrix previously saved under the specified name. If no record exists for that name, no restoration is performed.

`\transform_shift:nn` `\transform_shift:nn {⟨dx⟩} {⟨dy⟩}`
`\transform_shift:nnn` `\transform_shift:nnn {⟨dx⟩} {⟨dy⟩} {⟨dz⟩}`

New: 2026-11-21

Adds a translation, for two-dimensional and three-dimensional spaces, respectively. The offsets must be numerical values without units.

\transform_scale:nn `\transform_scale:nn {⟨sx⟩} {⟨sy⟩}`
\transform_scale:nnn `\transform_scale:nnn {⟨sx⟩} {⟨sy⟩} {⟨sz⟩}`
\transform_scale:nnnn `\transform_scale:nnnn {⟨x⟩} {⟨y⟩} {⟨sx⟩} {⟨sy⟩}`
\transform_scale:nnnnnn `\transform_scale:nnnnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨sx⟩} {⟨sy⟩} {⟨sz⟩}`

New: 2026-11-21

Adds a scaling transformation about the origin or a specified point, for two-dimensional and three-dimensional spaces, respectively. To perform a reflection, set the corresponding scaling factor to -1.

\transform_shear:nn `\transform_shear:nn {⟨xy coefficient⟩} {⟨yx coefficient⟩}`
\transform_shear:nnnn `\transform_shear:nnnn {⟨x⟩} {⟨y⟩} {⟨xy coefficient⟩} {⟨yx coefficient⟩}`
\transform_shear:nnnnnn `\transform_shear:nnnnnn`
\transform_shear:nnnnnnnnn `{⟨xy coefficient⟩} {⟨xz coefficient⟩}`
`{⟨yx coefficient⟩} {⟨yz coefficient⟩}`
`{⟨zx coefficient⟩} {⟨zy coefficient⟩}`
`\transform_shear:nnnnnnnnn {⟨x⟩} {⟨y⟩} {⟨z⟩}`
`{⟨xy coefficient⟩} {⟨xz coefficient⟩}`
`{⟨yx coefficient⟩} {⟨yz coefficient⟩}`
`{⟨zx coefficient⟩} {⟨zy coefficient⟩}`

New: 2026-11-21

Adds a shear transformation about the origin or a specified point, for two-dimensional and three-dimensional spaces, respectively.

\transform_rotate:n `\transform_rotate:n {⟨angle in radians⟩}`
\transform_rotated:n `\transform_rotated:n {⟨angle in degrees⟩}`
\transform_rotate:nnn `\transform_rotate:nnn {⟨x⟩} {⟨y⟩} {⟨angle in radians⟩}`
\transform_rotated:nnn `\transform_rotated:nnn {⟨x⟩} {⟨y⟩} {⟨angle in degrees⟩}`

New: 2026-11-21

Adds a two-dimensional rotation about the origin or a specified point. Positive angles indicate counterclockwise rotation.

\transform_rotate_x:n `\transform_rotate_x:n {⟨angle in radians⟩}`
\transform_rotate_y:n `\transform_rotate_y:n {⟨angle in radians⟩}`
\transform_rotate_z:n `\transform_rotate_z:n {⟨angle in radians⟩}`
\transform_rotated_x:n `\transform_rotated_x:n {⟨angle in degrees⟩}`
\transform_rotated_y:n `\transform_rotated_y:n {⟨angle in degrees⟩}`
\transform_rotated_z:n `\transform_rotated_z:n {⟨angle in degrees⟩}`
\transform_rotate_x:nnnn `\transform_rotate_x:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in radians⟩}`
\transform_rotate_y:nnnn `\transform_rotate_y:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in radians⟩}`
\transform_rotate_z:nnnn `\transform_rotate_z:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in radians⟩}`
\transform_rotated_x:nnnn `\transform_rotated_x:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in degrees⟩}`
\transform_rotated_y:nnnn `\transform_rotated_y:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in degrees⟩}`
\transform_rotated_z:nnnn `\transform_rotated_z:nnnn {⟨x⟩} {⟨y⟩} {⟨z⟩} {⟨angle in degrees⟩}`

New: 2026-11-21

Adds a three-dimensional rotation about the x, y, or z axis. The rotation center is either the origin or a specified point. Positive angles follow the right-hand rule.

2.6.2 Refreshing Transformations

When building transformations, the transformation commands only update the current transformation matrix. They do not automatically update the transformation functions used to calculate coordinates. The transformation functions must therefore be refreshed explicitly after the matrix has been constructed.

Refreshing uses local function definitions. If a refresh is performed within a TEX group, the transformation functions revert to their definitions from before the group when the group ends.

`\transform_refresh:` `\transform_refresh:`
`\transform_refresh:N` `\transform_refresh:N` ⟨*matrix*⟩

New: 2026-11-21

Refreshes the corresponding transformation functions using the specified transformation matrix. The specified matrix must be a valid homogeneous transformation matrix: a 3×3 matrix for a two-dimensional transformation or a 4×4 matrix for a three-dimensional transformation. In either case, the bottom-right element must be equal to 1.

2.6.3 Applying Transformations

Unlike the transformation matrix, this module always maintains two sets of transformation functions: one for two-dimensional transformations and one for three-dimensional transformations. Therefore, two-dimensional and three-dimensional transformations can be used independently.

`\transform_apply:nnn` ⋆ `\transform_apply:nnn {`⟨*x*⟩`} {`⟨*y*⟩`} {`⟨*suffix*⟩`}`
`\transform_apply:nnnn` ⋆ `\transform_apply:nnnn {`⟨*x*⟩`} {`⟨*y*⟩`} {`⟨*z*⟩`} {`⟨*suffix*⟩`}`

New: 2026-11-21

Applies the corresponding transformation to the specified coordinates and outputs the result. The coordinates should be plain numbers without units. The resulting components are output in order, separated by commas, and do not include the homogeneous component. ⟨*suffix*⟩ is appended to each transformed coordinate component, for example, to add a unit. It may be left empty when no additional content is required.

\transform_apply:nn ⋆ \transform_apply:nn {⟨*coordinates*⟩} {⟨*suffix*⟩}

New: 2026-11-21

Applies the corresponding transformation to the specified coordinates and outputs the result. ⟨*coordinates*⟩ is a comma-separated list of coordinate components, containing either two or three components for two-dimensional or three-dimensional transformations, respectively. The coordinates should be plain numbers without units. The resulting components are output in order, separated by commas, and do not include the homogeneous component. ⟨*suffix*⟩ is appended to each transformed coordinate component, for example, to add a unit. It may be left empty when no additional content is required.

\transform_apply_inline:nnn \transform_apply_inline:nnn {⟨*x*⟩} {⟨*y*⟩} {⟨*code*⟩}
\transform_apply_inline:nnnn \transform_apply_inline:nnnn {⟨*x*⟩} {⟨*y*⟩} {⟨*z*⟩} {⟨*code*⟩}

New: 2026-11-23

Applies the current transformation to the given coordinates and passes the transformed coordinate components to ⟨*code*⟩ for execution. Within ⟨*code*⟩, the transformed coordinate components can be referenced as `#1`, `#2`, and (for the three-dimensional variant only) `#3`. The coordinates should be plain numbers without units. This function cannot be called recursively (i.e., it must not be invoked from within its own ⟨*code*⟩ argument).

2.7 LaTeX2e Commands

This section describes the LaTeX2e-style command interfaces.

2.7.1 Initialization and Saving or Restoring

\TransformInit \TransformInit [⟨*dimension*⟩]

New: 2026-11-21

Initializes the current transformation matrix to the identity matrix. ⟨*dimension*⟩ specifies the transformation dimension and must be either `2` or `3`. If omitted, it defaults to `2`. Two-dimensional transformations use a 3×3 identity matrix, while three-dimensional transformations use a 4×4 identity matrix. This command should be called to initialize the transformation state at the beginning of each new sequence of transformations.

\TransformSave `\TransformSave {⟨name⟩}`
\TransformRestore `\TransformRestore {⟨name⟩}`

New: 2026-11-21

`\TransformSave` saves the current transformation matrix under the specified ⟨*name*⟩, while `\TransformRestore` restores the transformation matrix previously saved under the same name. The name distinguishes different saved records. If no record exists for the specified name, `\TransformRestore` performs no restoration.

2.7.2 Translation, Scaling, and Shearing

\TransformShift `\TransformShift {⟨dx, dy⟩}`
`\TransformShift {⟨dx, dy, dz⟩}`

New: 2026-11-21

Adds a translation. The two-dimensional form uses horizontal and vertical offsets `dx` and `dy`; the three-dimensional form also includes the depth offset `dz`. The number of parameters must match the current transformation dimension.

\TransformScale `\TransformScale [⟨center⟩] {⟨scale⟩}`
`\TransformScale [⟨center⟩] {⟨sx, sy⟩}`
`\TransformScale [⟨center⟩] {⟨sx, sy, sz⟩}`

New: 2026-11-21

Adds a scaling transformation. A single scaling factor applies the same scale to all directions; two factors specify the scaling factors for the two-dimensional axes, and three factors specify the scaling factors for the three-dimensional axes. The optional argument ⟨*center*⟩ specifies the scaling center. For two-dimensional transformations, it has the form `x,y`; for three-dimensional transformations, it has the form `x,y,z`. If omitted, the origin is used as the center. The number of scaling factors must match the current transformation dimension, except when a single uniform scaling factor is supplied.

\TransformShear `\TransformShear [⟨center⟩] {⟨a, b⟩}`
`\TransformShear [⟨center⟩] {⟨a, b, c, d, e, f⟩}`

New: 2026-11-21

Adds a shear transformation. The two-dimensional form uses two coefficients and corresponds to the transformation matrix $\left[\begin{smallmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right]$. The three-dimensional form uses six coefficients and corresponds to the transformation matrix $\left[\begin{smallmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix}\right]$. The optional argument ⟨*center*⟩ specifies the shear center. Its format is the same as that used by `\TransformScale`. If omitted, the origin is used as the center. The number of coefficients must match the current transformation dimension.

2.7.3 Rotations

\TransformRotate `\TransformRotate [⟨center⟩] {⟨angle in radians⟩}`
\TransformRotated `\TransformRotated [⟨center⟩] {⟨angle in degrees⟩}`

New: 2026-11-21

Adds a two-dimensional rotation. The current transformation must be two-dimensional. The optional argument ⟨*center*⟩ specifies the rotation center in the form `x, y`. If omitted, the rotation is performed about the origin.

\TransformRotateX `\TransformRotateX [⟨center⟩] {⟨angle in radians⟩}`
\TransformRotateY `\TransformRotateY [⟨center⟩] {⟨angle in radians⟩}`
\TransformRotateZ `\TransformRotateZ [⟨center⟩] {⟨angle in radians⟩}`
\TransformRotatedX `\TransformRotatedX [⟨center⟩] {⟨angle in degrees⟩}`
\TransformRotatedY `\TransformRotatedY [⟨center⟩] {⟨angle in degrees⟩}`
\TransformRotatedZ `\TransformRotatedZ [⟨center⟩] {⟨angle in degrees⟩}`

New: 2026-11-21

Adds a three-dimensional rotation about the x, y, or z axis. The current transformation must be three-dimensional. The optional argument ⟨*center*⟩ specifies the rotation center in the form `x, y, z`. If omitted, the rotation is performed about the origin.

2.7.4 Refreshing and Applying Transformations

Transformation commands only update the current transformation matrix. To convert the matrix coefficients into callable coordinate transformation functions, a refresh operation is required. The application commands can then be used to calculate coordinates.

\TransformRefresh `\TransformRefresh`
`\TransformRefresh [⟨matrix⟩]`

New: 2026-11-21

Refreshes the internal coordinate transformation functions using a transformation matrix. If the optional argument is omitted, the current transformation matrix is used. If ⟨*matrix*⟩ is specified, the transformation functions are extracted from that matrix instead. A two-dimensional transformation matrix must be 3×3, and a three-dimensional transformation matrix must be 4×4. In both cases, the bottom-right element must be equal to 1; otherwise, an invalid transformation matrix error is reported.

\TransformApply `\TransformApply [⟨suffix⟩] {⟨coordinates⟩}`

New: 2026-11-21

Applies the corresponding transformation to the specified coordinates and outputs the result. ⟨*coordinates*⟩ is a comma-separated list of coordinate components, containing either two or three components for two-dimensional or three-dimensional transformations, respectively. Coordinate components must be numerical values without units. The resulting components are output in order, separated by commas, and do not include the homogeneous component. ⟨*suffix*⟩ is appended to each transformed coordinate component, for example, to add a unit. It may be left empty when no additional content is required.

2.8 Error Conditions

The current implementation reports an error under the following conditions:

- The number of rows or columns is negative when creating a matrix;
- The matrix variable has already been defined when creating a matrix;
- A constant matrix is used as the target of a modification operation;
- The number of items in the element list exceeds the total number of elements in the matrix when creating or setting a matrix;
- The dimensions of the two operands do not match when performing an element-wise binary operation between matrices;
- The number of columns in the left operand does not match the number of rows in the right operand when performing matrix multiplication;
- The number of arguments supplied to a geometric transformation does not match the corresponding two-dimensional or three-dimensional form;
- The matrix supplied when refreshing a transformation is not a valid 3×3 or 4×4 transformation matrix, or its bottom-right element is not equal to 1.

