Transformation

Computer Graphics 4052

6 October

Dr Anton Gerdelan
gerdela@scss.tcd.ie

Simplest Example

- Easiest way to **scale** our triangle?
- Easiest way to **move** our triangle?
- **Demo time**

First – Maths Revision

- **Q. Define a**
  - 3d point
  - 3d vector
  - 3d unit vector
- **Q. What do we use each for in graphics?**

Modify from Main Programme?

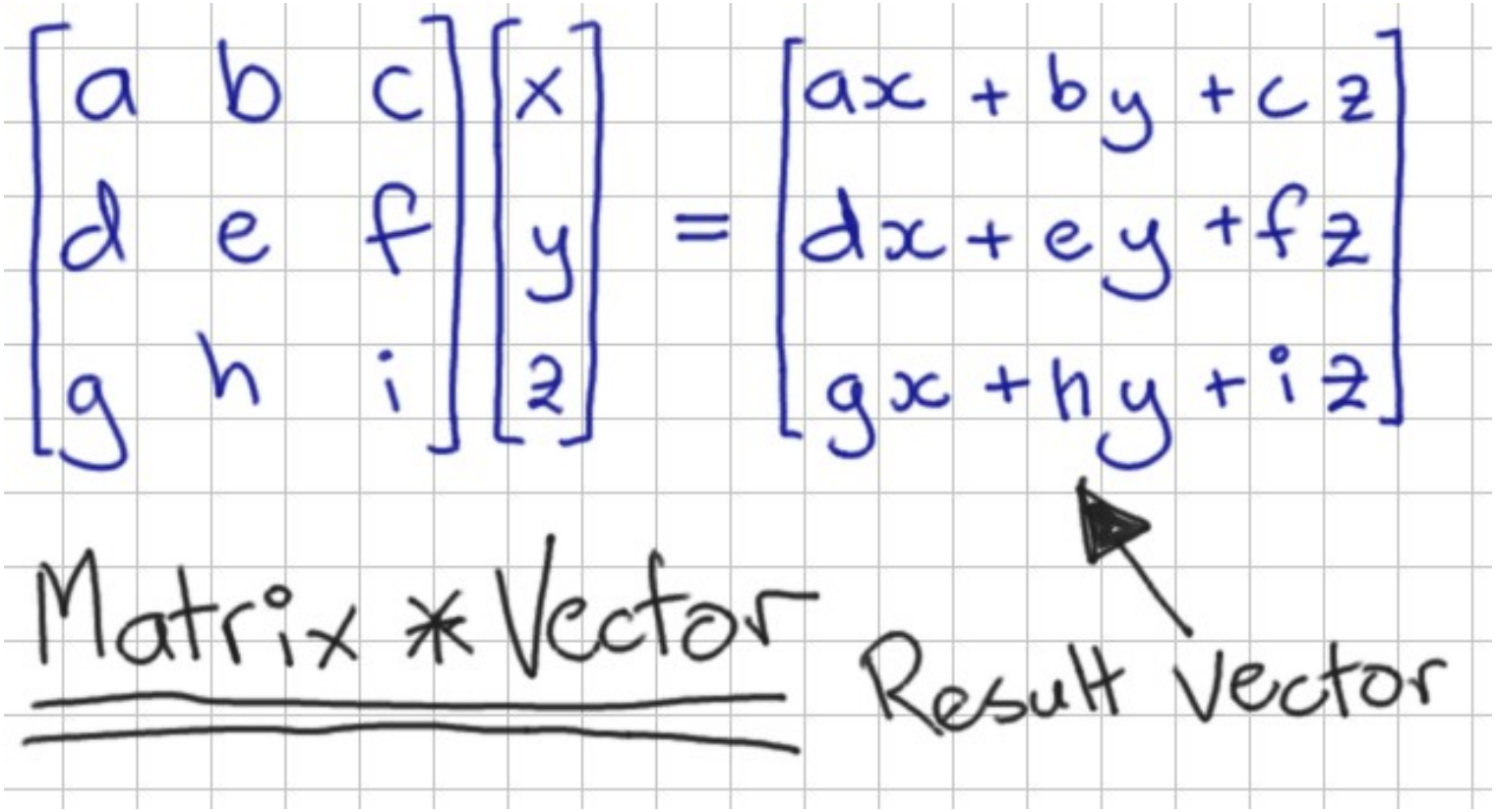
- How can we update/change the amount that we move or scale?
- Hint: use a shader keyword
- **Demo time**

