|  |  |
| --- | --- |
| **Column-major:** | **Row-major :** |
| Array Indices:    Subscript Indices:  Basis (Axis Direction) Vectors:  Multiplication order: **CBAx** | Array Indices:    Subscript Indices:  Basis (Axis Direction) Vectors:  Multiplication order: **xABC** |

Notes:

* Array indices correspond to the same elements in both row and column-major matrices.
* This document uses column-major stuff from now on (transpose and reverse the order of multiplication to get row-major).

**Rotation:**

|  |  |  |
| --- | --- | --- |
| x-axis: | y-axis: | z-axis |
| arbitrary axis: | | |

**Scale:**

**Matrix Multiplication:**

Take dot product of rows of the first matrix, and column of second matrix (same for both row and column-major matrices).

Column-major:

Row-major:

**Projection:**

|  |  |
| --- | --- |
| **Orthographic:**  Offsets from camera (in world space):   * l, r, t, b – left, right, top, bottom of screen. * n, f – near and far planes. | **Perspective:** |

**Transforms:**

* Model matrix – model local to world space.
* Camera matrix – camera local (view space) to world space.
* View matrix – world to view space (inverse camera matrix).
* Projection matrix – view space to clip space.

**Spaces:**

* Clip space – geometry outside of the frustum is clipped here. Then vertices are normalized (divided by w – the “perspective divide”) to get normalized device coordinates (NDC).
* NDC – coordinates between -1.0 and 1.0 (D3D uses 0.0 to 1.0 for z instead). The viewport transformation is then applied to get screen space.
* Screen Space – pixel positions (i.e. gl\_FragCoord).

**Inverse:**

// from glm

mat4 matrix\_inverse(mat4 m)

{

float c00 = m[2][2] \* m[3][3] - m[3][2] \* m[2][3];

float c02 = m[1][2] \* m[3][3] - m[3][2] \* m[1][3];

float c03 = m[1][2] \* m[2][3] - m[2][2] \* m[1][3];

float c04 = m[2][1] \* m[3][3] - m[3][1] \* m[2][3];

float c06 = m[1][1] \* m[3][3] - m[3][1] \* m[1][3];

float c07 = m[1][1] \* m[2][3] - m[2][1] \* m[1][3];

float c08 = m[2][1] \* m[3][2] - m[3][1] \* m[2][2];

float c10 = m[1][1] \* m[3][2] - m[3][1] \* m[1][2];

float c11 = m[1][1] \* m[2][2] - m[2][1] \* m[1][2];

float c12 = m[2][0] \* m[3][3] - m[3][0] \* m[2][3];

float c14 = m[1][0] \* m[3][3] - m[3][0] \* m[1][3];

float c15 = m[1][0] \* m[2][3] - m[2][0] \* m[1][3];

float c16 = m[2][0] \* m[3][2] - m[3][0] \* m[2][2];

float c18 = m[1][0] \* m[3][2] - m[3][0] \* m[1][2];

float c19 = m[1][0] \* m[2][2] - m[2][0] \* m[1][2];

float c20 = m[2][0] \* m[3][1] - m[3][0] \* m[2][1];

float c22 = m[1][0] \* m[3][1] - m[3][0] \* m[1][1];

float c23 = m[1][0] \* m[2][1] - m[2][0] \* m[1][1];

vec4 f0 = vec4(c00, c00, c02, c03);

vec4 f1 = vec4(c04, c04, c06, c07);

vec4 f2 = vec4(c08, c08, c10, c11);

vec4 f3 = vec4(c12, c12, c14, c15);

vec4 f4 = vec4(c16, c16, c18, c19);

vec4 f5 = vec4(c20, c20, c22, c23);

vec4 v0 = vec4(m[1][0], m[0][0], m[0][0], m[0][0]);

vec4 v1 = vec4(m[1][1], m[0][1], m[0][1], m[0][1]);

vec4 v2 = vec4(m[1][2], m[0][2], m[0][2], m[0][2]);

vec4 v3 = vec4(m[1][3], m[0][3], m[0][3], m[0][3]);

vec4 inv0 = vec4(v1 \* f0 - v2 \* f1 + v3 \* f2);

vec4 inv1 = vec4(v0 \* f0 - v2 \* f3 + v3 \* f4);

vec4 inv2 = vec4(v0 \* f1 - v1 \* f3 + v3 \* f5);

vec4 inv3 = vec4(v0 \* f2 - v1 \* f4 + v2 \* f5);

vec4 signA = vec4(+1, -1, +1, -1);

vec4 signB = vec4(-1, +1, -1, +1);

mat4 inverse = mat4(inv0 \* signA, inv1 \* signB, inv2 \* signA, inv3 \* signB);

vec4 row0 = vec4(inverse[0][0], inverse[1][0], inverse[2][0], inverse[3][0]);

vec4 dot0 = vec4(m[0] \* row0);

float d = (dot0.x + dot0.y) + (dot0.z + dot0.w);

return inverse \* (1.0 / d);

}