3 实现设计

一个矩阵对象由矩阵描述符和矩阵数据两部分组成。矩阵描述符中记录了矩阵的行数、列数、元素总数以及矩阵数据的逻辑索引；矩阵数据则存储在某个 fparray 数组中。

矩阵变量直接存储矩阵描述符，矩阵操作函数则通过矩阵描述符找到矩阵数据。

由于 fparray 数组的修改采用全局操作，为保证矩阵描述符与矩阵数据的更新保持同步，矩阵的创建和修改也均采用全局操作。即矩阵的创建和修改不会因普通 TEX 分组结束而自动恢复。

3.1 矩阵描述符

矩阵描述符是一个特殊的凭据表，由以下四个整数字段组成：`<length>`、`<rows>`、`<columns>` 和 `<space-index>`。各字段之间使用专用扫描标记分隔，并以结束标记结尾。各字段的含义如下：

`<length>` 矩阵的元素总数，即行数与列数的乘积；

`<rows>` / `<columns>` 矩阵的行数和列数；

`<space-index>` 矩阵数据所在空间的逻辑索引。

3.2 存储管理

矩阵数据存储在 fparray 数组的一段连续空间中。存储管理器负责这段空间的分配、回收与定位。

3.2.1 存储类别

通常，存储管理器将固定长度的 fparray 数组按照固定的空间大小（`<size>`）划分为若干个独立空间，每个空间用于存储一个矩阵。目前，存储管理器提供 A、B、C 三种常规存储类别，分别用于存储小型、中型和大型矩阵。对于超大型矩阵，则采用特殊的 D 类存储，为每个矩阵单独创建数组。

各类别的当前参数如下：

类别	数组大小	空间的大小	独立空间数
A	4000	20	200
B	4000	100	40
C	4000	1000	4
D	矩阵长度	矩阵长度	1

3.2.2 空间定位

矩阵实际使用存储类别由其元素总数所决定，具体划分如下：元素总数不超过 20 时使用 A 类；超过 20 且不超过 100 时使用 B 类；超过 100 且不超过 1000 时使用 C 类；超过 1000 时使用 D 类。

存储类别与逻辑索引共同确定矩阵数据空间的实际位置：实际数组及偏移量。

数组名称由存储类别字母和数组编号组成（`\g_@@_<class>_<sn>_fparray`）。存储管理器根据逻辑索引以及该类别每个数组所能容纳的空间数（$C_{\texttt{<class>}}$），计算矩阵所属的数组编号，其公式如下：[1]

$$\texttt{<sn>} = \left\lceil \frac{\texttt{<space-index>}}{C_{\texttt{<class>}}} \right\rceil = \left\lfloor \frac{\texttt{<space-index>} - 1}{C_{\texttt{<class>}}} \right\rfloor + 1$$

逻辑索引对应数组的第 k 个空间及其偏移量的公式如下：

$$k = \texttt{<space-index>} - (\texttt{<sn>} - 1)C_{\texttt{<class>}}$$
$$\texttt{<offset>} = (k-1)S_{\texttt{<class>}}$$

对于 D 类，这三个公式可以简化：

$$\texttt{<sn>} = \texttt{<space-index>}$$
$$k = 1$$
$$\texttt{<offset>} = 0$$

3.2.3 元素定位

矩阵数据是按行顺序存储的，因此，对于任意矩阵 A，其第 i 行第 j 列的元素，其位置由以下公式确定：

$$\texttt{<element-index>} = (i-1)\texttt{<columns>} + j$$
$$\texttt{<array-index>} = \texttt{<offset>} + \texttt{<element-index>}$$

3.2.4 分配与回收

空间分配优先使用回收序列中的索引；没有可回收空间时，才使用该类别的下一个索引。元素总数为零时不分配实际空间。普通空矩阵使用索引 0，常量空矩阵使用索引 -1；这两个值是空矩阵的逻辑标记，不对应任何数组或数组区间。

回收时，A、B、C 类只将空间索引放回对应的回收序列，底层数组继续保留，后续分配可以复用该空间。D 类回收时，除登记索引外，还会删除对应的独立数组。空间回收函数本身不判断矩阵是否为常量；常量空间是否允许回收由矩阵操作层保证。

[1] expl3 宏包暂时还没有实现向上取整及向下取整，所以使用截断除来处理。

3.3 常量矩阵

常量矩阵不使用额外的常量标记字段。宏包使用一个全局序列保存常量矩阵的完整描述符内容，并通过描述符内容比较判断矩阵是否为常量。修改矩阵之前，矩阵操作层检查目标描述符是否存在于该序列中；匹配时报告常量矩阵错误，并中止当前操作。

这种识别方式以描述符内容而不是控制序列名称为依据。因此，复制或等价引用常量矩阵描述符后，只要描述符内容不变，仍会被识别为常量矩阵。常量矩阵的非空数据仍按正常类别规则分配空间。

常量保护由矩阵操作接口实现，存储管理器不承担常量判断。该保护针对宏包提供的矩阵修改接口，不等同于对底层 fparray 变量的访问控制。

3.4 零矩阵与空矩阵

零矩阵是所有元素均为零的矩阵，不是一种单独的存储类别。零矩阵仍按其元素总数选择 A、B、C 或 D 类空间；归零操作直接将其现有空间置零。

空矩阵是元素总数为零的合法矩阵，包括 0×0、$m\times 0$ 和 $0\times n$。空矩阵不分配实际存储空间，但行数和列数仍保留在描述符中。因此，不同维度的空矩阵仍是不同形状的矩阵，参与运算时仍须执行维度兼容性检查。

对于结果元素总数为零的运算，结果描述符仍会按目标维度更新，但不会分配空间。

4 宏包准备

```
1 ⟨*package⟩
2 ⟨@@=matrix⟩
3 \NeedsTeXFormat{LaTeX2e}
4 \ProvidesExplPackage{lt3matrix}{2026-11-23}{0.2}
5   {a lightweight global matrix base-on fparray}
```

定义一些 expl3 函数的常用变体。

```
6 \cs_generate_variant:Nn \iow_log:n { V, e }
```

\l__matrix_tmpa_int
\l__matrix_tmpb_int
\l__matrix_tmpa_fp
\l__matrix_tmpb_fp
\l__matrix_tmpa_tl
\l__matrix_tmpb_tl
\g__matrix_tmpa_tl
\g__matrix_tmpb_tl

定义一些通用的临时变量。

```
7  \int_new:N  \l__matrix_tmpa_int
8  \int_new:N  \l__matrix_tmpb_int
9  \fp_new:N   \l__matrix_tmpa_fp
10 \fp_new:N   \l__matrix_tmpb_fp
11 \tl_new:N   \l__matrix_tmpa_tl
12 \tl_new:N   \l__matrix_tmpb_tl
```

```
\tl_new:N   \g__matrix_tmpa_tl
\tl_new:N   \g__matrix_tmpb_tl
```

(*End of definition for* `\l__matrix_tmpa_int` *and others.*)

5 核心命令

5.1 消息

negative-dimension
constant-matrix
too-many-elements
dimension-mismatch
invalid-transformation-matrix

错误消息。

```
\msg_new:nnn { matrix } { negative-dimension }
  {
    Matrix~dimensions('#1')~must~be~non-negative:.
  }
\msg_new:nnn { matrix } { constant-matrix }
  {
    The~matrix('#1')~is~constant~and~cannot~be~modified.
  }
\msg_new:nnn { matrix } { too-many-elements }
  {
    The~input~list('#1')~contains~too~many~elements.
  }
\msg_new:nnn { matrix } { dimension-mismatch }
  {
    The~dimensions~of~the~operands('#1')~are~incompatible~for~this~operation.
  }
\msg_new:nnn { matrix } { invalid-transformation-matrix }
  { The~transformation~matrix('#1')~is~not~a~valid~transformation~matrix. }
```

(*End of definition for* `negative-dimension` *and others. These variables are documented on page* ??.)

5.2 元信息

矩阵的元信息包含总长度、行数、列数及实际数据索引，索引指向实际数据。

5.2.1 基础变量

\s__matrix
\s__matrix_mark
\s__matrix_stop

存储数据用的扫描标记。

```
\scan_new:N \s__matrix
\scan_new:N \s__matrix_mark
\scan_new:N \s__matrix_stop
```

(*End of definition for* `\s__matrix`, `\s__matrix_mark`, *and* `\s__matrix_stop`.)

\g__matrix_const_seq 常量矩阵登记表。

```
\seq_new:N \g__matrix_const_seq
```

(*End of definition for* `\g__matrix_const_seq`.)

\l__matrix_length_int 保存矩阵信息的临时变量。
\l__matrix_row_int
\l__matrix_col_int
\l__matrix_index_int

```
\int_new:N  \l__matrix_length_int
\int_new:N  \l__matrix_row_int
\int_new:N  \l__matrix_col_int
\int_new:N  \l__matrix_index_int
```

(*End of definition for* `\l__matrix_length_int` *and others.*)

5.2.2 设置信息

__matrix_meta_gset:N 将 #1 的值设置为矩阵基本信息。
__matrix_meta_gset:Nnnnn

```
\cs_new:Nn \__matrix_meta_gset:Nnnnn
  {
    \tl_gset:Nn #1
      {
        \s__matrix      #2 % Length
        \s__matrix_mark #3 % Row Count
        \s__matrix_mark #4 % Column Count
        \s__matrix_mark #5 % Index
        \s__matrix_stop
      }
  }
\cs_generate_variant:Nn \__matrix_meta_gset:Nnnnn { NVVVV }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_meta_gset:N
  {
    \__matrix_meta_gset:NVVVV #1
      \l__matrix_length_int
      \l__matrix_row_int
      \l__matrix_col_int
      \l__matrix_index_int
  }
```

(*End of definition for* `\__matrix_meta_gset:N` *and* `\__matrix_meta_gset:Nnnnn`.)

5.2.3 解析信息

__matrix_meta_parse_aux:w 解析矩阵信息，并结果设置到 #5~#8。

```
\cs_new:Npn \__matrix_meta_parse_aux:w
  \s__matrix      #1 \s__matrix_mark #2
  \s__matrix_mark #3 \s__matrix_mark #4
  \s__matrix_stop #5#6#7#8
  {
    \int_set:Nn #5 {#1}
    \int_set:Nn #6 {#2}
    \int_set:Nn #7 {#3}
    \int_set:Nn #8 {#4}
  }
```

(*End of definition for* __matrix_meta_parse_aux:w.)

__matrix_meta_parse:N 解析矩阵信息，并结果设置到 #2~#5。

__matrix_meta_parse:NNNNN

```
\cs_new:Nn \__matrix_meta_parse:NNNNN
  {
    \exp_last_unbraced:NV
      \__matrix_meta_parse_aux:w #1 #2#3#4#5
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_meta_parse:N
  {
    \exp_last_unbraced:NV
      \__matrix_meta_parse_aux:w #1
      \l__matrix_length_int
      \l__matrix_row_int
      \l__matrix_col_int
      \l__matrix_index_int
  }
```

(*End of definition for* __matrix_meta_parse:N *and* __matrix_meta_parse:NNNNN.)

5.2.4 维度判定

__matrix_meta_if_empty:w 判定矩阵长度是否为空。

```
\cs_new:Npn \__matrix_meta_if_empty:w
  \s__matrix      #1 \s__matrix_mark #2
  \s__matrix_mark #3 \s__matrix_mark #4
  \s__matrix_stop
  { \int_compare_p:nNn {#1} = {0} }
```

(*End of definition for* `\__matrix_meta_if_empty:w`.)

`\__matrix_meta_if_square:w` 判定矩阵是否为方阵。

```
\cs_new:Npn \__matrix_meta_if_square:w
  \s__matrix      #1 \s__matrix_mark #2
  \s__matrix_mark #3 \s__matrix_mark #4
  \s__matrix_stop
  { \int_compare_p:nNn {#2} = {#3} }
```

(*End of definition for* `\__matrix_meta_if_square:w`.)

`\__matrix_meta_if_same_dim:w` 判定两个矩阵的行列数是否相等。

```
\cs_new:Npn \__matrix_meta_if_same_dim:w
  \s__matrix      #1 \s__matrix_mark #2
  \s__matrix_mark #3 \s__matrix_mark #4
  \s__matrix_stop
  \s__matrix      #5 \s__matrix_mark #6
  \s__matrix_mark #7 \s__matrix_mark #8
  \s__matrix_stop
  {
    \bool_lazy_and_p:nn
      { \int_compare_p:nNn {#2} = {#6} }
      { \int_compare_p:nNn {#3} = {#7} }
  }
```

(*End of definition for* `\__matrix_meta_if_same_dim:w`.)

5.3 数据池

在实际实现中，每个 fparray 按固定大小划分为若干空间，每个空间只存储一份矩阵数据。根据空间大小，预定义小（A 类）、中（B 类）和大（C 类）三种存储池，以适应不同大小的矩阵。

对于元素总数超过 C 类空间容量的矩阵（D 类），则为每个矩阵单独分配一个数组。

5.3.1 常量

`\c__matrix_size_a_int` `\c__matrix_size_b_int` `\c__matrix_size_c_int` 三种划分 fparray 对应的空间大小。

```
\int_const:Nn \c__matrix_size_a_int { 20   }
\int_const:Nn \c__matrix_size_b_int { 100  }
\int_const:Nn \c__matrix_size_c_int { 1000 }
```

(*End of definition for* `\c__matrix_size_a_int`, `\c__matrix_size_b_int`, *and* `\c__matrix_size_c_int`.)

\c__matrix_array_length_int 每个 fparray 变量的长度。

```
\int_const:Nn \c__matrix_array_length_int { 4000 }
```

(*End of definition for* \c__matrix_array_length_int.)

\c__matrix_count_a_int 单个 fparray 实际划分出的空间数量。
\c__matrix_count_b_int
\c__matrix_count_c_int

```
\int_const:Nn \c__matrix_count_a_int
  { \c__matrix_array_length_int / \c__matrix_size_a_int }
\int_const:Nn \c__matrix_count_b_int
  { \c__matrix_array_length_int / \c__matrix_size_b_int }
\int_const:Nn \c__matrix_count_c_int
  { \c__matrix_array_length_int / \c__matrix_size_c_int }
```

(*End of definition for* \c__matrix_count_a_int, \c__matrix_count_b_int, *and* \c__matrix_count_c_int.)