Rotation

- Q. How can we rotate our triangle?
- How can we rotate a 3d point?
- Rotation is around the **origin** (the 0,0,0 point)

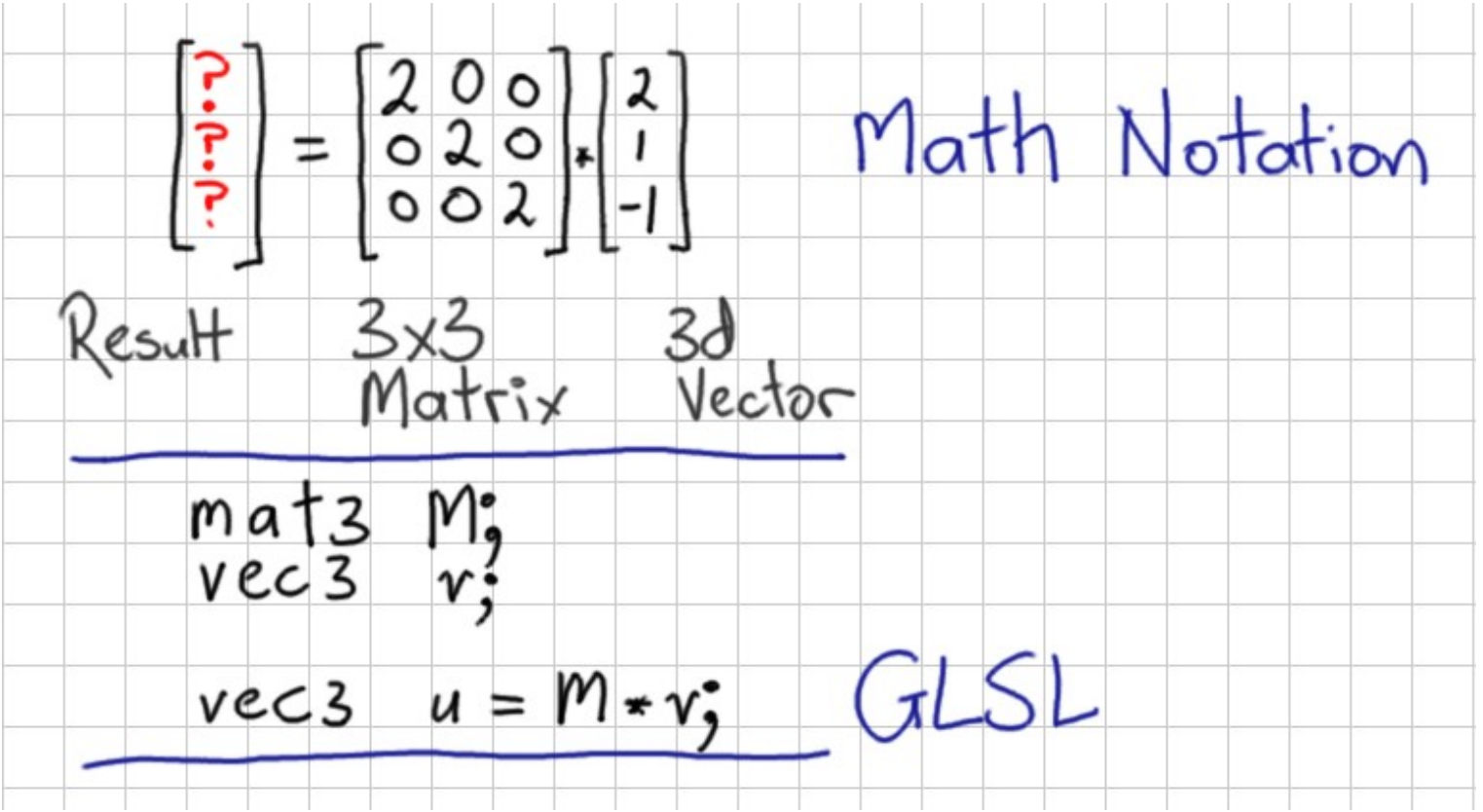
Transformation Matrices

- Translation / rotation / scale are called **affine** transforms
- Multiply 3x3 matrix with 3d vector to apply it



