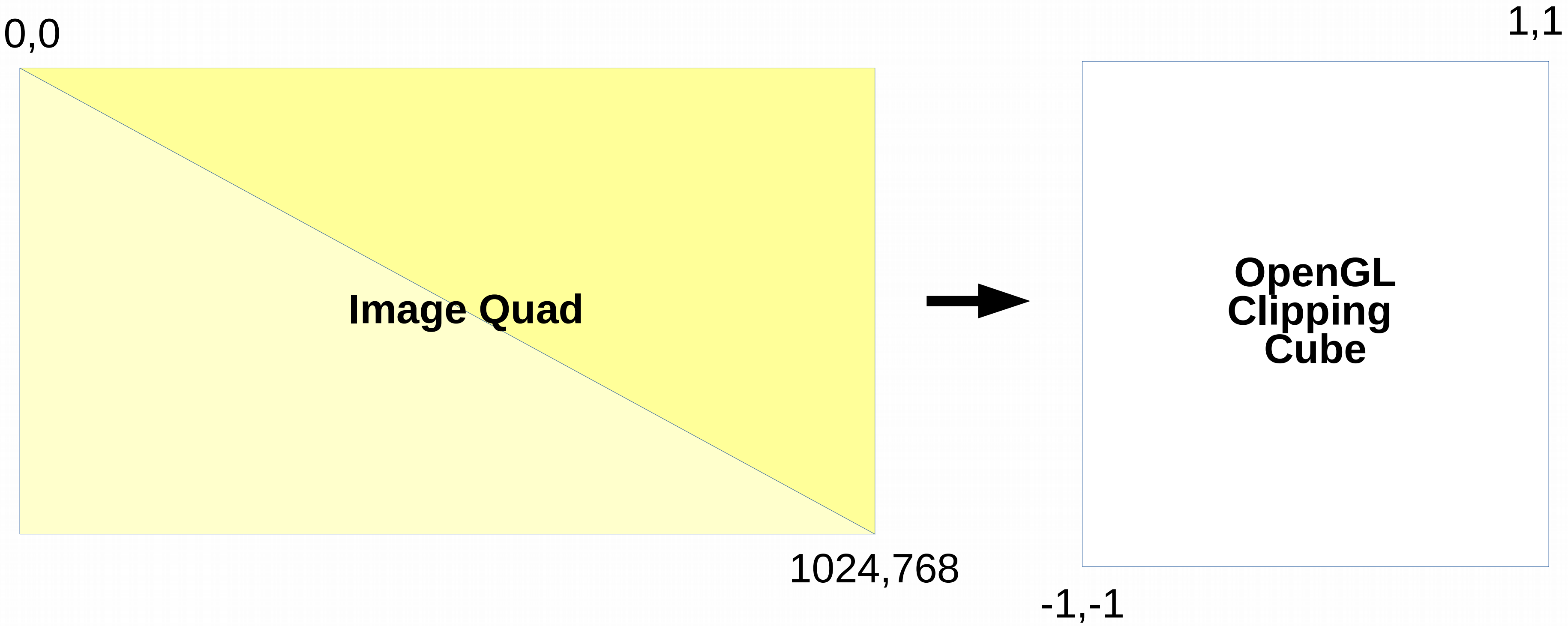
CPSC 453

BONUS: Foolproof Transforms

For when you can’t figure out translation/rotation/etc.

A Simple Example



A Matching Vertex Shader

```
in vec3 INCOMING;
out vec3 OUTGOING;
uniform mat3 MODELVIEW;

void main() {

    OUTGOING = MODELVIEW * INCOMING;
    }
```

A Matching Vertex Shader

```
in vec3 INCOMING;
out vec3 OUTGOING;
uniform mat3 MODELVIEW;       // what goes here?

void main() {

   OUTGOING = MODELVIEW * INCOMING;
   }
```

Matrix Math

$$O = M \cdot I$$

$$O \cdot I^{-1} = M \cdot I \cdot I^{-1}$$

$$O \cdot I^{-1} = M$$

Matrix Math

$$O = M \cdot I$$

$$O \cdot I^{-1} = M \cdot I \cdot I^{-1}$$

$$O \cdot I^{-1} = M$$

(assuming I is a square matrix and $det(I) \neq 0$)

A Matching Vertex Shader

```
in vec3 INCOMING;         // not a matrix :’(
out vec3 OUTGOING;
uniform mat3 MODELVIEW;

void main() {

   OUTGOING = MODELVIEW * INCOMING;
   }
```