5.3.2 变量

\g__matrix_a_i_fparray 四种数据池演示，当前不实际申请（使用时自动申请）。
\g__matrix_b_i_fparray 数据池的变量名形式为：\g__matrix_<class>_<array index>_faparray。
\g__matrix_c_i_fparray
\g__matrix_d_i_fparray

```
⟨demo⟩\fparray_new:Nn \g__matrix_a_i_fparray { \c__matrix_array_length_int }
⟨demo⟩\fparray_new:Nn \g__matrix_b_i_fparray { \c__matrix_array_length_int }
⟨demo⟩\fparray_new:Nn \g__matrix_c_i_fparray { \c__matrix_array_length_int }
⟨demo⟩\fparray_new:Nn \g__matrix_d_i_fparray { \l__matrix_length_int }
```

(*End of definition for* \g__matrix_a_i_fparray *and others.*)

\g__matrix_next_a_int 数据池中的下一个空间索引。
\g__matrix_next_b_int
\g__matrix_next_c_int
\g__matrix_next_d_int

```
\int_new:N    \g__matrix_next_a_int
\int_new:N    \g__matrix_next_b_int
\int_new:N    \g__matrix_next_c_int
\int_new:N    \g__matrix_next_d_int

\int_gset:Nn  \g__matrix_next_a_int { 1 }
\int_gset:Nn  \g__matrix_next_b_int { 1 }
\int_gset:Nn  \g__matrix_next_c_int { 1 }
\int_gset:Nn  \g__matrix_next_d_int { 1 }
```

(*End of definition for* \g__matrix_next_a_int *and others.*)

\g__matrix_free_a_seq 数据池回收的空间索引。
\g__matrix_free_b_seq
\g__matrix_free_c_seq
\g__matrix_free_d_seq

```
\seq_new:N    \g__matrix_free_a_seq
\seq_new:N    \g__matrix_free_b_seq
\seq_new:N    \g__matrix_free_c_seq
\seq_new:N    \g__matrix_free_d_seq
```

(*End of definition for* `\g__matrix_free_a_seq` *and others.*)

`\l__matrix_space_class_tl` 用于存储空间属性的临时变量。

`\l__matrix_space_size_int`

```
\tl_new:N   \l__matrix_space_class_tl
\int_new:N  \l__matrix_space_size_int
```

(*End of definition for* `\l__matrix_space_class_tl` *and* `\l__matrix_space_size_int`.)

`\l__matrix_fparray` 用于存储实际分配的空间的信息的临时变量。

`\l__matrix_offset_int`

```
\cs_new:Npn \l__matrix_fparray {}
\int_new:N  \l__matrix_offset_int
```

(*End of definition for* `\l__matrix_fparray` *and* `\l__matrix_offset_int`.)

5.3.3 空间选择

`\__matrix_space_select:` 查询某个长度（`#1`）的矩阵对应的空间的相关属性（类型与大小）。结果保存在 `\l__matrix_space_type_tl` 与 `\l__matrix_space_size_int` 中。

`\__matrix_space_select:n`

```
\cs_new:Nn \__matrix_space_select:n
  {
    \int_compare:nNnTF {#1} > { \c__matrix_size_a_int }
      {
        \int_compare:nNnTF {#1} > { \c__matrix_size_b_int }
          {
            \int_compare:nNnTF {#1} > { \c__matrix_size_c_int }
              { % Class D
                \tl_set:Nn  \l__matrix_space_class_tl {d}
                \int_set:Nn \l__matrix_space_size_int {#1}
              }
              {  % Class C
                \tl_set:Nn  \l__matrix_space_class_tl {c}
                \int_set:Nn \l__matrix_space_size_int { \c__matrix_size_c_int }
              }
          }
          { % Class B
              \tl_set:Nn  \l__matrix_space_class_tl {b}
              \int_set:Nn \l__matrix_space_size_int { \c__matrix_size_b_int }
          }
      }
      {
        \int_compare:nNnTF {#1} = { 0 }
          { % Empty
            \tl_set:Nn  \l__matrix_space_class_tl {}
```

```
            \int_set:Nn \l__matrix_space_size_int { 0 }
          }
          { % Class A
            \tl_set:Nn  \l__matrix_space_class_tl {a}
            \int_set:Nn \l__matrix_space_size_int { \c__matrix_size_a_int }
          }
      }
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_space_select:
  {
    \__matrix_space_select:n { \l__matrix_length_int }
  }
```

(*End of definition for* `\__matrix_space_select:` *and* `\__matrix_space_select:n`.)

5.3.4 空间分配

`\__matrix_space_allocate_aux:n` 获取下一个可用空间索引。`#1` 空间类型，结果保存在 `\l__matrix_index_int` 中。

```
\cs_new:Nn \__matrix_space_allocate_aux:n
  {
    \seq_if_empty:cTF { g__matrix_free_#1_seq }
      { % 无回收空间，使用下一个空间
        \int_set_eq:Nc \l__matrix_index_int { g__matrix_next_#1_int  }
        \int_set_eq:Nc \l__matrix_tmpa_int  { c__matrix_count_#1_int }
        \int_compare:nNnT
          { \int_mod:nn { \l__matrix_index_int } { \l__matrix_tmpa_int } } = { 1 }
          { % 需要新数组
            \int_set:Nn \l__matrix_tmpb_int
              { \int_div_truncate:nn { \l__matrix_index_int - 1} { \l__matrix_tmpa_int } }
            \int_incr:N \l__matrix_tmpb_int
            \exp_args:Nc \fparray_new:Nn
              { g__matrix_#1_ \int_to_roman:n { \l__matrix_tmpb_int } _fparray }
              { \c__matrix_array_length_int }
          }
          \int_gincr:c { g__matrix_next_#1_int }
      }
      { % 有回收空间，直接使用
        \seq_gpop_left:cN { g__matrix_free_#1_seq } \l__matrix_tmpa_tl
        \int_set:Nn \l__matrix_index_int { \l__matrix_tmpa_tl }
      }
  }
```

```
\cs_generate_variant:Nn \__matrix_space_allocate_aux:n { V }
```

(*End of definition for* `\__matrix_space_allocate_aux:n`.)

`\__matrix_space_allocate_aux_d:n` 获取下一个可用空间索引（D 类型）。#1 空间大小，结果保存在 `\l__matrix_index_int` 中。

```
\cs_new:Nn \__matrix_space_allocate_aux_d:n
  {
    \seq_if_empty:NTF \g__matrix_free_d_seq
      { % 无回收空间，使用下一个空间
        \int_set_eq:NN \l__matrix_index_int \g__matrix_next_d_int
        \int_gincr:N \g__matrix_next_d_int
      }
      { % 有回收空间，直接使用
        \seq_gpop_left:NN \g__matrix_free_d_seq \l__matrix_tmpa_tl
        \int_set:Nn \l__matrix_index_int { \l__matrix_tmpa_tl }
      }
      \exp_args:Nc \fparray_new:Nn
        { g__matrix_d_ \int_to_roman:n { \l__matrix_index_int } _fparray }
        { #1 }
  }
```

(*End of definition for* `\__matrix_space_allocate_aux_d:n`.)

`\__matrix_space_allocate:` `\__matrix_space_allocate:n` 申请空间。#1 空间大小，结果保存在 `\l__matrix_index_int` 中。

```
\cs_new:Nn \__matrix_space_allocate:n
  {
    \int_compare:nNnTF {#1} = { 0 }
      { \int_set:Nn \l__matrix_index_int { 0 } } % Emtpy Matrices
      {
        \__matrix_space_select:n {#1}
        \int_compare:nNnTF {#1} > { \c__matrix_size_c_int }
          { \__matrix_space_allocate_aux_d:n {#1} } % Class D
          { \__matrix_space_allocate_aux:V \l__matrix_space_class_tl } % Class A ~ C
      }
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_space_allocate:
  {
    \__matrix_space_allocate:n { \l__matrix_length_int }
  }
```

(*End of definition for* `\__matrix_space_allocate:` *and* `\__matrix_space_allocate:n`.)

5.3.5 空间回收

__matrix_space_free_aux:nn 回收空间辅助函数。#1 空间大小，#2 空间索引。

```
\cs_new:Nn \__matrix_space_free_aux:nn
  {
    \__matrix_space_select:n {#1}
    \seq_gput_right:cn { g__matrix_free_ \l__matrix_space_class_tl _seq } {#2}
    \int_compare:nNnT {#1} > { \c__matrix_size_c_int }
      { \cs_undefine:c { g__matrix_d_ \int_to_roman:n { #2 } _fparray } }
  }
\cs_generate_variant:Nn \__matrix_space_free_aux:nn { VV }
```

(*End of definition for* __matrix_space_free_aux:nn.)

__matrix_space_free: __matrix_space_free:nn 回收空间函数。#1 空间大小，#2 空间索引。不负责常量空间校对。

```
\cs_new:Nn \__matrix_space_free:nn
  {
    \int_compare:nNnT {#1} > {0}
      {
        \int_set:Nn \l__matrix_tmpa_int {#1}
        \int_set:Nn \l__matrix_tmpb_int {#2}
        \__matrix_space_free_aux:VV \l__matrix_tmpa_int \l__matrix_tmpb_int
      }
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_space_free:
  {
    \__matrix_space_free:nn
      { \l__matrix_length_int }
      { \l__matrix_index_int  }
  }
```

(*End of definition for* __matrix_space_free: *and* __matrix_space_free:nn.)

5.3.6 查询定位

__matrix_space_query_aux:nn 查询指定类型的空间的信息。#1 空间类型，#2 空间索引。结果保存在 \l__matrix_fparray 与 \l__matrix_offset_int 中。

```
\cs_new:Nn \__matrix_space_query_aux:nn
  {
    \int_set_eq:Nc \l__matrix_tmpa_int { c__matrix_count_#1_int }
    \int_set:Nn    \l__matrix_tmpb_int
      { \int_div_truncate:nn { #2 - 1 } { \l__matrix_tmpa_int } }
```

```
    \int_incr:N    \l__matrix_tmpb_int
    \cs_set_eq:Nc  \l__matrix_fparray
      { g__matrix_#1_ \int_to_roman:n { \l__matrix_tmpb_int } _fparray }
    \int_set:Nn    \l__matrix_offset_int
      {
        \int_mod:nn { #2 - 1 } { \l__matrix_tmpa_int } *
        \int_use:c  { c__matrix_size_#1_int }
      }
  }
\cs_generate_variant:Nn \__matrix_space_query_aux:nn { V }
```

(*End of definition for* `\__matrix_space_query_aux:nn`.)

`\__matrix_space_query:`
`\__matrix_space_query:nn`

查询空间信息。#1 空间大小，#2 空间索引。结果保存在 `\l__matrix_fparray` 与 `\l__matrix_offset_int` 中。

```
\cs_new:Nn \__matrix_space_query:nn
  {
    \__matrix_space_select:n {#1}
    \int_compare:nNnTF {#1} = { 0 }
      {
        \int_zero:N \l__matrix_offset_int
        \cs_set:Npn \l__matrix_fparray {}
      }
      {
        \int_compare:nNnTF {#1} > { \c__matrix_size_c_int }
          { % Class D
            \int_set:Nn   \l__matrix_offset_int { 0 }
            \cs_set_eq:Nc \l__matrix_fparray
              { g__matrix_d_ \int_to_roman:n {#2} _fparray }
          }
          { \__matrix_space_query_aux:Vn \l__matrix_space_class_tl {#2} } % Class A ~ C
      }
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_space_query:
  {
    \__matrix_space_query:nn
      { \l__matrix_length_int }
      { \l__matrix_index_int  }
  }
```

(*End of definition for* `\__matrix_space_query:` *and* `\__matrix_space_query:nn`.)

5.3.7 数据设置

__matrix_data_gzero:
__matrix_data_gzero:Nnn

将指定 fparray 的指定区间的数据置零。#1 数组，#2 偏移量，#3 长度。

```
\cs_new:Nn \__matrix_data_gzero:Nnn
  {
    \int_step_inline:nnn { #2 + 1 } { #2 + #3 }
      { \fparray_gset:Nnn #1 {##1} { 0 } }
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_data_gzero:
  {
    \__matrix_data_gzero:Nnn \l__matrix_fparray
      { \l__matrix_offset_int }
      { \l__matrix_space_size_int }
  }
```

(*End of definition for* `\__matrix_data_gzero:` *and* `\__matrix_data_gzero:Nnn`.)

__matrix_data_gset:n
__matrix_data_gset:Nnnn

使用指定数据将填充 fparray 的指定区间，数据不足时补零。#1 数组，#2 偏移量，#3 长度，#4 指定数据。

```
\cs_new:Nn \__matrix_data_gset:Nnnn
  {
    \int_set:Nn \l__matrix_tmpa_int { #2 + 1 }
    \clist_map_inline:nn {#4} % Fill Data
      {
        \fparray_gset:Nnn #1 { \l__matrix_tmpa_int } {##1}
        \int_incr:N \l__matrix_tmpa_int
      }
    \int_step_inline:nnn      % Fill 0
      { \l__matrix_tmpa_int } { #2 + #3 }
      { \fparray_gset:Nnn #1 {##1} { 0 } }
  }
```

这是一个使用临时变量的简化版本。

```
\cs_new:Nn \__matrix_data_gset:n
  {
    \__matrix_data_gset:Nnnn \l__matrix_fparray
      { \l__matrix_offset_int }
      { \l__matrix_space_size_int }
      {#1}
  }
```

(*End of definition for* `\__matrix_data_gset:n` *and* `\__matrix_data_gset:Nnnn`.)

__matrix_data_gcopy:NnNnn 使两块区域的内容完成相等。#1 目标数组，#2 目标偏移量，#3 源数组，#4 源偏移量，#4 长度。

```
\cs_new:Nn \__matrix_data_gcopy:NnNnn
  {
    \int_step_inline:nn {#5}
      {
        \fparray_gset:Nnn #1 { #2 + ##1 }
          { \fparray_item:Nn #3 { #4 + ##1 } }
      }
  }
```

(*End of definition for* __matrix_data_gcopy:NnNnn.)

5.3.8 数据输出

__matrix_data_typeset:Nnnn 在公式环境中排版 fparray 内容，命令本身不会自动切换到公式环境。#1 数组，#2 偏移量，#3 长度，#4 每行列数。

```
\cs_new:Nn \__matrix_data_typeset:Nnnn
  {
    \tl_set:Nn \l__matrix_tmpa_tl
      { \left( \begin{array}{*{#4}{c}} }
    \int_step_inline:nnn { #2 + 1 } { #2 + #3 }
      {
        \tl_put_right:Ne \l__matrix_tmpa_tl
          { \fparray_item:Nn #1 {##1} }
        \int_compare:nNnTF
          { \int_mod:nn { ##1 - #2 } { #4 } } = { 0 }
          { \tl_put_right:Nn \l__matrix_tmpa_tl { \\ } }
          { \tl_put_right:Nn \l__matrix_tmpa_tl { & } }
      }
    \regex_replace_once:nnN { \c{\\}\Z } {} \l__matrix_tmpa_tl
    \tl_put_right:Nn \l__matrix_tmpa_tl
      { \end{array} \right) }
    \l__matrix_tmpa_tl
  }
```

(*End of definition for* __matrix_data_typeset:Nnnn.)