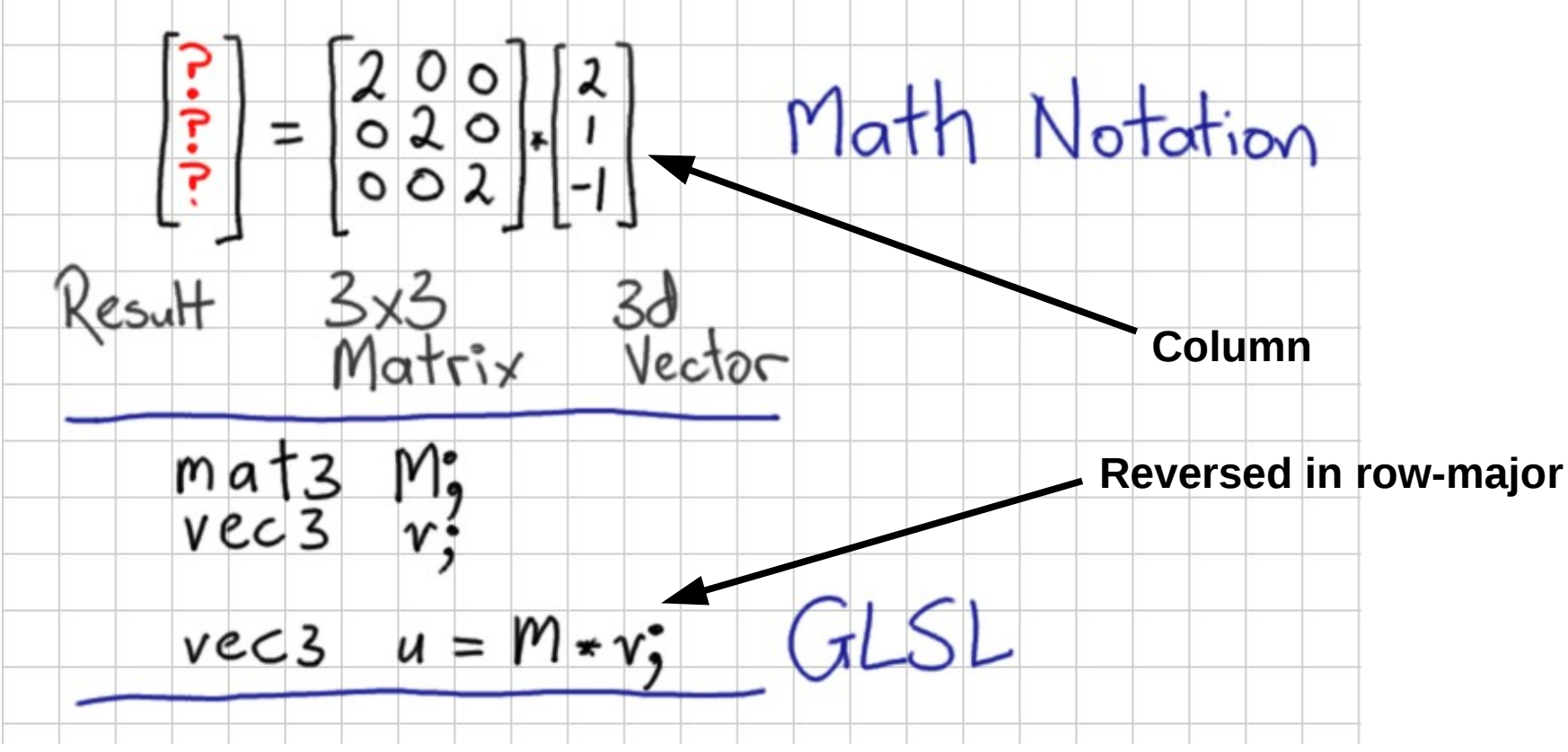
Matrix * Vector

- **Q. What is the result in the vector?**
- i.e. what does this matrix do?



