RenderScript Matrix Functions

Overview

These functions let you manipulate square matrices of rank 2x2, 3x3, and 4x4. They are particularly useful for graphical transformations and are compatible with OpenGL.

We use a zero-based index for rows and columns. E.g. the last element of a rs_matrix4x4 is found at (3, 3).

RenderScript uses column-major matrices and column-based vectors. Transforming a vector is done by postmultiplying the vector, e.g. `(matrix * vector)`, as provided by rsMatrixMultiply().

To create a transformation matrix that performs two transformations at once, multiply the two source matrices, with the first transformation as the right argument. E.g. to create a transformation matrix that applies the transformation s1 followed by s2, call `rsMatrixLoadMultiply(&combined, &s2, &s1)`. This derives from `s2 * (s1 * v)`, which is `(s2 * s1) * v`.

We have two style of functions to create transformation matrices: rsMatrixLoad*Transformation* and rsMatrix*Transformation*. The former style simply stores the transformation matrix in the first argument. The latter modifies a pre-existing transformation matrix so that the new transformation happens first. E.g. if you call rsMatrixTranslate() on a matrix that already does a scaling, the resulting matrix when applied to a vector will first do the translation then the scaling.

Summary

Functions	
rsExtractFrustumPlanes	Compute frustum planes
rsIsSphereInFrustum	Checks if a sphere is within the frustum planes
rsMatrixGet	Get one element
rsMatrixInverse	Inverts a matrix in place
rsMatrixInverseTranspose	Inverts and transpose a matrix in place
rsMatrixLoad	Load or copy a matrix
rsMatrixLoadFrustum	Load a frustum projection matrix
rsMatrixLoadIdentity	Load identity matrix
rsMatrixLoadMultiply	Multiply two matrices
rsMatrixLoadOrtho	Load an orthographic projection matrix
rsMatrixLoadPerspective	Load a perspective projection matrix
rsMatrixLoadRotate	Load a rotation matrix
rsMatrixLoadScale	Load a scaling matrix
rsMatrixLoadTranslate	Load a translation matrix
rsMatrixMultiply	Multiply a matrix by a vector or another matrix
rsMatrixRotate	Apply a rotation to a transformation matrix
rsMatrixScale	Apply a scaling to a transformation matrix
rsMatrixSet	Set one element
rsMatrixTranslate	Apply a translation to a transformation matrix

rsMatrixTranspose	Transpose a matrix place

Functions

rsExtractFrustumPlanes : Compute frustum planes

void rsExtractFrustumPlanes(const rs_matrix4x4* viewProj, float4* left, float4* righ, float4* top, float4* bottom, float4* near, float4* far);

Added in API level 24

void rsExtractFrustumPlanes(const rs_matrix4x4* viewProj, float4* left, float4* right, float4* top, float4* bottom, float4* near, float4* far);

Removed from API level 24 and higher

Parameters

viewProj	Matrix to extract planes from.
left	Left plane.
right	Right plane.
top	Top plane.
bottom	Bottom plane.
near	Near plane.
far	Far plane.
righ	

Computes 6 frustum planes from the view projection matrix

rsIsSphereInFrustum : Checks if a sphere is within the frustum planes

bool rsIsSphereInFrustum(float4* sphere, float4* left, float4* right, float4* top, float4* bottom, float4* near, float4* far);

Parameters

sphere	float4 representing the sphere.
left	Left plane.
right	Right plane.
top	Top plane.
bottom	Bottom plane.
near	Near plane.
far	Far plane.

Returns true if the sphere is within the 6 frustum planes.

rsMatrixGet : Get one element

float rsMatrixGet(const rs_matrix2x2* m, uint32_t col, uint32_t row);

float rsMatrixGet(const rs_matrix3x3* m, uint32_t col, uint32_t row);

float rsMatrixGet(const rs_matrix4x4* m, uint32_t col, uint32_t row);

Parameters

m	Matrix to extract the element from.
col	Zero-based column of the element to be extracted.
row	Zero-based row of the element to extracted.

Returns one element of a matrix.

Warning: The order of the column and row parameters may be unexpected.

rsMatrixInverse : Inverts a matrix in place

bool rsMatrixInverse(rs_matrix4x4* m);

bool rsMatrixInverse(rs_matrix4x4* m);

Parameters

m Matrix to invert.

Returns true if the matrix was successfully inverted.

rsMatrixInverseTranspose : Inverts and transpose a matrix in place

bool rsMatrixInverseTranspose(rs_matrix4x4* m);

Parameters

m Matrix to modify.

The matrix is first inverted then transposed. Returns true if the matrix was successfully inverted.

rsMatrixLoad : Load or copy a matrix

void rsMatrixLoad(rs_matrix2x2* destination, const float* array);

void rsMatrixLoad(rs_matrix2x2* destination, const rs_matrix2x2* source);

void rsMatrixLoad(rs_matrix3x3* destination, const float* array);

void rsMatrixLoad(rs_matrix3x3* destination, const rs_matrix3x3* source);

void rsMatrixLoad(rs_matrix4x4* destination, const float* array);

void rsMatrixLoad(rs_matrix4x4* destination, const rs_matrix2x2* source);

void rsMatrixLoad(rs_matrix4x4* destination, const rs_matrix3x3* source);

void rsMatrixLoad(rs_matrix4x4* destination, const rs_matrix4x4* source);

Parameters

destination Matrix to set.

array Array of values to set the matrix to. These arrays should be 4, 9, or 16 floats long, depending on the matrix size.

source Source matrix.