__matrix_data_log:Nnnn 在日志中查看 fparray 内容。#1 数组，#2 偏移量，#3 长度，#4 每行列数。

```
\cs_new:Nn \__matrix_data_log:Nnnn
  {
    \int_set:Nn \l__matrix_tmpa_int { 1 }
    \tl_set:Ne \l__matrix_tmpa_tl
```

```
      { <\int_use:N \l__matrix_tmpa_int>:~ }
    \int_step_inline:nnn { #2 + 1 } { #2 + #3 }
      {
        \tl_put_right:Ne \l__matrix_tmpa_tl
          { \fparray_item:Nn #1 {##1} }
        \int_compare:nNnTF
          { \int_mod:nn { ##1 - #2 } { #4 } } = { 0 }
          {
            \iow_log:V  \l__matrix_tmpa_tl
            \int_incr:N \l__matrix_tmpa_int
            \tl_set:Ne  \l__matrix_tmpa_tl
              { <\int_use:N \l__matrix_tmpa_int>:~ }
          }
          { \tl_put_right:Nn \l__matrix_tmpa_tl  { ,~ } }
      }
  }
```

(*End of definition for* `\__matrix_data_log:Nnnn`.)

`\__matrix_data_log:Nnnnn` 在日志中查看 fparray 内容。#1 数组，#2 偏移量，#3 长度，#4 每行列数, #5 空间长度。

```
\cs_new:Nn \__matrix_data_log:Nnnnn
  {
    \__matrix_data_log:Nnnn #1 {#2} {#3} {#4}
    \tl_set:Nn \l__matrix_tmpa_tl { <unused>: }
    \int_step_inline:nnn { #3 + 1 } { #5 }
      {
        \tl_put_right:Ne \l__matrix_tmpa_tl
          { ~\fparray_item:Nn #1 { #2 + ##1 }, }
      }
    \int_compare:nNnTF {#3} = {#5}
      { \tl_put_right:Nn \l__matrix_tmpa_tl { ~none. }  }
      { \regex_replace_once:nnN { ,\Z } { . } \l__matrix_tmpa_tl  }
    \iow_log:V \l__matrix_tmpa_tl
  }
```

(*End of definition for* `\__matrix_data_log:Nnnnn`.)

6 矩阵命令

6.1 创建与初始化矩阵

6.1.1 创建矩阵

\matrix_new:N
\matrix_new:Nnn
\matrix_new:cnn

定义新矩阵。

```
\cs_new_protected:Nn \matrix_new:Nnn
  {
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn {#2} < {0} }
      { \int_compare_p:nNn {#3} < {0} }
      { \msg_error:nnn { matrix } { negative-dimension } {#2x#3} }
      {
        \tl_new:N #1  % 防止变量已使用
        \int_set:Nn \l__matrix_length_int { #2 * #3 }
        \int_set:Nn \l__matrix_row_int {#2}
        \int_set:Nn \l__matrix_col_int {#3}
        \int_compare:nNnTF
          { \l__matrix_length_int } = { 0 }
          { \int_set:Nn \l__matrix_index_int { 0 } } % Empty Matrices
          { % Non-Empty Matrices
            \__matrix_space_allocate:
            \__matrix_space_query:
            \__matrix_data_gzero:
          }
        \__matrix_meta_gset:N #1
      }
  }
\cs_generate_variant:Nn \matrix_new:Nnn { c }
\cs_new_protected:Nn \matrix_new:N
  { \matrix_new:Nnn #1 {0} {0} }
```

(*End of definition for* `\matrix_new:N` *and* `\matrix_new:Nnn`. *These functions are documented on page* 16.)

\matrix_const:Nnnn
\matrix_const:cnnn

定义新矩阵。

```
\cs_new_protected:Nn \matrix_const:Nnnn
  {
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn {#2} < {0} }
      { \int_compare_p:nNn {#3} < {0} }
      { \msg_error:nnn { matrix } { negative-dimension } {#2x#3} }
      {
```

```
      \int_set:Nn \l__matrix_length_int { #2 * #3 }
      \int_set:Nn \l__matrix_row_int {#2}
      \int_set:Nn \l__matrix_col_int {#3}
      \int_compare:nNnTF
        { \clist_count:n {#4} } > { \l__matrix_length_int }
        { \msg_error:nnn { matrix } { too-many-elements } {#4} }
        {
          \tl_new:N #1  % 防止变量已使用
          \int_compare:nNnTF
            { \l__matrix_length_int } = { 0 }
            { \int_set:Nn \l__matrix_index_int { -1 } } % Empty Matrices
            { % Non-Empty Matrices
              \__matrix_space_allocate:
              \__matrix_space_query:
              \__matrix_data_gset:n {#4}
            }
          \__matrix_meta_gset:N #1
          \seq_gput_right:NV \g__matrix_const_seq #1
        }
    }
  }
\cs_generate_variant:Nn \matrix_const:Nnnn { c }
```

(*End of definition for* `\matrix_const:Nnnn`. *This function is documented on page* 16.)

6.1.2 设置矩阵

`\matrix_gzero:N` `\matrix_gzero:c`

矩阵归零。

```
\cs_new_protected:Nn \matrix_gzero:N
  {
    \seq_if_in:NVTF \g__matrix_const_seq #1
      { \msg_error:nnn { matrix } { constant-matrix } {#1} }
      {
        \__matrix_meta_parse:N #1
        \int_compare:nNnT
          { \l__matrix_length_int } > { 0 }
          { \__matrix_space_query: \__matrix_data_gzero: }
      }
  }
\cs_generate_variant:Nn \matrix_gzero:N { c }
```

(*End of definition for* `\matrix_gzero:N`. *This function is documented on page* 16.)

\matrix_gset:Nn
\matrix_gset:cn

设置矩阵的值。

```
\cs_new_protected:Nn \matrix_gset:Nn
  {
    \seq_if_in:NVTF \g__matrix_const_seq #1
      { \msg_error:nnn { matrix } { constant-matrix } {#1} }
      {
        \__matrix_meta_parse:N #1
        \int_compare:nNnTF
          { \clist_count:n {#2} } > { \l__matrix_length_int }
          { \msg_error:nnn { matrix } { too-many-elements } {#2} }
          { \__matrix_space_query: \__matrix_data_gset:n {#2} }
      }
  }
\cs_generate_variant:Nn \matrix_gset:Nn { c, Ne, ce }
```

(*End of definition for* `\matrix_gset:Nn`. *This function is documented on page* 17.)

6.1.3 矩阵条件句

__matrix_fp_case:nnnnnn

浮点数条件分支。#1 要判定的浮点数，#2 为零的动作，#3 为 1 的动作，#4 为 -1 的动作，#5 为正且非 1 的动作，#6 为负且非 -1 的动作。

```
\cs_set:Nn \__matrix_fp_case:nnnnnn
  {
    \fp_compare:nNnTF { abs(#1) } < { 1e-12 }
      {#2}
      {
        \fp_compare:nNnTF {#2} > { 0 }
          { \fp_compare:nNnTF { abs(#1 - 1) } < { 1e-12 } {#3} {#5} }
          { \fp_compare:nNnTF { abs(#1 + 1) } < { 1e-12 } {#4} {#6} }
      }
  }
```

(*End of definition for* `\__matrix_fp_case:nnnnnn`.)

__matrix_fp_case:nnnn

浮点数条件分支。#1 要判定的浮点数，#2 为零的动作，#3 为 1 的动作，#4 其余情况。

```
\cs_set:Nn \__matrix_fp_case:nnnn
  {
    \fp_compare:nNnTF { abs(#1) } < { 1e-12 }
      {#2}
      { \fp_compare:nNnTF { abs(#1 - 1) } < { 1e-12 } {#3} {#4} }
  }
```

(*End of definition for* `\__matrix_fp_case:nnnn`.)

`\__matrix_fp_if_eq_p:nn` 浮点数相等条件句。#1 浮点数 1，#2 浮点数 2。

`\__matrix_fp_if_eq:nnTF`

```
\prg_set_conditional:Nnn  \__matrix_fp_if_eq:nn { p, T, F, TF }
  {
    \fp_compare:nNnTF { abs(#1 - #2) } < { 1e-12 }
      { \prg_return_true:  }
      { \prg_return_false: }
  }
```

(*End of definition for* `\__matrix_fp_if_eq:nnTF`.)

纯维度相关的矩阵条件句。

`\matrix_if_empty_p:N`
`\matrix_if_empty:NTF`
`\matrix_if_square_p:N`
`\matrix_if_square:NTF`
`\matrix_if_same_dim_p:NN`
`\matrix_if_same_dim:NNTF`
`\matrix_if_same_dim_p:Nnn`
`\matrix_if_same_dim:NnnTF`

```
\prg_set_conditional:Nnn \matrix_if_empty:N  { p, T, F, TF }
  {
    \bool_if:nTF
      { \exp_last_unbraced:NV \__matrix_meta_if_empty:w #1 }
      { \prg_return_true:  }
      { \prg_return_false: }
  }
\prg_set_conditional:Nnn \matrix_if_square:N { p, T, F, TF }
  {
    \bool_if:nTF
      { \exp_last_unbraced:NV \__matrix_meta_if_square:w #1 }
      { \prg_return_true:  }
      { \prg_return_false: }
  }
\prg_set_conditional:Nnn \matrix_if_same_dim:NN  { p, T, F, TF }
  {
    \bool_if:nTF
      { \exp_last_unbraced:Ne \__matrix_meta_if_same_dim:w  { #1#2 } }
      { \prg_return_true:  }
      { \prg_return_false: }
  }
\prg_set_conditional:Nnn \matrix_if_same_dim:Nnn { p, T, F, TF }
  {
    \__matrix_meta_gset:Nnnnn \g__matrix_tmpa_tl {#2*#3} {#2} {#3} {0}
    \bool_if:nTF
      {
        \exp_last_unbraced:Ne \__matrix_meta_if_same_dim:w
          { #1 \g__matrix_tmpa_tl }
      }
      { \prg_return_true:  }
```

```
      { \prg_return_false: }
  }
```

(*End of definition for* `\matrix_if_empty:NTF` *and others. These functions are documented on page* 18.)

`\matrix_if_zero:NTF` 指定矩阵是否为零矩阵。

```
\prg_set_conditional:Nnn \matrix_if_zero:N { T, F, TF }
  {
    \__matrix_meta_parse:N #1
    \int_compare:nNnTF { \l__matrix_length_int } = { 0 }
      { \prg_return_false: }
      {
        \__matrix_space_query:
        \int_zero:N \l__matrix_tmpa_int
        \int_step_inline:nn { \l__matrix_length_int }
          {
            \fp_set:Nn \l__matrix_tmpa_fp
              { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + ##1 } }
            \__matrix_fp_if_eq:nnF { \l__matrix_tmpa_fp } { 0 }
              { \int_incr:N \l__matrix_tmpa_int \prg_break: }
          }
        \prg_break_point:
        \int_compare:nNnTF { \l__matrix_tmpa_int } = { 0 }
          { \prg_return_true:  }
          { \prg_return_false: }
      }
  }
```

(*End of definition for* `\matrix_if_zero:NTF`. *This function is documented on page* 19.)

`\matrix_if_identity:NTF` 指定矩阵是否为单位矩阵。

```
\prg_set_conditional:Nnn \matrix_if_identity:N { T, F, TF }
  {
    \__matrix_meta_parse:N #1
    \bool_lazy_and:nnTF
      { \int_compare_p:nNn \l__matrix_length_int > { 0 } }
      { \int_compare_p:nNn \l__matrix_row_int = \l__matrix_col_int }
      {
        \__matrix_space_query:
        \int_zero:N \l__matrix_tmpa_int
        \int_step_inline:nn { \l__matrix_length_int }
          {
            \fp_set:Nn \l__matrix_tmpa_fp
              { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + ##1 } }
```

```
            \int_compare:nNnTF
              { \int_mod:nn {##1} { \l__matrix_col_int + 1 } } = {1}
              { \__matrix_fp_if_eq:nnF { \l__matrix_tmpa_fp } { 1 } }
              { \__matrix_fp_if_eq:nnF { \l__matrix_tmpa_fp } { 0 } }
              { \int_incr:N \l__matrix_tmpa_int \prg_break: }
          }
        \prg_break_point:
        \int_compare:nNnTF { \l__matrix_tmpa_int } = { 0 }
          { \prg_return_true:  }
          { \prg_return_false: }
      }
      { \prg_return_false: }
  }
```

(*End of definition for* `\matrix_if_identity:NTF`. *This function is documented on page 19.*)

6.1.4 显示矩阵

`\matrix_use:N` `\matrix_use:c` 显示矩阵。

```
\cs_new_protected:Nn \matrix_use:N
  {
    \__matrix_meta_parse:N #1
    \int_compare:nNnTF
      { \l__matrix_length_int } = { 0 }
      { \left(~<\int_use:N \l__matrix_row_int>~x~<\int_use:N \l__matrix_col_int>~\right) }
      {
        \__matrix_space_query:
        \__matrix_data_typeset:Nnnn \l__matrix_fparray
          { \l__matrix_offset_int } { \l__matrix_length_int } { \l__matrix_col_int }
      }
  }
\cs_generate_variant:Nn \matrix_use:N { c }
```

(*End of definition for* `\matrix_use:N`. *This function is documented on page 17.*)

`\matrix_log:N` 查看矩阵。

```
\cs_new_protected:Nn \matrix_log:N
  {
    \__matrix_meta_parse:N #1
    \iow_log:n { The~matrix's~base~information:~ }
    \iow_log:V #1
    \int_compare:nNnTF { \l__matrix_length_int } > { 0 }
      {
```

```
      \__matrix_space_query:nn { \l__matrix_length_int } { \l__matrix_index_int }
      \iow_log:e { The~matrix's~data: <offset:~\int_use:N \l__matrix_offset_int>~ }
      \__matrix_data_log:Nnnnn \l__matrix_fparray
        { \l__matrix_offset_int } { \l__matrix_length_int }
        { \l__matrix_col_int  } { \l__matrix_space_size_int }
    }
    { \iow_log:n { The~matrix~does~not~has~any~data~! } }
  }
\cs_generate_variant:Nn \matrix_log:N { c }
```

(*End of definition for* `\matrix_log:N`. *This function is documented on page* 17.)