Row-Major vs. Column-Major

- We will use column-major notation
- Multiplication order for column-major is right-to-left
- It is possible to use row-major instead (most DX apps do)



3X3 Rotation Around Z-Axis

$$\begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0.0 \\ \sin(\theta) & \cos(\theta) & 0.0 \\ 0.0 & 0.0 & 1.0 \end{vmatrix}$$

- Theta will be in radians in C
- Right-hand rule for rotation direction
- **Q. Which way will my triangle turn on screen?**

Rotation Demo

- Define matrix as an array of floats (how many for a 3x3?)

- Array memory is in **column order** for OpenGL

- Update shader uniform for matrix inside main loop
- `glUniform...()` family of functions
- `glUniformMatrix3fv()` takes an array of 9 floats

TRANSFORMATION PART II

4x4 Homogenous Matrices

- You can use 3d matrices, but we tend to use 4d matrices in graphics, and 4d vectors/points.
- These are not 4d hyper-geometry - it's a sneaky exploit.
- **Q. Any idea why we might like 4d matrices?**

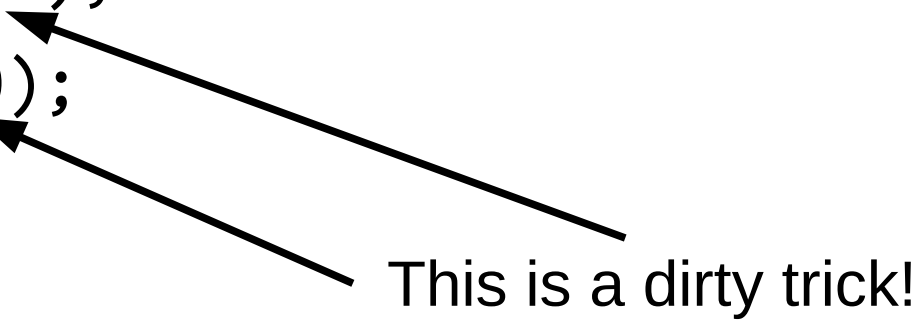
Matrix * Vector Rules

- A 3x3 matrix (`mat3`) can only multiply with a 3d vector (`vec3`)
- A 4x4 matrix (`mat4`) can only multiply with a 4d (`vec4`)
- **Q. If we have a 4x4 matrix and a 3d point, how do we make our 3d point into a 4d point?**

4d Vectors

- XYZ and W
- `vec4` in GLSL
- For POINTS set the 4th component to 1.0
- For VECTORS set the 4th component to 0.0
- **Q. Any idea why?**

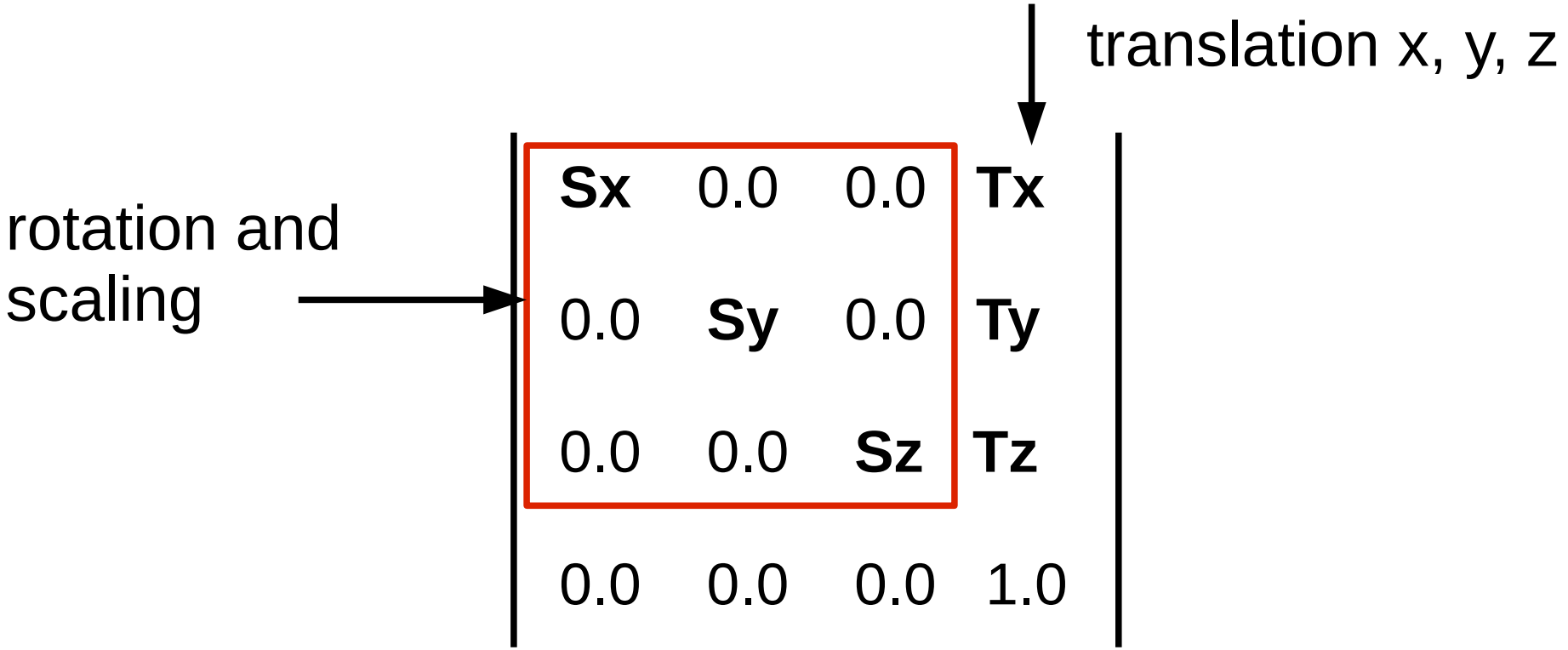
- `vec4 (1.0, 5.0, -20.0, 0.0);`
- `vec4 (0.0, -1.0, 0.0, 1.0);`