Set the elements of a matrix from an array of floats or from another matrix.

If loading from an array, the floats should be in row-major order, i.e. the element a `row 0, column 0` should be first, followed by the element at `row 0, column 1`, etc.

If loading from a matrix and the source is smaller than the destination, the rest of the destination is filled with elements of the identity matrix. E.g. loading a rs_matrix2x2 into a rs_matrix4x4 will give:

m00	m01	0.0	0.0
m10	m11	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

rsMatrixLoadFrustum : Load a frustum projection matrix

void rsMatrixLoadFrustum(rs_matrix4x4* m, float left, float right, float bottom, float top, float near, float far);

Parameters

m Matrix to set.

left

right

bottom

top

near

far

far

Constructs a frustum projection matrix, transforming the box identified by the six clipping planes `left, right, bottom, top, near, far`.

To apply this projection to a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixLoadIdentity : Load identity matrix

void rsMatrixLoadIdentity(rs_matrix2x2* m);

void rsMatrixLoadIdentity(rs_matrix3x3* m);

void rsMatrixLoadIdentity(rs_matrix4x4* m);

Parameters

m Matrix to set.

Set the elements of a matrix to the identity matrix.

rsMatrixLoadMultiply : Multiply two matrices

void rsMatrixLoadMultiply(rs_matrix2x2* m, const rs_matrix2x2* lhs, const rs_matrix2x2* rhs);

void rsMatrixLoadMultiply(rs_matrix3x3* m, const rs_matrix3x3* lhs, const rs_matrix3x3* rhs);

void rsMatrixLoadMultiply(rs_matrix4x4* m, const rs_matrix4x4* lhs, const rs_matrix4x4* rhs);

Parameters

m Matrix to set.

lhs Left matrix of the product.

rhs Right matrix of the product.

Sets m to the matrix product of `lhs * rhs`.

To combine two 4x4 transformaton matrices, multiply the second transformation matrix by the first transformation matrix. E.g. to create a transformation matrix that applies the transformation s1 followed by s2, call `rsMatrixLoadMultiply(&combined, &s2, &s1)`.

Warning: Prior to version 21, storing the result back into right matrix is not supported and will result in undefined behavior. Use rsMatrixMulitply instead. E.g. instead of doing rsMatrixLoadMultiply (&m2r, &m2r, &m2l), use rsMatrixMultiply (&m2r, &m2l). rsMatrixLoadMultiply (&m2l, &m2r, &m2l) works as expected.

rsMatrixLoadOrtho : Load an orthographic projection matrix

void rsMatrixLoadOrtho(rs_matrix4x4* m, float left, float right, float bottom, float top, float near, float far);

Parameters

m Matrix to set.

left

right

bottom

top

near

far

Constructs an orthographic projection matrix, transforming the box identified by the six clipping planes `left, right, bottom, top, near, far` into a unit cube with a corner at `(-1, -1, -1)` and the opposite at `(1, 1, 1)`.

To apply this projection to a vector, multiply the vector by the created matrix using rsMatrixMultiply().

See https://en.wikipedia.org/wiki/Orthographic_projection .

rsMatrixLoadPerspective : Load a perspective projection matrix

void rsMatrixLoadPerspective(rs_matrix4x4* m, float fovy, float aspect, float near, float far);

Parameters

m	Matrix to set.
fovy	Field of view, in degrees along the Y axis.
aspect	Ratio of x / y.
near	Near clipping plane.
far	Far clipping plane.

Constructs a perspective projection matrix, assuming a symmetrical field of view.

To apply this projection to a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixLoadRotate : Load a rotation matrix

void rsMatrixLoadRotate(rs_matrix4x4* m, float rot, float x, float y, float z);

Parameters

m	Matrix to set.
rot	How much rotation to do, in degrees.
x	X component of the vector that is the axis of rotation.
y	Y component of the vector that is the axis of rotation.
z	Z component of the vector that is the axis of rotation.

This function creates a rotation matrix. The axis of rotation is the `(x, y, z)` vector.

To rotate a vector, multiply the vector by the created matrix using rsMatrixMultiply().

See http://en.wikipedia.org/wiki/Rotation_matrix .

rsMatrixLoadScale : Load a scaling matrix

void rsMatrixLoadScale(rs_matrix4x4* m, float x, float y, float z);

Parameters

m	Matrix to set.
x	Multiple to scale the x components by.
y	Multiple to scale the y components by.
z	Multiple to scale the z components by.

This function creates a scaling matrix, where each component of a vector is multiplied by a number. This number can be negative.