6.1.5 预定义矩阵

`\g_tmpa_matrix` `\g_tmpb_matrix`

本宏包永不使用的临时变量。

```
\matrix_new:Nnn  \g_tmpa_matrix { 0 } { 0 }
\matrix_new:Nnn  \g_tmpb_matrix { 0 } { 0 }
```

(*End of definition for* `\g_tmpa_matrix` *and* `\g_tmpb_matrix`. *These functions are documented on page* 18.)

`\c_empty_matrix` `\c_identity_two_matrix` `\c_identity_three_matrix` `\c_identity_four_matrix`

定义长度为零的特殊矩阵，零索引不指向任何实际数据。

```
\matrix_const:Nnnn \c_empty_matrix
  { 0 } { 0 } {}
\matrix_const:Nnnn \c_identity_two_matrix
  { 2 } { 2 } { 1, 0, 0, 1 }
\matrix_const:Nnnn \c_identity_three_matrix
  { 3 } { 3 } { 1, 0, 0, 0, 1, 0, 0, 0, 1 }
\matrix_const:Nnnn \c_identity_four_matrix
  { 4 } { 4 }
  {
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
  }
```

(*End of definition for* `\c_empty_matrix` *and others. These functions are documented on page* 17.)

6.2 映射与运算

6.2.1 运算专用变量

`\l__matrix_opa_row_int` `\l__matrix_opb_row_int` `\l__matrix_opa_col_int` `\l__matrix_opb_col_int` `\l__matrix_old_length_int` `\l__matrix_old_index_int`

矩阵运算时使用的临时元数据。

```
\int_new:N  \l__matrix_opa_row_int
```

```
\int_new:N  \l__matrix_opb_row_int
\int_new:N  \l__matrix_opa_col_int
\int_new:N  \l__matrix_opb_col_int
\int_new:N  \l__matrix_old_length_int
\int_new:N  \l__matrix_old_index_int
```

(*End of definition for* `\l__matrix_opa_row_int` *and others.*)

`\l__matrix_opa_fparray` 矩阵运算时使用的临时数组信息。

`\l__matrix_opb_fparray`

`\l__matrix_opa_offset_int`

`\l__matrix_opb_offset_int`

```
\cs_new:Npn \l__matrix_opa_fparray {}
\cs_new:Npn \l__matrix_opb_fparray {}
\int_new:N  \l__matrix_opa_offset_int
\int_new:N  \l__matrix_opb_offset_int
```

(*End of definition for* `\l__matrix_opa_fparray` *and others.*)

6.2.2 运算准备与收尾

`\__matrix_left_operand_prepare:N` 操作数准备。

`\__matrix_right_operand_prepare:N`

```
\cs_new:Nn \__matrix_left_operand_prepare:N
  {
    \__matrix_meta_parse:N #1
    \int_set_eq:NN \l__matrix_opa_row_int    \l__matrix_row_int
    \int_set_eq:NN \l__matrix_opa_col_int    \l__matrix_col_int
    \__matrix_space_query:
    \cs_set_eq:NN  \l__matrix_opa_fparray    \l__matrix_fparray
    \int_set_eq:NN \l__matrix_opa_offset_int   \l__matrix_offset_int
  }
\cs_new:Nn \__matrix_right_operand_prepare:N
  {
    \__matrix_meta_parse:N #1
    \int_set_eq:NN \l__matrix_opb_row_int    \l__matrix_row_int
    \int_set_eq:NN \l__matrix_opb_col_int    \l__matrix_col_int
    \__matrix_space_query:
    \cs_set_eq:NN   \l__matrix_opb_fparray    \l__matrix_fparray
    \int_set_eq:NN  \l__matrix_opb_offset_int   \l__matrix_offset_int
  }
\cs_generate_variant:Nn \__matrix_left_operand_prepare:N  { c }
\cs_generate_variant:Nn \__matrix_right_operand_prepare:N { c }
```

(*End of definition for* `\__matrix_left_operand_prepare:N` *and* `\__matrix_right_operand_prepare:N`.)

`\__matrix_inplace_prepare:N` 原地修改准备，清空旧的空间信息以误回收。

```
\cs_new:Nn \__matrix_inplace_prepare:N
```

```
  {
    \seq_if_in:NVTF \g__matrix_const_seq #1
      {
        \msg_error:nnn { matrix } { constant-matrix } {#1}
        \matrix_map_break:
      }
      {
        \__matrix_meta_parse:N #1
        \__matrix_space_query:
        \int_zero:N \l__matrix_old_length_int
        \int_zero:N \l__matrix_old_index_int
      }
  }
\cs_generate_variant:Nn \__matrix_inplace_prepare:N { c }
```

(*End of definition for* `\__matrix_inplace_prepare:N`.)

`\__matrix_result_prepare:Nnn` 结果准备，额外提供行列数，需要重新申请空间。旧空间不保留暂不回收。

```
\cs_new:Nn \__matrix_result_prepare:Nnn
  {
    \seq_if_in:NVTF \g__matrix_const_seq #1
      {
        \msg_error:nnn { matrix } { constant-matrix } {#1}
        \matrix_map_break:
      }
      {
        \__matrix_meta_parse:N #1
        \int_set_eq:NN \l__matrix_old_length_int \l__matrix_length_int
        \int_set_eq:NN \l__matrix_old_index_int  \l__matrix_index_int
        \int_set:Nn \l__matrix_row_int {#2}
        \int_set:Nn \l__matrix_col_int {#3}
        \int_set:Nn \l__matrix_length_int { #2 * #3 }
        \__matrix_space_allocate:
        \__matrix_meta_gset:N #1
        \__matrix_space_query:
        \int_step_inline:nnn { \l__matrix_length_int + 1 } { \l__matrix_space_size_int }
          { \fparray_gset:Nnn \l__matrix_fparray { \l__matrix_offset_int + ##1 } { 0 } }
      }
  }
\cs_generate_variant:Nn \__matrix_result_prepare:Nnn { c }
```

(*End of definition for* `\__matrix_result_prepare:Nnn`.)

`\__matrix_operation_cleanup:` 运算收尾。

```
\cs_new:Nn \__matrix_operation_cleanup:
  {
    \__matrix_space_free:nn
      { \l__matrix_old_length_int } { \l__matrix_old_index_int }
    \int_zero:N \l__matrix_old_length_int
    \int_zero:N \l__matrix_old_index_int
  }
```

(*End of definition for* `\__matrix_operation_cleanup:`.)

6.2.3 通用映射函数

`\g__matrix_map_depth_int` 映射嵌套深度变量。

```
\int_new:N  \g__matrix_map_depth_int
```

(*End of definition for* `\g__matrix_map_depth_int`.)

`\g__matrix_map_intarray` 有限的嵌套变量。

```
\intarray_new:Nn \g__matrix_map_intarray { 400 }
```

(*End of definition for* `\g__matrix_map_intarray`.)

`\__matrix_map_state_save:` 有限的嵌套变量的保存与恢复。

`\__matrix_map_state_restore:`

```
\cs_new_protected:Nn \__matrix_map_state_save:
  {
    \int_set:Nn \l__matrix_tmpa_int { (\g__matrix_map_depth_int - 2) * 10 }
    \intarray_gset:Nnn \g__matrix_map_intarray
      { \l__matrix_tmpa_int + 1 } { \l__matrix_length_int }
    \intarray_gset:Nnn \g__matrix_map_intarray
      { \l__matrix_tmpa_int + 2 } { \l__matrix_row_int }
    \intarray_gset:Nnn \g__matrix_map_intarray
      { \l__matrix_tmpa_int + 3 } { \l__matrix_col_int }
    \intarray_gset:Nnn \g__matrix_map_intarray
      { \l__matrix_tmpa_int + 4 } { \l__matrix_index_int }
    \intarray_gset:Nnn \g__matrix_map_intarray
      { \l__matrix_tmpa_int + 5 } { \l__matrix_offset_int }
  }
\cs_new_protected:Nn \__matrix_map_state_restore:
  {
    \int_set:Nn \l__matrix_tmpa_int { (\g__matrix_map_depth_int - 2) * 10 }
    \int_set:Nn \l__matrix_length_int
      { \intarray_item:Nn \g__matrix_map_intarray { \l__matrix_tmpa_int + 1 } }
    \int_set:Nn \l__matrix_row_int
      { \intarray_item:Nn \g__matrix_map_intarray { \l__matrix_tmpa_int + 2 } }
```

```
    \int_set:Nn \l__matrix_col_int
      { \intarray_item:Nn \g__matrix_map_intarray { \l__matrix_tmpa_int + 3 } }
    \int_set:Nn \l__matrix_index_int
      { \intarray_item:Nn \g__matrix_map_intarray { \l__matrix_tmpa_int + 4 } }
    \int_set:Nn \l__matrix_offset_int
      { \intarray_item:Nn \g__matrix_map_intarray { \l__matrix_tmpa_int + 5 } }
  }
```

(*End of definition for* `\__matrix_map_state_save:` *and* `\__matrix_map_state_restore:`.)

`\__matrix_map_item:n`
`\__matrix_map_push:`
`\__matrix_map_pop:`

当前映射元素处理函数。

```
\cs_new:Npn \__matrix_map_item:n #1 { }
```

保存当前元素处理函数。

```
\cs_new_protected:Nn \__matrix_map_push:
  {
    \int_gincr:N \g__matrix_map_depth_int
    \int_compare:nNnT
      { \g__matrix_map_depth_int } > { 1 }
      {
        \cs_gset_eq:cN
          { __matrix_map_item_ \int_use:N \g__matrix_map_depth_int :n }
          \__matrix_map_item:n
        \cs_gset_eq:cN
          { g__matrix_map_\int_use:N \g__matrix_map_depth_int _fparray }
          \l__matrix_fparray
        \__matrix_map_state_save:
      }
  }
```

恢复上一级元素处理函数。

```
\cs_new_protected:Nn \__matrix_map_pop:
  {
    \int_compare:nNnT
      { \g__matrix_map_depth_int } > { 1 }
      {
        \cs_gset_eq:Nc \__matrix_map_item:n
          { __matrix_map_item_ \int_use:N \g__matrix_map_depth_int :n }
        \cs_set_eq:Nc \l__matrix_fparray
          { g__matrix_map_\int_use:N \g__matrix_map_depth_int _fparray }
        \__matrix_map_state_restore:
      }
    \int_gdecr:N \g__matrix_map_depth_int
  }
```

(*End of definition for* `\__matrix_map_item:n`, `\__matrix_map_push:`, *and* `\__matrix_map_pop:`.)

`\matrix_map_break:` 定义中断函数。

`\matrix_map_break:n`

```
\cs_new_protected:Nn \matrix_map_break:
  { \prg_map_break:Nn \matrix_map_break: {  } }
\cs_new_protected:Nn \matrix_map_break:n
  { \prg_map_break:Nn \matrix_map_break: {#1} }
```

(*End of definition for* `\matrix_map_break:` *and* `\matrix_map_break:n`. *These functions are documented on page* 22.)

`\matrix_map_function:NN` 函数式映射函数。

```
\cs_new_protected:Nn \matrix_map_function:NN
  {
    \__matrix_map_push:
    \__matrix_meta_parse:N #1
    \int_compare:nNnT { \l__matrix_length_int } = { 0 }
      { \matrix_map_break: }
    \__matrix_space_query:
    \int_step_inline:nn { \l__matrix_length_int }
      {
        \exp_args:Ne #2
          { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + ##1 } }
      }
    \prg_break_point:Nn \matrix_map_break: { \__matrix_map_pop: }
  }
```

(*End of definition for* `\matrix_map_function:NN`. *This function is documented on page* 22.)

`\matrix_map_inline:Nn` 内联式映射函数。

```
\cs_new_protected:Nn \matrix_map_inline:Nn
  {
    \__matrix_map_push:
    \__matrix_meta_parse:N #1
    \cs_gset:Nn \__matrix_map_item:n {#2}
    \int_compare:nNnT { \l__matrix_length_int } = { 0 }
      { \matrix_map_break: }
    \__matrix_space_query:
    \int_step_inline:nn { \l__matrix_length_int }
      {
        \exp_args:Ne \__matrix_map_item:n
          { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + ##1 } }
      }
    \prg_break_point:Nn \matrix_map_break: { \__matrix_map_pop: }
```

```
  }
```

(*End of definition for* `\matrix_map_inline:Nn`. *This function is documented on page* 22.)

6.2.4 原地映射运算

`\__matrix_map_inplace:Nn` 通过映射修改矩阵的元素。

```
\cs_new:Nn \__matrix_map_inplace:Nn
  {
    \__matrix_map_push:
    \__matrix_inplace_prepare:N #1
    \int_compare:nNnT { \l__matrix_length_int } = { 0 }
      { \matrix_map_break: }
    \cs_gset:Nn \__matrix_map_item:n {#2}
    \int_step_inline:nn { \l__matrix_length_int }
      {
        \fparray_gset:Nnn \l__matrix_fparray { \l__matrix_offset_int + ##1 }
          {
            \exp_args:Ne \__matrix_map_item:n
              { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + ##1 } }
          }
      }
    \prg_break_point:Nn \matrix_map_break: { \__matrix_map_pop: }
  }
```

(*End of definition for* `\__matrix_map_inplace:Nn`.)

`\matrix_gnegate:N` 求 $-A$、$|A|$、$sign(A)$ 与 $round(A,n)$。
`\matrix_gabs:N`
`\matrix_gsign:N`
`\matrix_ground:Nn`

```
\cs_new_protected:Nn \matrix_gnegate:N
  { \__matrix_map_inplace:Nn #1 { -##1 } }
\cs_new_protected:Nn \matrix_gabs:N
  { \__matrix_map_inplace:Nn #1 { abs(##1) }  }
\cs_new_protected:Nn \matrix_gsign:N
  { \__matrix_map_inplace:Nn #1 { sign(##1) } }
\cs_new_protected:Nn \matrix_ground:Nn
  { \__matrix_map_inplace:Nn #1 { round(##1, #2) } }
```

(*End of definition for* `\matrix_gnegate:N` *and others. These functions are documented on page* 19.)