Now, What Was the Point in Going 4d?

- We can combine many matrices together by multiplication
  - `mat4 M = R * T * S;`
  - `vec4 result = M * vec4 (vp.xyz, 1.0);`
- Send fewer matrix uniforms updated over the bus
- Create a **transformation pipeline** (more on that soon)

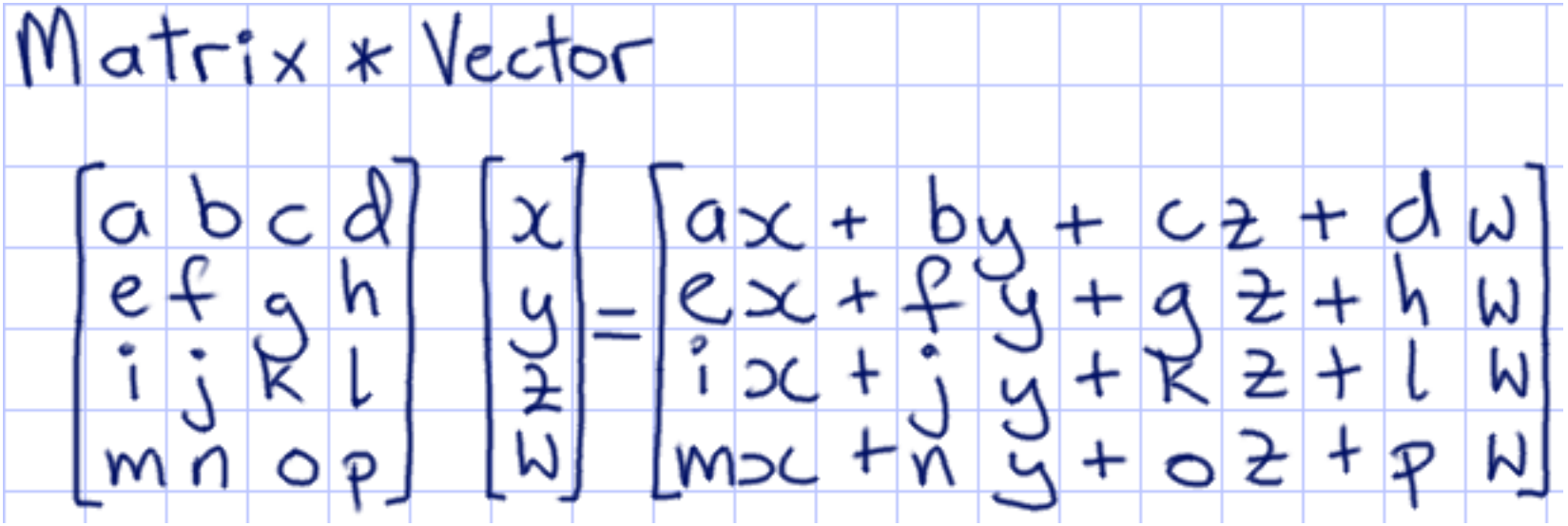
4X4 Homogenous Matrix



Putting translation in the final column lets us do our sneaky trick...