A Matching Vertex Shader

```
in vec3 INCOMING;         // not a matrix ...
out vec3 OUTGOING;
uniform mat3 MODELVIEW;

void main() {

    vec3 temp[3];   // but matrix = series of vectors
    mat3 MATRIX = mat3( temp[0], temp[1], temp[2] );

    OUTGOING = MODELVIEW * INCOMING;
    }
```

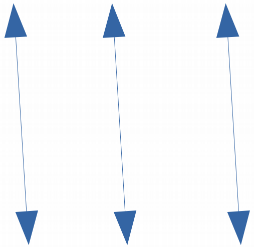
Matrix Multiplication as Mapping Function

$$O = M \cdot I$$

$$M : I \rightarrow O$$

Vectors to Matricies

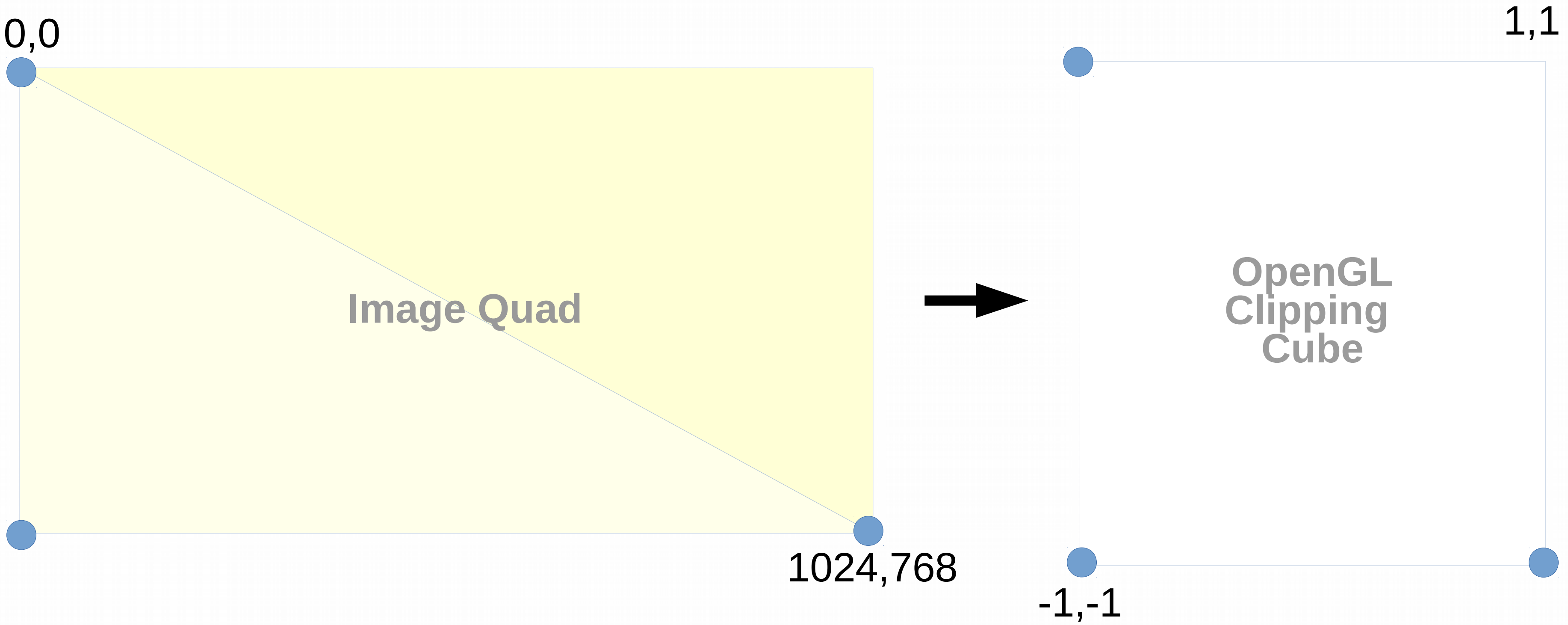
$$I = \begin{bmatrix} \\ \\ \\ \end{bmatrix}\begin{bmatrix} \\ \\ \\ \end{bmatrix}\begin{bmatrix} \\ \\ \\ \end{bmatrix}$$



(equivalent points)

$$O = \begin{bmatrix} \\ \\ \\ \end{bmatrix}\begin{bmatrix} \\ \\ \\ \end{bmatrix}\begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Pick Unique Points