`\matrix_gadd:Nn` 求 $A+c$、$A-c$、cA、A/c 及 A^c。
`\matrix_gsub:Nn`
`\matrix_gmul:Nn`
`\matrix_gdiv:Nn`
`\matrix_gpower:Nn`

```
\cs_new_protected:Nn \matrix_gadd:Nn
  { \__matrix_map_inplace:Nn #1 { ##1 + #2 } }
\cs_new_protected:Nn \matrix_gsub:Nn
  { \__matrix_map_inplace:Nn #1 { ##1 - #2 } }
```

```
\cs_new_protected:Nn \matrix_gmul:Nn
  { \__matrix_map_inplace:Nn #1 { ##1 * #2 } }
\cs_new_protected:Nn \matrix_gdiv:Nn
  { \__matrix_map_inplace:Nn #1 { ##1 / #2 } }
\cs_new_protected:Nn \matrix_gpower:Nn
  { \__matrix_map_inplace:Nn #1 { ##1 ^ #2 } }
```

(*End of definition for* `\matrix_gadd:Nn` *and others. These functions are documented on page 20.*)

6.2.5 单矩阵映射运算

`\__matrix_map_operate:NNn` 通过映射逐元素计算，操作数只有一个矩阵。

```
\cs_new:Nn \__matrix_map_operate:NNn
  {
    \__matrix_map_push:
    \__matrix_left_operand_prepare:N #2
    \__matrix_result_prepare:Nnn #1 { \l__matrix_opa_row_int } { \l__matrix_opa_col_int }
    \int_compare:nNnT { \l__matrix_length_int } = { 0 }
      { \matrix_map_break: }
    \cs_gset:Nn \__matrix_map_item:n {#3}
    \int_step_inline:nn { \l__matrix_length_int }
      {
        \fparray_gset:Nnn \l__matrix_fparray { \l__matrix_offset_int + ##1 }
          {
            \exp_args:Ne \__matrix_map_item:n
              { \fparray_item:Nn \l__matrix_opa_fparray { \l__matrix_opa_offset_int + ##1 }
          }
      }
    \prg_break_point:Nn \matrix_map_break:
      { \__matrix_map_pop: \__matrix_operation_cleanup: }
  }
```

(*End of definition for* `\__matrix_map_operate:NNn`.)

`\matrix_gset_eq:NN` `\matrix_gset_eq:Nc` `\matrix_gset_eq:cN` 矩阵赋值。

```
\cs_new_protected:Nn \matrix_gset_eq:NN
  { \__matrix_map_operate:NNn #1 #2 {##1} }
\cs_generate_variant:Nn \matrix_gset_eq:NN { c, Nc }
```

(*End of definition for* `\matrix_gset_eq:NN`. *This function is documented on page 17.*)

`\matrix_gnegate:NN` `\matrix_gabs:NN` `\matrix_gsign:NN` `\matrix_ground:NNn` 求 $Z=-A$、$Z=|A|$ 与 $Z=sign(A)$ 与 $Z=round(A,n)$。

```
\cs_new_protected:Nn \matrix_gnegate:NN
  { \__matrix_map_operate:NNn #1 #2 {-##1} }
```

```
\cs_new_protected:Nn \matrix_gabs:NN
  { \__matrix_map_operate:NNn #1 #2 { abs(##1) }  }
\cs_new_protected:Nn \matrix_gsign:NN
  { \__matrix_map_operate:NNn #1 #2 { sign(##1) } }
\cs_new_protected:Nn \matrix_ground:NNn
  { \__matrix_map_operate:NNn #1 #2 { round(##1, #3) } }
```

(*End of definition for* `\matrix_gnegate:NN` *and others. These functions are documented on page 20.*)

`\matrix_gadd:Nn` `\matrix_gsub:Nn` `\matrix_gmul:Nn` `\matrix_gdiv:Nn` `\matrix_gpower:Nn`

求 $Z = A + c$、$Z = A - c$、$Z = cA$、$Z = A/c$ 及 $Z = A^c$。

```
\cs_new_protected:Nn \matrix_gadd:NNn
  { \__matrix_map_operate:NNn #1 #2 { ##1 + #3 } }
\cs_new_protected:Nn \matrix_gsub:NNn
  { \__matrix_map_operate:NNn #1 #2 { ##1 - #3 } }
\cs_new_protected:Nn \matrix_gmul:NNn
  { \__matrix_map_operate:NNn #1 #2 { ##1 * #3 } }
\cs_new_protected:Nn \matrix_gdiv:NNn
  { \__matrix_map_operate:NNn #1 #2 { ##1 / #3 } }
\cs_new_protected:Nn \matrix_gpower:NNn
  { \__matrix_map_operate:NNn #1 #2 { ##1 ^ #3 } }
```

(*End of definition for* `\matrix_gadd:Nn` *and others. These functions are documented on page 20.*)

6.2.6 双矩阵映射运算

`\__matrix_map_operate:NNNn` 通过映射逐元素计算，操作数有两个矩阵。

```
\cs_new:Nn \__matrix_map_operate:NNNn
  {
    \__matrix_map_push:
    \__matrix_left_operand_prepare:N  #2
    \__matrix_right_operand_prepare:N #3
    \bool_lazy_and:nnF
      { \int_compare_p:nNn \l__matrix_opa_row_int = \l__matrix_opb_row_int }
      { \int_compare_p:nNn \l__matrix_opa_col_int = \l__matrix_opb_col_int }
      {
        \msg_error:nne { matrix } { dimension-mismatch }
          {
            (\int_use:N \l__matrix_opa_row_int, \int_use:N \l__matrix_opa_col_int) vs.
            (\int_use:N \l__matrix_opb_row_int, \int_use:N \l__matrix_opb_col_int)
          }
        \matrix_map_break:
      }
    \__matrix_result_prepare:Nnn #1 { \l__matrix_opa_row_int } { \l__matrix_opa_col_int }
```

```
    \int_compare:nNnT { \l__matrix_length_int } = { 0 }
      { \matrix_map_break: }
    \cs_gset:Npn \__matrix_map_item:n ##1 ##2 {#4}
    \int_step_inline:nn { \l__matrix_length_int }
      {
        \fparray_gset:Nnn \l__matrix_fparray { \l__matrix_offset_int + ##1 }
          {
            \exp_args:Nee \__matrix_map_item:n
              { \fparray_item:Nn \l__matrix_opa_fparray { \l__matrix_opa_offset_int + ##1 }
              { \fparray_item:Nn \l__matrix_opb_fparray { \l__matrix_opb_offset_int + ##1 }
          }
      }
    \prg_break_point:Nn \matrix_map_break:
      { \__matrix_map_pop: \__matrix_operation_cleanup: }
  }
```

(*End of definition for* `\__matrix_map_operate:NNNn`.)

`\matrix_gadd:NNN` `\matrix_gsub:NNN` 求 $Z = A + B$、$Z = A - B$

```
\cs_new_protected:Nn \matrix_gadd:NNN
  { \__matrix_map_operate:NNNn #1 #2 #3 { ##1 + ##2 } }
\cs_new_protected:Nn \matrix_gsub:NNN
  { \__matrix_map_operate:NNNn #1 #2 #3 { ##1 - ##2 } }
```

(*End of definition for* `\matrix_gadd:NNN` *and* `\matrix_gsub:NNN`. *These functions are documented on page 21.*)

`\matrix_gadd:NN` `\matrix_gsub:NN` 求 $A + B$、$A - B$。

```
\cs_new_protected:Nn \matrix_gadd:NN
  { \__matrix_map_operate:NNNn #1 #1 #2 { ##1 + ##2 } }
\cs_new_protected:Nn \matrix_gsub:NN
  { \__matrix_map_operate:NNNn #1 #1 #2 { ##1 - ##2 } }
```

(*End of definition for* `\matrix_gadd:NN` *and* `\matrix_gsub:NN`. *These functions are documented on page 21.*)

6.3 非映射运算

`\__matrix_multiply_aux:nn` 计算 $Z_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$。注意，为减少运算，此处代码坐标从 0 计。

```
\cs_new:Nn \__matrix_multiply_aux:nn
  {
    \fp_zero:N \l__matrix_tmpa_fp
    \int_step_inline:nn { \l__matrix_opa_col_int }
      {
        \fp_add:Nn \l__matrix_tmpa_fp
          {
```

```
            \fparray_item:Nn \l__matrix_opa_fparray
              { \l__matrix_opa_offset_int + (  #1 - 1 ) * \l__matrix_opa_col_int + ##1 }
            *
            \fparray_item:Nn \l__matrix_opb_fparray
              { \l__matrix_opb_offset_int + ( ##1 - 1 ) * \l__matrix_opb_col_int +  #2 }
          }
      }
    \fparray_gset:Nnn \l__matrix_fparray
      { \l__matrix_offset_int + ( #1 - 1 ) * \l__matrix_col_int + #2 }
      { \l__matrix_tmpa_fp }
  }
```

(*End of definition for* `\__matrix_multiply_aux:nn`.)

`\matrix_gmul:NNN` 计算 $Z = AB$。

```
\cs_new_protected:Nn \matrix_gmul:NNN
  {
    \group_begin:
    \__matrix_left_operand_prepare:N  #2
    \__matrix_right_operand_prepare:N #3
    \int_compare:nNnF
      { \l__matrix_opa_col_int } = { \l__matrix_opb_row_int }
      {
        \msg_error:nne { matrix } { dimension-mismatch }
          {
            (\int_use:N \l__matrix_opa_row_int, \int_use:N \l__matrix_opa_col_int) vs.
            (\int_use:N \l__matrix_opb_row_int, \int_use:N \l__matrix_opb_col_int)
          }
        \matrix_map_break:
      }
    \__matrix_result_prepare:Nnn #1 { \l__matrix_opa_row_int } { \l__matrix_opb_col_int }
    \int_compare:nNnT { \l__matrix_length_int } = { 0 }
      { \matrix_map_break: }
    \int_step_inline:nn { \l__matrix_opa_row_int }
      {
        \int_step_inline:nn { \l__matrix_opb_col_int }
          {
            \__matrix_multiply_aux:nn {##1} {####1}
          }
      }
    \prg_break_point:Nn \matrix_map_break:
      { \__matrix_operation_cleanup: }
    \group_end:
```

```
  }
```

(*End of definition for* `\matrix_gmul:NNN`. *This function is documented on page* 21.)

`\matrix_gmul:NN` 求 AB。

```
\cs_new_protected:Nn \matrix_gmul:NN
  { \matrix_gmul:NNN #1 #1 #2 }
```

(*End of definition for* `\matrix_gmul:NN`. *This function is documented on page* 21.)

7 几何变换

7.1 提取变换

`\__matrix_append_term:Nnn` 向变换求值公式追加项目。

```
\cs_new:Nn \__matrix_append_term:Nnn
  {
    \fp_compare:nNnF { abs(#2) } < { 1e-12 }
      {
        \fp_compare:nNnTF {#2} < { 0 }
          {
            \fp_compare:nNnTF { abs(#2 + 1) } < { 1e-12 }
              { \tl_put_right:Nn #1 {    - #3 } }
              { \tl_put_right:Nn #1 { #2 * #3 } }
          }
          {
            \tl_if_empty:NF #1
              { \tl_put_right:Nn #1 { + } }
            \fp_compare:nNnTF { abs(#2 - 1) } < { 1e-12 }
              { \tl_put_right:Nn #1 {      #3 } }
              { \tl_put_right:Nn #1 { #2 * #3 } }
          }
      }
  }
```

(*End of definition for* `\__matrix_append_term:Nnn`.)

`\__matrix_set_transform:Nnnnnn` 设置二维变换的某个变换函数。

```
\cs_new:Nn \__matrix_set_transform:Nnnnnn
  {
    \tl_clear:N \l__matrix_tmpa_tl
    \tl_clear:N \l__matrix_tmpb_tl
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {1}  {#4}
```

```
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {#2} {##1}
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {#3} {##2}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {1}  {1}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {#5} {##1}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {#6} {##2}
    \cs_set:Npe #1 ##1 ##2
      {
        \str_if_eq:VnTF \l__matrix_tmpb_tl {1}
          { \exp_not:N \fp_eval:n { \l__matrix_tmpa_tl } }
          {
            \exp_not:N \fp_eval:n
              { (\l__matrix_tmpa_tl) / (\l__matrix_tmpb_tl) }
          }
      }
  }
  \cs_generate_variant:Nn \__matrix_set_transform:Nnnnnn { Neeeee }
```

(*End of definition for* `\__matrix_set_transform:Nnnnnn`.)

`\__matrix_set_transform:Nnnnnnnn` 设置三维变换的某个变换函数。

```
\cs_new:Nn \__matrix_set_transform:Nnnnnnnn
  {
    \tl_clear:N \l__matrix_tmpa_tl
    \tl_clear:N \l__matrix_tmpb_tl
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {1}  {#5}
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {#2} {##1}
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {#3} {##2}
    \__matrix_append_term:Nnn \l__matrix_tmpa_tl {#4} {##3}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {1}  {1}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {#6} {##1}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {#7} {##2}
    \__matrix_append_term:Nnn \l__matrix_tmpb_tl {#8} {##3}
    \cs_set:Npe #1 ##1 ##2 ##3
      {
        \str_if_eq:VnTF \l__matrix_tmpb_tl {1}
          { \exp_not:N \fp_eval:n { \l__matrix_tmpa_tl } }
          {
            \exp_not:N \fp_eval:n
              { (\l__matrix_tmpa_tl) / (\l__matrix_tmpb_tl) }
          }
      }
  }
\cs_generate_variant:Nn \__matrix_set_transform:Nnnnnnnn { Neeeeeee }
```

(*End of definition for* `\__matrix_set_transform:Nnnnnnnn`.)