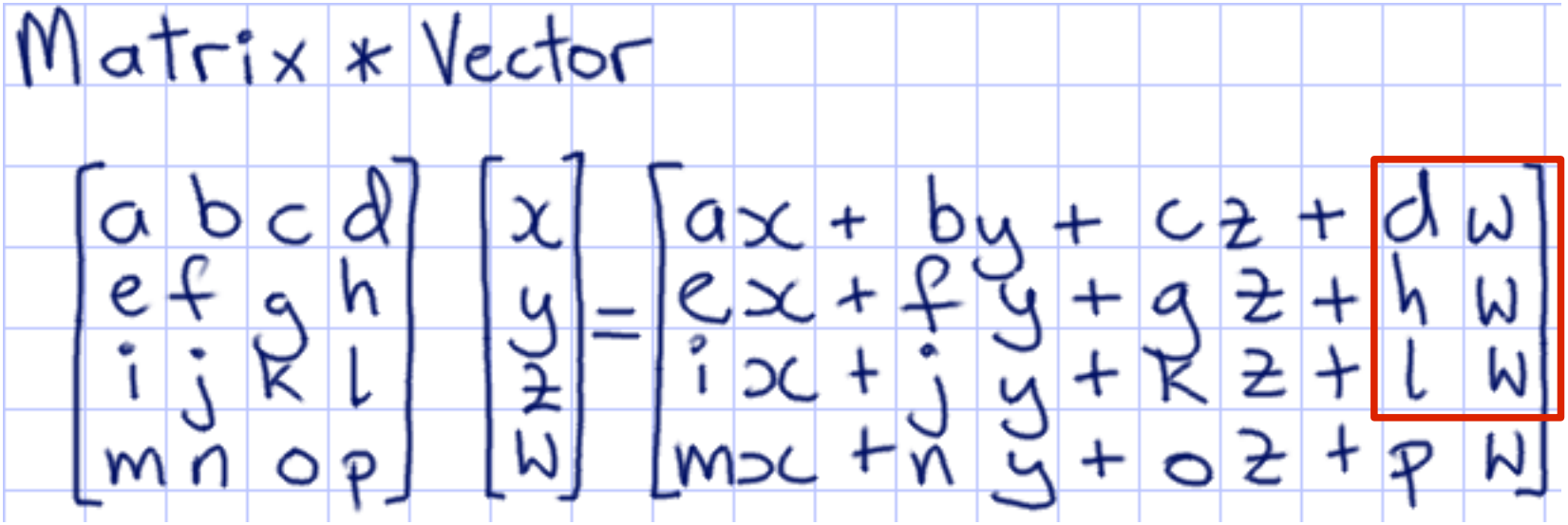
Matrix * Vector

- **Q. Can you figure it out?**
  What the 0 or 1 does at the end of a vec4?



Matrix * Vector

- **Q. Can you figure it out?**
  What the 0 or 1 does at the end of a vec4?