To scale a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixLoadTranslate : Load a translation matrix

void rsMatrixLoadTranslate(rs_matrix4x4* m, float x, float y, float z);

Parameters

m	Matrix to set.
x	Number to add to each x component.
y	Number to add to each y component.
z	Number to add to each z component.

This function creates a translation matrix, where a number is added to each element of a vector.

To translate a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixMultiply : Multiply a matrix by a vector or another matrix

float2 rsMatrixMultiply(const rs_matrix2x2* m, float2 in);	Added in API level 14
float2 rsMatrixMultiply(rs_matrix2x2* m, float2 in);	Removed from API level 14 and higher
float3 rsMatrixMultiply(const rs_matrix3x3* m, float2 in);	Added in API level 14
float3 rsMatrixMultiply(const rs_matrix3x3* m, float3 in);	Added in API level 14
float3 rsMatrixMultiply(rs_matrix3x3* m, float2 in);	Removed from API level 14 and higher
float3 rsMatrixMultiply(rs_matrix3x3* m, float3 in);	Removed from API level 14 and higher
float4 rsMatrixMultiply(const rs_matrix4x4* m, float2 in);	Added in API level 14
float4 rsMatrixMultiply(const rs_matrix4x4* m, float3 in);	Added in API level 14
float4 rsMatrixMultiply(const rs_matrix4x4* m, float4 in);	Added in API level 14
float4 rsMatrixMultiply(rs_matrix4x4* m, float2 in);	Removed from API level 14 and higher
float4 rsMatrixMultiply(rs_matrix4x4* m, float3 in);	Removed from API level 14 and higher
float4 rsMatrixMultiply(rs_matrix4x4* m, float4 in);	Removed from API level 14 and higher

void rsMatrixMultiply(rs_matrix2x2* m, const rs_matrix2x2* rhs);

void rsMatrixMultiply(rs_matrix3x3* m, const rs_matrix3x3* rhs);

void rsMatrixMultiply(rs_matrix4x4* m, const rs_matrix4x4* rhs);

Parameters

m Left matrix of the product and the matrix to be set.

rhs Right matrix of the product.

in

For the matrix by matrix variant, sets m to the matrix product `m * rhs`.

When combining two 4x4 transformation matrices using this function, the resulting matrix will correspond to performing the rhs transformation first followed by the original m transformation.

For the matrix by vector variant, returns the post-multiplication of the vector by the matrix, ie. `m * in`.

When multiplying a float3 to a rs_matrix4x4, the vector is expanded with (1).

When multiplying a float2 to a rs_matrix4x4, the vector is expanded with (0, 1).

When multiplying a float2 to a rs_matrix3x3, the vector is expanded with (0).

Starting with API 14, this function takes a const matrix as the first argument.

rsMatrixRotate : Apply a rotation to a transformation matrix

void rsMatrixRotate(rs_matrix4x4* m, float rot, float x, float y, float z);

Parameters

m Matrix to modify.

rot How much rotation to do, in degrees.

x X component of the vector that is the axis of rotation.

y Y component of the vector that is the axis of rotation.

z Z component of the vector that is the axis of rotation.

Multiply the matrix m with a rotation matrix.

This function modifies a transformation matrix to first do a rotation. The axis of rotation is the `(x, y, z)` vector.

To apply this combined transformation to a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixScale : Apply a scaling to a transformation matrix

void rsMatrixScale(rs_matrix4x4* m, float x, float y, float z);

Parameters

m Matrix to modify.

x Multiple to scale the x components by.

y Multiple to scale the y components by.

z Multiple to scale the z components by.

Multiply the matrix m with a scaling matrix.

This function modifies a transformation matrix to first do a scaling. When scaling, each component of a vector is multiplied by a number. This number can be negative.

To apply this combined transformation to a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixSet : Set one element

void rsMatrixSet(rs_matrix2x2* m, uint32_t col, uint32_t row, float v);

void rsMatrixSet(rs_matrix3x3* m, uint32_t col, uint32_t row, float v);

void rsMatrixSet(rs_matrix4x4* m, uint32_t col, uint32_t row, float v);

Parameters

m Matrix that will be modified.

col Zero-based column of the element to be set.

row Zero-based row of the element to be set.

v Value to set.

Set an element of a matrix.

Warning: The order of the column and row parameters may be unexpected.

rsMatrixTranslate : Apply a translation to a transformation matrix

void rsMatrixTranslate(rs_matrix4x4* m, float x, float y, float z);

Parameters

m Matrix to modify.

x Number to add to each x component.

y Number to add to each y component.

z Number to add to each z component.

Multiply the matrix m with a translation matrix.

This function modifies a transformation matrix to first do a translation. When translating, a number is added to each component of a vector.

To apply this combined transformation to a vector, multiply the vector by the created matrix using rsMatrixMultiply().

rsMatrixTranspose : Transpose a matrix place

void rsMatrixTranspose(rs_matrix2x2* m);

void rsMatrixTranspose(rs_matrix3x3* m);

void rsMatrixTranspose(rs_matrix4x4* m);

Parameters

m Matrix to transpose.

Transpose the matrix m in place.