`\matrix_extract_transform:NNN` 提取二维变换。`#1` 变换矩阵，`#2#3` 变换函数。

```
\cs_new_protected:Nn \matrix_extract_transform:NNN
  {
    \__matrix_meta_parse:N #1
    \bool_lazy_and:nnF
      { \int_compare_p:nNn { \l__matrix_row_int } = {3} }
      { \int_compare_p:nNn { \l__matrix_col_int } = {3} }
      {
        \msg_error:nnn { matrix } { invalid-transformation-matrix } {#1}
        \prg_break:
      }
      \__matrix_space_query:
      \__matrix_fp_if_eq:nnF
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 9 } } {1}
        {
          \msg_error:nnn { matrix } { invalid-transformation-matrix } {#1}
          \prg_break:
        }
      \__matrix_set_transform:Neeeee #2
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 1 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 2 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 3 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 7 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 8 } }
      \__matrix_set_transform:Neeeee #3
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 4 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 5 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 6 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 7 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 8 } }
    \prg_break_point:
  }
```

(*End of definition for* `\matrix_extract_transform:NNN`. *This function is documented on page* **??**.)

`\matrix_extract_transform:NNNN` 提取三维变换。`#1` 变换矩阵，`#2#3#4` 变换函数。

```
\cs_new_protected:Nn \matrix_extract_transform:NNNN
  {
    \__matrix_meta_parse:N #1
    \bool_lazy_and:nnF
      { \int_compare_p:nNn { \l__matrix_row_int } = {4} }
```

```
      { \int_compare_p:nNn { \l__matrix_col_int } = {4} }
      {
        \msg_error:nnn { matrix } { invalid-transformation-matrix } {#1}
        \prg_break:
      }
      \__matrix_space_query:
      \__matrix_fp_if_eq:nnF
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 16 } } {1}
        {
          \msg_error:nnn { matrix } { invalid-transformation-matrix } {#1}
          \prg_break:
        }
      \__matrix_set_transform:Neeeeeee #2
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  1 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  2 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  3 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  4 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 13 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 14 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 15 } }
      \__matrix_set_transform:Neeeeeee #3
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  5 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  6 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  7 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  8 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 13 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 14 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 15 } }
      \__matrix_set_transform:Neeeeeee #4
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int +  9 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 10 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 11 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 12 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 13 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 14 } }
        { \fparray_item:Nn \l__matrix_fparray { \l__matrix_offset_int + 15 } }
    \prg_break_point:
  }
```

(*End of definition for* `\matrix_extract_transform:NNNN`. *This function is documented on page* ??.)

7.2 构建变换

切换变换专用前缀。

```
⟨@@=trans⟩
```

\g__trans_matrix
\g__trans_two_matrix
\g__trans_three_matrix
\g__trans_dim_int

用于存储变换矩阵的数据。

```
\matrix_new:N   \g__trans_matrix
\matrix_new:Nnn \g__trans_two_matrix   {3} {3}
\matrix_new:Nnn \g__trans_three_matrix {4} {4}
\int_new:N      \g__trans_dim_int
```

(*End of definition for* \g__trans_matrix *and others.*)

\g__trans_prop
\g__trans_tmp_matrix
\g__trans_empty_matrix

用于处理变换矩阵保存与恢复相关功能。

```
\prop_new:N     \g__trans_prop
\matrix_new:N   \g__trans_tmp_matrix
\matrix_new:N   \g__trans_empty_matrix
```

(*End of definition for* \g__trans_prop, \g__trans_tmp_matrix, *and* \g__trans_empty_matrix.)

\l__trans_sin_fp
\l__trans_cos_fp

用于计算变换矩阵的临时变量。

```
\fp_new:N       \l__trans_sin_fp
\fp_new:N       \l__trans_cos_fp
```

(*End of definition for* \l__trans_sin_fp *and* \l__trans_cos_fp.)

__trans_compose:N

将新变换矩阵作用于当前变换矩阵。#1 新变换矩阵。

```
\cs_new_protected:Nn \__trans_compose:N
  { \matrix_gmul:NNN \g__trans_matrix #1 \g__trans_matrix }
```

(*End of definition for* __trans_compose:N.)

7.2.1 初始化与复用

\transform_init:n
\TransformInit

初始化一个不做任何变换的 #1 维变换矩阵，即单位矩阵。

```
\cs_new_protected:Nn \transform_init:n
  {
    \int_case:nnTF {#1}
      {
        { 2 } { \matrix_gset_eq:NN \g__trans_matrix \c_identity_three_matrix }
        { 3 } { \matrix_gset_eq:NN \g__trans_matrix \c_identity_four_matrix }
      }
      { \int_gset:Nn \g__trans_dim_int {#1} }
      { \msg_error:nnn { matrix } { dimension-mismatch } {#1} }
  }
```

```
\NewDocumentCommand \TransformInit { O{2} }
  { \transform_init:n {#1} }
```

(*End of definition for* `\transform_init:n` *and* `\TransformInit`*. These functions are documented on page 23.*)

`\transform_save:n` `\transform_restore:n` `\TransformSave` `\TransformRestore`

保存与恢复变换矩阵。注意，保存的矩阵与当前矩阵不应当引用同一份数据，否则更新矩阵是会出问题。

```
\cs_new_protected:Nn \transform_save:n
  {
    \prop_get:NnNF \g__trans_prop {#1} \g__trans_tmp_matrix
      { \tl_gset_eq:NN \g__trans_tmp_matrix \g__trans_empty_matrix }
    \matrix_gset_eq:NN \g__trans_tmp_matrix \g__trans_matrix
    \prop_gput:NnV \g__trans_prop {#1} \g__trans_matrix
  }
\cs_new_protected:Nn \transform_restore:n
  {
    \prop_get:NnNT \g__trans_prop {#1} \g__trans_tmp_matrix
      {
        \matrix_gset_eq:NN \g__trans_matrix \g__trans_tmp_matrix
        \matrix_if_same_dim:NnnTF \g__trans_matrix {3} {3}
          { \int_gset:Nn \g__trans_dim_int {2} }
          { \int_gset:Nn \g__trans_dim_int {3} }
      }
  }
\cs_gset_eq:NN \TransformSave    \transform_save:n
\cs_gset_eq:NN \TransformRestore \transform_restore:n
```

(*End of definition for* `\transform_save:n` *and others. These functions are documented on page 23.*)

7.2.2 旋转变换

`\transform_rotate:n` `\transform_rotated:n` `\__trans_rotate_aux:nn`

绕原点旋转平面。

```
\cs_new:Nn \__trans_rotate_aux:nn
  {
    \matrix_gset:Nn \g__trans_two_matrix
      {
        #2, -#1,  0,
        #1,  #2,  0,
         0,   0,  1
      }
    \__trans_compose:N \g__trans_two_matrix
  }
\cs_generate_variant:Nn \__trans_rotate_aux:nn { VV }
```

```
\cs_new_protected:Nn \transform_rotate:n
  {
    \fp_set:Nn \l__trans_sin_fp { sin(#1) }
    \fp_set:Nn \l__trans_cos_fp { cos(#1) }
    \__trans_rotate_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
\cs_new_protected:Nn \transform_rotated:n
  {
    \fp_set:Nn \l__trans_sin_fp { sind(#1) }
    \fp_set:Nn \l__trans_cos_fp { cosd(#1) }
    \__trans_rotate_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
```

(*End of definition for* `\transform_rotate:n`, `\transform_rotated:n`, *and* `\__trans_rotate_aux:nn`. *These functions are documented on page 24.*)

`\transform_rotate_x:n`
`\transform_rotated_x:n`
`\__trans_rotate_x_aux:nn`

绕原点放置 X 轴。

```
\cs_new:Nn \__trans_rotate_x_aux:nn
  {
    \matrix_gset:Nn \g__trans_three_matrix
      {
         1,   0,   0,  0,
         0,  #2, -#1,  0,
         0,  #1,  #2,  0,
         0,   0,   0,  1
      }
    \__trans_compose:N \g__trans_three_matrix
  }
\cs_generate_variant:Nn \__trans_rotate_x_aux:nn { VV }
\cs_new_protected:Nn \transform_rotate_x:n
  {
    \fp_set:Nn \l__trans_sin_fp { sin(#1) }
    \fp_set:Nn \l__trans_cos_fp { cos(#1) }
    \__trans_rotate_x_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
\cs_new_protected:Nn \transform_rotated_x:n
  {
    \fp_set:Nn \l__trans_sin_fp { sind(#1) }
    \fp_set:Nn \l__trans_cos_fp { cosd(#1) }
    \__trans_rotate_x_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
```

(*End of definition for* `\transform_rotate_x:n`, `\transform_rotated_x:n`, *and* `\__trans_rotate_x_aux:nn`. *These functions are documented on page 24.*)

\transform_rotate_y:n
\transform_rotated_y:n
__trans_rotate_y_aux:nn

绕原点放置 Y 轴。

```
\cs_new:Nn \__trans_rotate_y_aux:nn
  {
    \matrix_gset:Nn \g__trans_three_matrix
      {
         #2,  0,  #1,  0,
          0,  1,   0,  0,
        -#1,  0,  #2,  0,
          0,  0,   0,  1
      }
    \__trans_compose:N \g__trans_three_matrix
  }
\cs_generate_variant:Nn \__trans_rotate_y_aux:nn { VV }
\cs_new_protected:Nn \transform_rotate_y:n
  {
    \fp_set:Nn \l__trans_sin_fp { sin(#1) }
    \fp_set:Nn \l__trans_cos_fp { cos(#1) }
    \__trans_rotate_y_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
\cs_new_protected:Nn \transform_rotated_y:n
  {
    \fp_set:Nn \l__trans_sin_fp { sind(#1) }
    \fp_set:Nn \l__trans_cos_fp { cosd(#1) }
    \__trans_rotate_y_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
```

(*End of definition for* `\transform_rotate_y:n`, `\transform_rotated_y:n`, *and* `\__trans_rotate_y_aux:nn`. *These functions are documented on page* 24.)

\transform_rotate_z:n
\transform_rotated_z:n
__trans_rotate_z_aux:nn

绕原点放置 Z 轴。

```
\cs_new:Nn \__trans_rotate_z_aux:nn
  {
    \matrix_gset:Nn \g__trans_three_matrix
      {
        #2, -#1,  0,  0,
        #1,  #2,  0,  0,
         0,   0,  1,  0,
         0,   0,  0,  1
      }
    \__trans_compose:N \g__trans_three_matrix
  }
\cs_generate_variant:Nn \__trans_rotate_z_aux:nn { VV }
```

```
\cs_new_protected:Nn \transform_rotate_z:n
  {
    \fp_set:Nn \l__trans_sin_fp { sin(#1) }
    \fp_set:Nn \l__trans_cos_fp { cos(#1) }
    \__trans_rotate_z_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
\cs_new_protected:Nn \transform_rotated_z:n
  {
    \fp_set:Nn \l__trans_sin_fp { sind(#1) }
    \fp_set:Nn \l__trans_cos_fp { cosd(#1) }
    \__trans_rotate_z_aux:VV \l__trans_sin_fp \l__trans_cos_fp
  }
```

(*End of definition for* `\transform_rotate_z:n`, `\transform_rotated_z:n`, *and* `\__trans_rotate_z_aux:nn`. *These functions are documented on page* 24.)

7.2.3 绽放、平移、错切

`\transform_scale:nn`
`\transform_scale:nnn`

添加缩放变换。

```
\cs_new_protected:Nn \transform_scale:nn
  {
    \matrix_gset:Nn \g__trans_two_matrix
      {
        #1,  0, 0,
         0, #2, 0,
         0,  0, 1
      }
    \__trans_compose:N \g__trans_two_matrix
  }
\cs_new_protected:Nn \transform_scale:nnn
  {
    \matrix_gset:Nn \g__trans_three_matrix
      {
        #1,  0,  0, 0,
         0, #2,  0, 0,
         0,  0, #3, 0,
         0,  0,  0, 1
      }
    \__trans_compose:N \g__trans_three_matrix
  }
```

(*End of definition for* `\transform_scale:nn` *and* `\transform_scale:nnn`. *These functions are documented on page* 24.)

\transform_shift:nn
\transform_shift:nnn

添加平移变换。

```
\cs_new_protected:Nn \transform_shift:nn
  {
    \matrix_gset:Nn \g__trans_two_matrix
      {
        1, 0, #1,
        0, 1, #2,
        0, 0,  1
      }
    \__trans_compose:N \g__trans_two_matrix
  }
\cs_new_protected:Nn \transform_shift:nnn
  {
    \matrix_gset:Nn \g__trans_three_matrix
      {
        1, 0, 0, #1,
        0, 1, 0, #2,
        0, 0, 1, #3,
        0, 0, 0,  1
      }
    \__trans_compose:N \g__trans_three_matrix
  }
```

(*End of definition for* `\transform_shift:nn` *and* `\transform_shift:nnn`. *These functions are documented on page 23.*)

\transform_shear:nn
\transform_shear:nnnnnn

添加错切变换。

```
\cs_new_protected:Nn \transform_shear:nn
  {
    \matrix_gset:Nn \g__trans_two_matrix
      {
         1, #1, 0,
        #2,  1, 0,
         0,  0, 1
      }
    \__trans_compose:N \g__trans_two_matrix
  }
\cs_new_protected:Nn \transform_shear:nnnnnn
  {
    \matrix_gset:Nn \g__trans_three_matrix
      {
         1, #1, #2, 0,
        #3,  1, #4, 0,
```

```
        #5, #6,  1, 0,
         0,  0,  0, 1
      }
    \__trans_compose:N \g__trans_three_matrix
  }
```

(*End of definition for* `\transform_shear:nn` *and* `\transform_shear:nnnnnn`*. These functions are documented on page 24.*)

7.2.4 组合变换

`\transform_rotate:nnn`
`\transform_rotated:nnn`
`\transform_scale:nnnn`
`\transform_shear:nnnn`

绕指定点旋转。#1#2 为中心点，#3 旋转角度。

```
\cs_new_protected:Nn \transform_rotate:nnn
  {
    \transform_shift:nn  {-(#1)} {-(#2)}
    \transform_rotate:n  {#3}
    \transform_shift:nn  {#1}    {#2}
  }
\cs_new_protected:Nn \transform_rotated:nnn
  {
    \transform_shift:nn  {-(#1)} {-(#2)}
    \transform_rotated:n {#3}
    \transform_shift:nn  {#1}    {#2}
  }
```

绕指定点缩放或错切。#1#2 为中心点，#3#4 变换参数。

```
\cs_new_protected:Nn \transform_scale:nnnn
  {
    \transform_shift:nn  {-(#1)} {-(#2)}
    \transform_scale:nn  {#3}    {#4}
    \transform_shift:nn  {#1}    {#2}
  }
\cs_new_protected:Nn \transform_shear:nnnn
  {
    \transform_shift:nn {-(#1)} {-(#2)}
    \transform_shear:nn {#3}    {#4}
    \transform_shift:nn {#1}    {#2}
  }
```

(*End of definition for* `\transform_rotate:nnn` *and others. These functions are documented on page 24.*)

`\__trans_define_rotate_iii:NN`
`\__trans_define_rotate_iii:n`

三维组合变换辅助定义函数。

```
\cs_new:Nn \__trans_define_rotate_iii:NN
  {
```

```
    \cs_new_protected:Npn #1 ##1##2##3##4
      {
        \transform_shift:nnn {-(##1)} {-(##2)} {-(##3)}
        #2                   {##4}
        \transform_shift:nnn {##1}    {##2}    {##3}
      }
  }
 \cs_generate_variant:Nn \__trans_define_rotate_iii:NN { cc }
\cs_new:Nn \__trans_define_rotate_iii:n
  { \__trans_define_rotate_iii:cc {#1:nnnn} {#1:n} }
```

(*End of definition for* `\__trans_define_rotate_iii:NN` *and* `\__trans_define_rotate_iii:n`.)