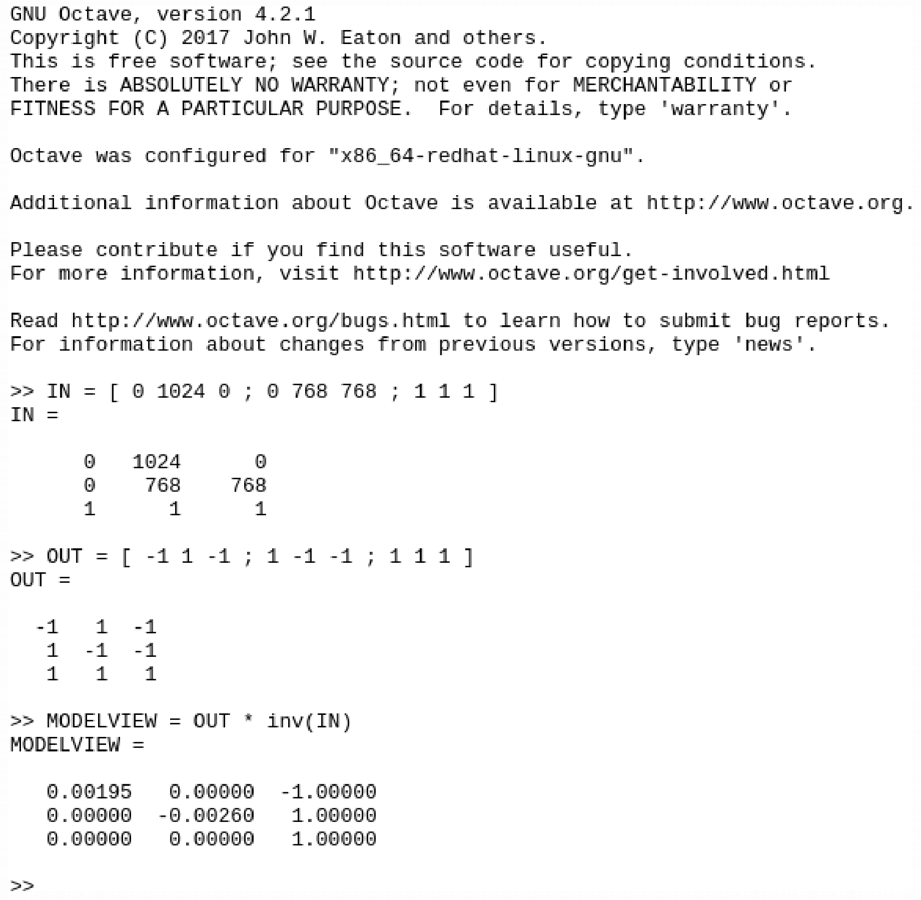


Vectors to Matricies

$$I = \begin{bmatrix} 0 & 1024 & 0 \\ 0 & 768 & 768 \\ 1 & 1 & 1 \end{bmatrix}$$

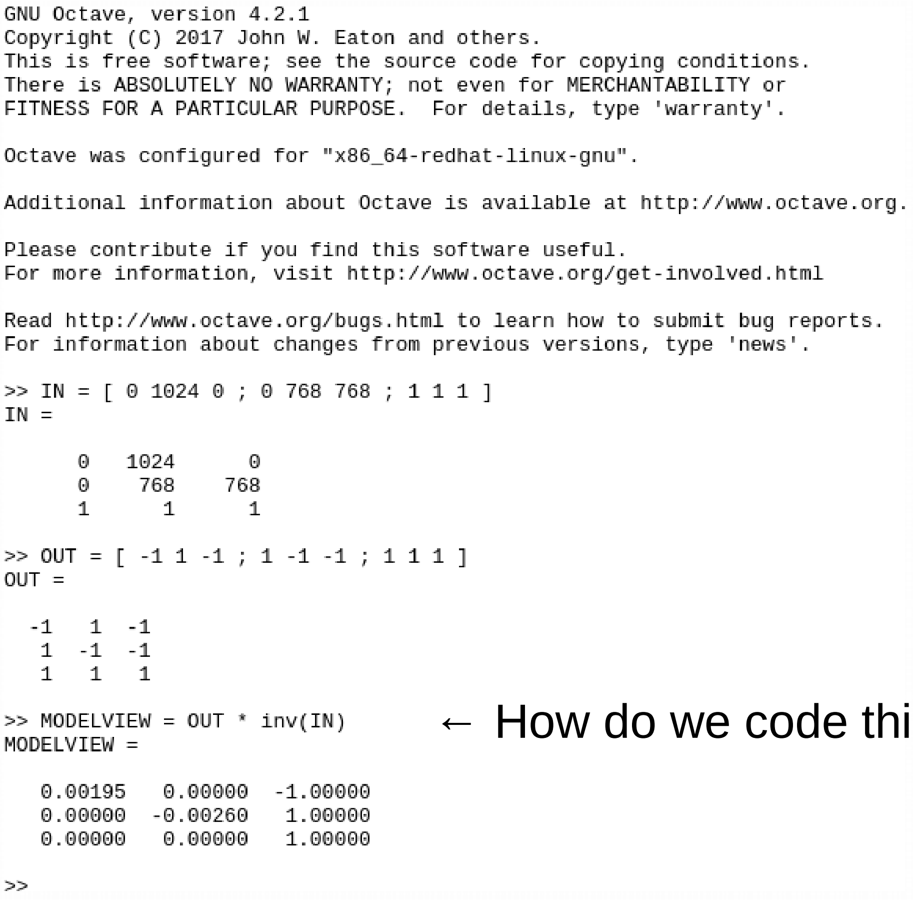
$$O = \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

Matrix Math



```
GNU Octave, version 4.2.1
Copyright (C) 2017 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-redhat-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> IN = [ 0 1024 0 ; 0 768 768 ; 1 1 1 ]
IN =

      0   1024      0
      0    768    768
      1      1      1

>> OUT = [ -1 1 -1 ; 1 -1 -1 ; 1 1 1 ]
OUT =

  -1   1  -1
   1  -1  -1
   1   1   1

>> MODELVIEW = OUT * inv(IN)
MODELVIEW =

   0.00195   0.00000  -1.00000
   0.00000  -0.00260   1.00000
   0.00000   0.00000   1.00000

>>
```

Matrix Math

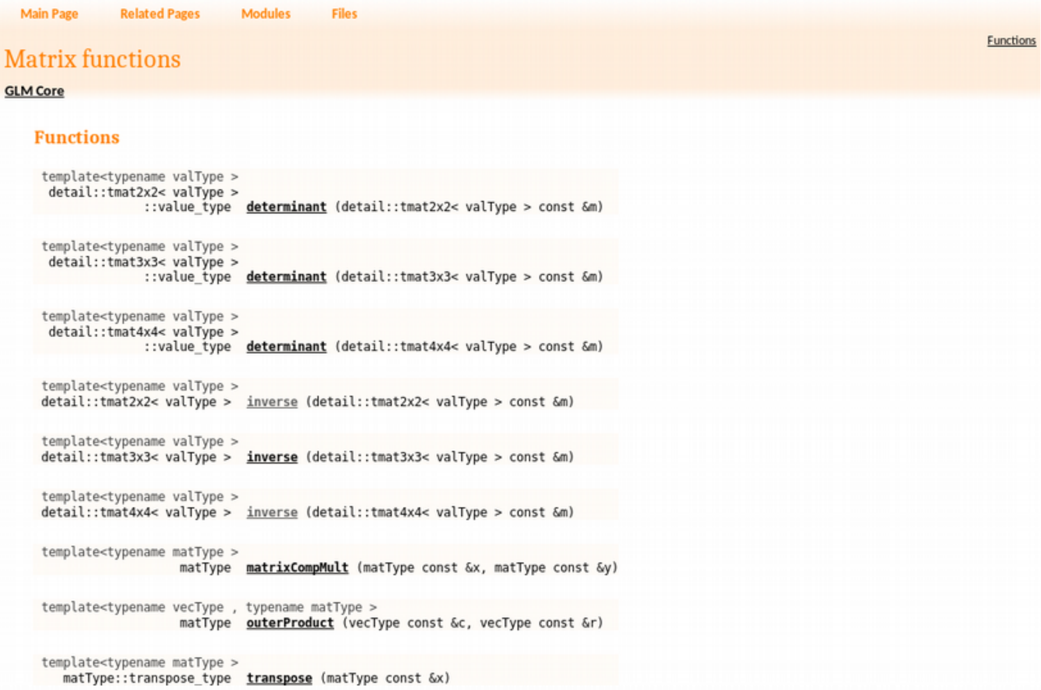


```
GNU Octave, version 4.2.1
Copyright (C) 2017 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-redhat-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> IN = [ 0 1024 0 ; 0 768 768 ; 1 1 1 ]
IN =

      0   1024      0
      0    768    768
      1      1      1

>> OUT = [ -1 1 -1 ; 1 -1 -1 ; 1 1 1 ]
OUT =

  -1   1  -1
   1  -1  -1
   1   1   1

>> MODELVIEW = OUT * inv(IN)
MODELVIEW =

   0.00195   0.00000  -1.00000
   0.00000  -0.00260   1.00000
   0.00000   0.00000   1.00000

>>
```

← How do we code this?