`\transform_rotate_x:nnnn`
`\transform_rotate_y:nnnn`
`\transform_rotate_z:nnnn`
`\transform_rotated_x:nnnn`
`\transform_rotated_y:nnnn`
`\transform_rotated_z:nnnn`

绕指定点旋转。`#1#2#3` 为中心点，`#4` 旋转角度。

```
\__trans_define_rotate_iii:n { transform_rotate_x }
\__trans_define_rotate_iii:n { transform_rotate_y }
\__trans_define_rotate_iii:n { transform_rotate_z }
\__trans_define_rotate_iii:n { transform_rotated_x }
\__trans_define_rotate_iii:n { transform_rotated_y }
\__trans_define_rotate_iii:n { transform_rotated_z }
```

(*End of definition for* `\transform_rotate_x:nnnn` *and others. These functions are documented on page* 24.)

`\transform_scale:nnnnnn`
`\transform_shear:nnnnnnnnn`

绕指定点缩放或错切。`#1#2#3` 为中心点，`#4#5#6(#7#8#9)` 变换参数。

```
\cs_new_protected:Nn \transform_scale:nnnnnn
  {
    \transform_shift:nnn {-(#1)} {-(#2)} {-(#3)}
    \transform_scale:nnn {#4}    {#5}    {#6}
    \transform_shift:nnn {#1}    {#2}    {#3}
  }
\cs_new_protected:Nn \transform_shear:nnnnnnnnn
  {
    \transform_shift:nnn {-(#1)} {-(#2)} {-(#3)}
    \transform_shear:nnnnnn {#4} {#5}    {#6}    {#7} {#8} {#9}
    \transform_shift:nnn    {#1} {#2}    {#3}
  }
```

(*End of definition for* `\transform_scale:nnnnnn` *and* `\transform_shear:nnnnnnnnn`. *These functions are documented on page* 24.)

7.2.5 LaTeX2e 变换命令

`\__trans_args_split:Nnn`
`\__trans_args_split:Nnnn`
`\__trans_args_split:Nnnnnnn`
`\__trans_args_split_negate:Nnn`
`\__trans_args_split_negate:Nnnn`

拆分参数命令。

```
\cs_new:Npn \__trans_args_split:Nnn  #1(#2,#3)
```

```
  { #1 {#2} {#3} }
\cs_new:Npn \__trans_args_split:Nnnn #1(#2,#3,#4)
  { #1 {#2} {#3} {#4} }
\cs_new:Npn \__trans_args_split:Nnnnnnn #1(#2,#3,#4,#5,#6,#7)
  { #1 {#2} {#3} {#4} {#5} {#6} {#7} }
\cs_new:Npn \__trans_args_split_negate:Nnn  #1(#2,#3)
  { #1 { #2 * (-1) } { #3* (-1) } }
\cs_new:Npn \__trans_args_split_negate:Nnnn #1(#2,#3,#4)
  { #1 { #2 * (-1) } { #3* (-1) } { #4* (-1) } }
```

(*End of definition for* `\__trans_args_split:Nnn` *and others.*)

`\__trans_with_center:nn` 绕指定点变换。#1 中心点，#2 变换操作。

```
\cs_new_protected:Nn \__trans_with_center:nn
  {
    \int_case:nnF {\clist_count:n {#1} }
      {
        { 0 } {#2}
        { 2 }
        {
          \__trans_args_split_negate:Nnn  \transform_shift:nn  (#1)
          #2
          \__trans_args_split:Nnn         \transform_shift:nn  (#1)
        }
        { 3 }
        {
          \__trans_args_split_negate:Nnnn \transform_shift:nnn (#1)
          #2
          \__trans_args_split:Nnnn        \transform_shift:nnn (#1)
        }
        { 1 }
        { \msg_error:nnn { matrix } { dimension-mismatch } {#1} }
      }
      { \msg_error:nnn { matrix } { too-many-elements } {#1} }
  }
```

(*End of definition for* `\__trans_with_center:nn`.)

`\__trans_choose:NNn` 根据参数数量选择操作。

```
\cs_new_protected:Nn \__trans_choose:NNn
  {
    \int_case:nnF { \clist_count:n {#3} }
      {
        { 1 }
```

```
        {
          \int_compare:nNnTF { \g__trans_dim_int } = { 2 }
            { #1 {#3} {#3} }
            { #2 {#3} {#3} {#3} }
        }
        { 2 }
        { \__trans_args_split:Nnn  #1(#3) }
        { 3 }
        { \__trans_args_split:Nnnn #2(#3) }
      }
      { \msg_error:nnn { matrix } { too-many-elements } {#3} }
  }
```

(*End of definition for* `\__trans_choose:NNn`.)

`\TransformShift` 平移。

```
\NewDocumentCommand \TransformShift { m }
  {
    \int_case:nnF {\clist_count:n {#1} }
      {
        { 2 }
        { \__trans_args_split:Nnn  \transform_shift:nn  (#1) }
        { 3 }
        { \__trans_args_split:Nnnn \transform_shift:nnn (#1) }
      }
      {
        \int_compare:nNnTF {\clist_count:n {#1} } > {3}
          { \msg_error:nnn { matrix } { too-many-elements }  {#1} }
          { \msg_error:nnn { matrix } { dimension-mismatch } {#1} }
      }
  }
```

(*End of definition for* `\TransformShift`. *This function is documented on page* 27.)

`\TransformScale` 绕指定点缩放。#1 指定点为缩放中心，#2 缩放系数。

```
\NewDocumentCommand \TransformScale { O{} m }
  {
    \__trans_with_center:nn {#1}
      {
        \__trans_choose:NNn
        \transform_scale:nn \transform_scale:nnn {#2}
      }
  }
```

(*End of definition for* `\TransformScale`. *This function is documented on page* 27.)

`\TransformShear` 绕指定点错切。#1 指定点为错切中心，#2 错切系数。

```
\NewDocumentCommand \TransformShear { O{} m }
  {
    \__trans_with_center:nn {#1}
      {
        \int_case:nnF {\clist_count:n {#2} }
          {
            { 2 }
            { \__trans_args_split:Nnn     \transform_shear:nn     (#2) }
            { 6 }
            { \__trans_args_split:Nnnnnnn \transform_shear:nnnnnn (#2) }
          }
          {
            \int_compare:nNnTF {\clist_count:n {#2} } > {3}
              { \msg_error:nnn { matrix } { too-many-elements }  {#2} }
              { \msg_error:nnn { matrix } { dimension-mismatch } {#2} }
          }
      }
  }
```

(*End of definition for* `\TransformShear`. *This function is documented on page* 27.)

`\TransformRotate` `\TransformRotated` 绕指定点旋转。#1 指定点为缩放中心，#2 放置角度。

```
\NewDocumentCommand \TransformRotate  { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotate:n  {#2} }
  }
\NewDocumentCommand \TransformRotated { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotated:n {#2} }
  }
```

(*End of definition for* `\TransformRotate` *and* `\TransformRotated`. *These functions are documented on page* 28.)

`\TransformRotateX` `\TransformRotateY` `\TransformRotateZ` `\TransformRotatedX` `\TransformRotatedY` `\TransformRotatedZ` 绕指定点及指定轴向旋转。#1 指定点为缩放中心，#2 放置角度。

```
\NewDocumentCommand \TransformRotateX  { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotate_x:n  {#2} }
  }
\NewDocumentCommand \TransformRotatedX { O{} m }
  {
```

```
    \__trans_with_center:nn {#1} { \transform_rotated_x:n {#2} }
  }
\NewDocumentCommand \TransformRotateY  { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotate_y:n  {#2} }
  }
\NewDocumentCommand \TransformRotatedY { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotated_y:n {#2} }
  }
\NewDocumentCommand \TransformRotateZ  { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotate_z:n  {#2} }
  }
\NewDocumentCommand \TransformRotatedZ { O{} m }
  {
    \__trans_with_center:nn {#1} { \transform_rotated_z:n {#2} }
  }
```

(*End of definition for* `\TransformRotateX` *and others. These functions are documented on page* 28.)

7.3 执行变换

7.3.1 定义及刷新

`\__trans_apply_x:nn` 二维变换函数。

`\__trans_apply_y:nn`

```
\cs_new:Nn \__trans_apply_x:nn {#1}
\cs_new:Nn \__trans_apply_y:nn {#2}
```

(*End of definition for* `\__trans_apply_x:nn` *and* `\__trans_apply_y:nn`.)

`\__trans_apply_x:nnn` 三维变换函数。

`\__trans_apply_y:nnn`

`\__trans_apply_z:nnn`

```
\cs_new:Nn \__trans_apply_x:nnn {#1}
\cs_new:Nn \__trans_apply_y:nnn {#2}
\cs_new:Nn \__trans_apply_z:nnn {#3}
```

(*End of definition for* `\__trans_apply_x:nnn`, `\__trans_apply_y:nnn`, *and* `\__trans_apply_z:nnn`.)

`\transform_refresh:` 刷新变换函数。

`\transform_refresh:N`

`\TransformRefresh`

```
\cs_new_protected:Nn \transform_refresh:N
  {
    \matrix_if_same_dim:NnnTF #1 {3} {3}
      { % 2D
        \matrix_extract_transform:NNN  #1
```

```
          \__trans_apply_x:nn \__trans_apply_y:nn
      }
      { % 3D
        \matrix_extract_transform:NNNN #1
          \__trans_apply_x:nnn \__trans_apply_y:nnn \__trans_apply_z:nnn
      }
  }
\cs_new_protected:Nn \transform_refresh:
  { \transform_refresh:N \g__trans_matrix }
\NewDocumentCommand \TransformRefresh { O{\g__trans_matrix} }
  { \transform_refresh:N #1 }
```

(*End of definition for* `\transform_refresh:` , `\transform_refresh:N` , *and* `\TransformRefresh`. *These functions are documented on page* 25.)

7.3.2 应用变换

`\transform_apply:nnn` 应用二维变换。

```
\cs_new:Nn \transform_apply:nnn
  {
    \__trans_apply_x:nn {#1} {#2} #3,
    \__trans_apply_y:nn {#1} {#2} #3
  }
```

(*End of definition for* `\transform_apply:nnn`. *This function is documented on page* 25.)

`\transform_apply_inline:nnn` 应用二维变换。

```
\cs_new:Nn \transform_apply_inline:nnn
  {
    \cs_set:Nn \__trans_apply_inline:nn {#3}
    \__trans_apply_inline:nn
      { \__trans_apply_x:nn {#1} {#2} }
      { \__trans_apply_y:nn {#1} {#2} }
  }
```

(*End of definition for* `\transform_apply_inline:nnn`. *This function is documented on page* 26.)

`\transform_apply:nnnn` 应用三维变换。

```
\cs_new:Nn \transform_apply:nnnn
  {
    \__trans_apply_x:nnn {#1} {#2} {#3} #4,
    \__trans_apply_y:nnn {#1} {#2} {#3} #4,
    \__trans_apply_z:nnn {#1} {#2} {#3} #4
  }
```

(*End of definition for* `\transform_apply:nnnn`. *This function is documented on page 25.*)

`\transform_apply_inline:nnnn` 应用三维变换。

```
\cs_new:Nn \transform_apply_inline:nnnn
  {
    \cs_set:Nn \__trans_apply_inline:nnn {#4}
    \__trans_apply_inline:nnn
      { \__trans_apply_x:nnn {#1} {#2} {#3} }
      { \__trans_apply_y:nnn {#1} {#2} {#3} }
      { \__trans_apply_z:nnn {#1} {#2} {#3} }
  }
```

(*End of definition for* `\transform_apply_inline:nnnn`. *This function is documented on page 26.*)

`\transform_apply:nn` 应用变换。

`\TransformApply`

```
\cs_new:Nn \transform_apply:nn
  {
    \int_case:nnF { \clist_count:n {#1} }
      {
        { 2 }
        { \__trans_args_split:Nnn \transform_apply:nnn (#1) {#2} }
        { 3 }
        { \__trans_args_split:Nnnn \transform_apply:nnnn (#1) {#2} }
      }
      {
        \int_compare:nNnTF {\clist_count:n {#1} } > {3}
          { \msg_error:nnn { matrix } { too-many-elements }  {#1} }
          { \msg_error:nnn { matrix } { dimension-mismatch } {#1} }
      }
  }
\NewExpandableDocumentCommand \TransformApply { O{} m }
  { \transform_apply:nn {#2} {#1} }
```

(*End of definition for* `\transform_apply:nn` *and* `\TransformApply`. *These functions are documented on page 26.*)

⟨/package⟩

Change History

v0.2

General: 增加几何变换功能 62

增加无维度的创建函数 45

将空间基址改为空间偏移量 35

更新设计逻辑 29
添加矩阵条件句 47
补充中文用户文档及几何变换接口说明 1