CPU-side

https://glm.g-truc.net/0.9.4/api/a00133.html



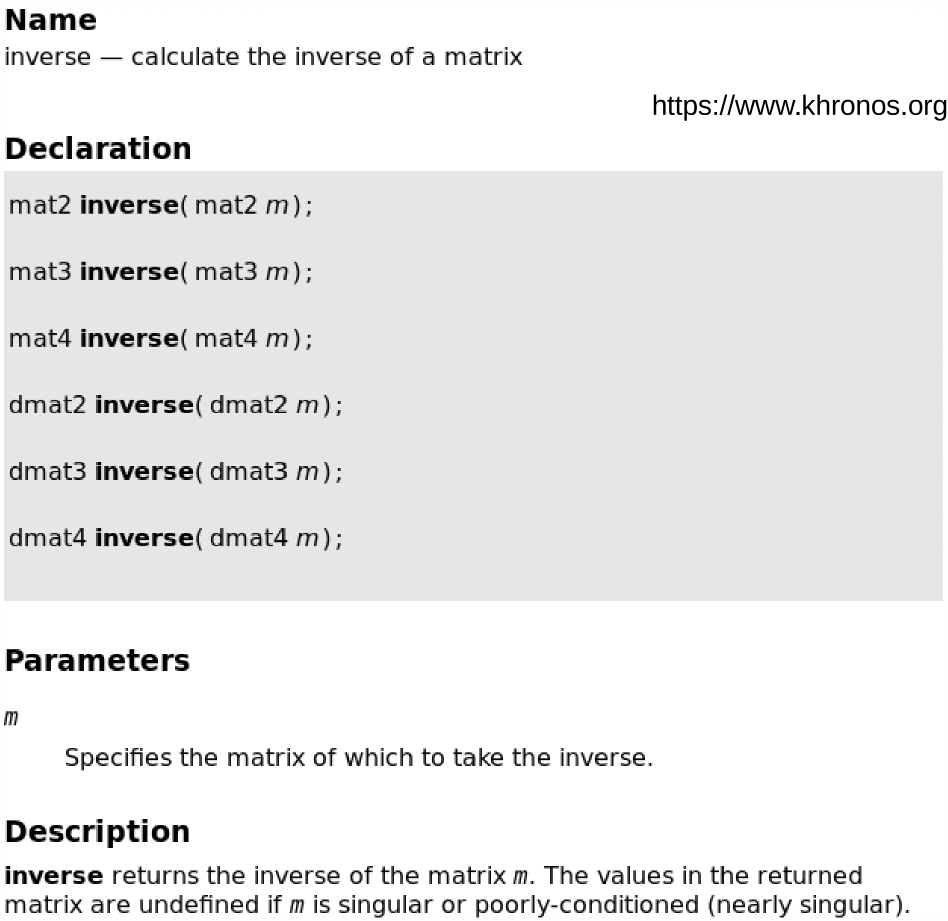
Matrix functions

GLM Core

Functions

```
template<typename valType >
 detail::tmat2x2< valType >
             ::value_type   determinant (detail::tmat2x2< valType > const &m)

template<typename valType >
 detail::tmat3x3< valType >
             ::value_type   determinant (detail::tmat3x3< valType > const &m)

template<typename valType >
 detail::tmat4x4< valType >
             ::value_type   determinant (detail::tmat4x4< valType > const &m)

template<typename valType >
detail::tmat2x2< valType >  inverse (detail::tmat2x2< valType > const &m)

template<typename valType >
detail::tmat3x3< valType >  inverse (detail::tmat3x3< valType > const &m)

template<typename valType >
detail::tmat4x4< valType >  inverse (detail::tmat4x4< valType > const &m)

template<typename matType >
                   matType  matrixCompMult (matType const &x, matType const &y)

template<typename vecType , typename matType >
                   matType  outerProduct (vecType const &c, vecType const &r)

template<typename matType >
    matType::transpose_type  transpose (matType const &x)
```

GPU-side



Name

inverse — calculate the inverse of a matrix

https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/inverse.xhtml

Declaration

```
mat2 inverse( mat2 m);

mat3 inverse( mat3 m);

mat4 inverse( mat4 m);

dmat2 inverse( dmat2 m);

dmat3 inverse( dmat3 m);

dmat4 inverse( dmat4 m);
```

Parameters

m

Specifies the matrix of which to take the inverse.

Description

inverse returns the inverse of the matrix *m*. The values in the returned matrix are undefined if *m* is singular or poorly-conditioned (nearly singular).