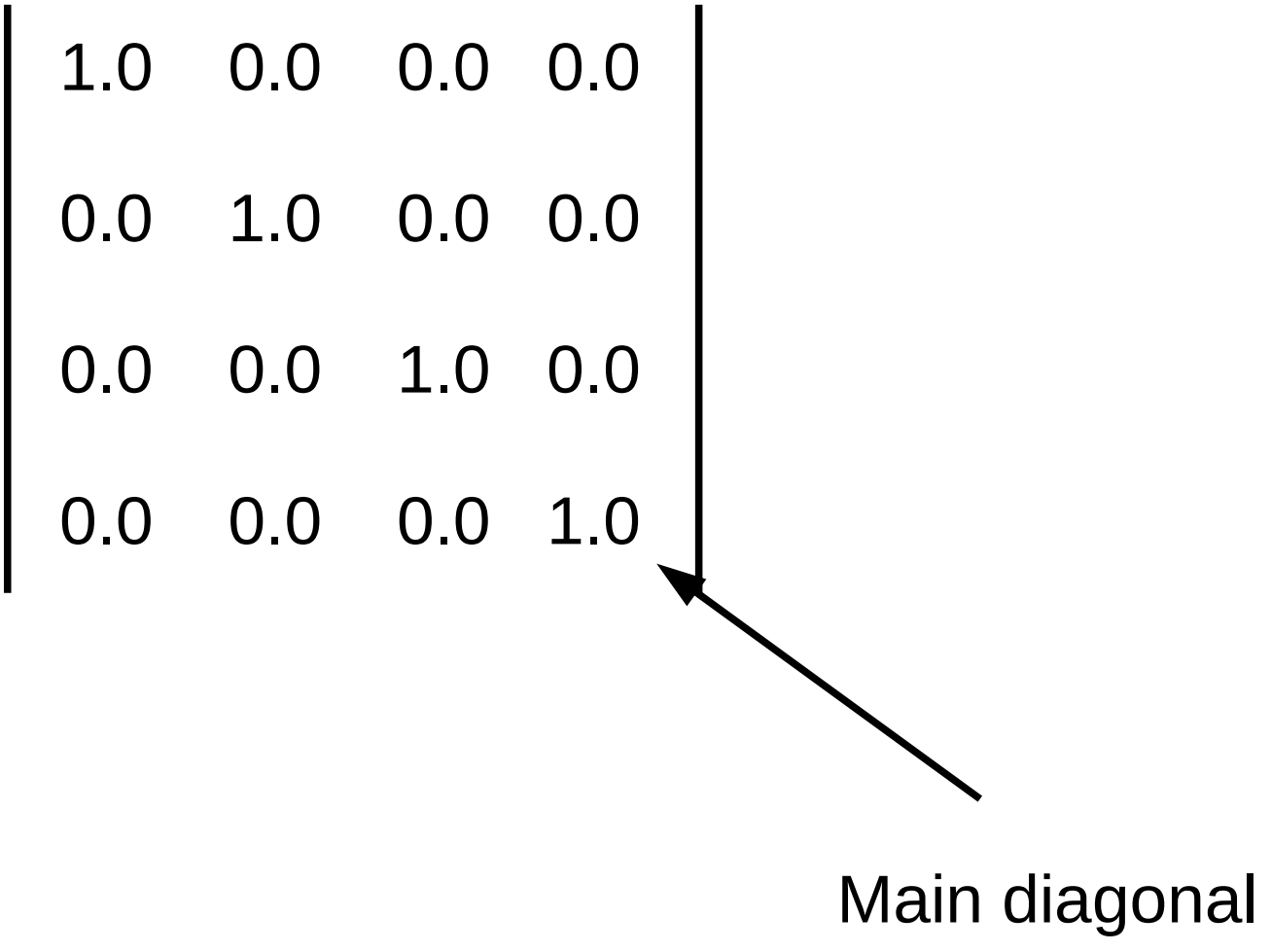
Matrix Multiplication

$$AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} (ae + bg) & (af + bh) \\ (ce + dg) & (cf + dh) \end{bmatrix}$$

A **B**

```
Each cell in result is =
   Sum of:
      A(our row, first col) * B(first row, our col) +
      ...                ...                ...
      + A(our row, last col) * B(last row, our col)
```

Identity Matrix



Transpose Matrix

- **Swaps between column-major and row-major layout**
- Flip values over the main diagonal
- **Q. Can y'all compute the transpose of this?**

$$\begin{vmatrix} 1.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 \\ -5.0 & 0.0 & 1.0 & -1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{vmatrix}$$

Inverse Matrix

- **Reverses any matrix transformation**
- Or transform relative to another object
- Quite complicated to compute
- Work out determinant, then multiply with cofactors

$$\begin{vmatrix} 1.0 & 0.0 & 0.0 & 2.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{vmatrix}$$

Q. Guess?

Most Important Homework

1. Find out how to build & use the following for 4x4 matrices **on paper**:
   - Identity
   - Scaling
   - Translation
   - Rotation around X axis, Y axis, and Z axis
   - Matrix * Matrix
2. Spot the difference between row and column-major layouts
   - Hint: column-major has the translation in part in a column
3. Know which order multiplication goes in R-to-L or L-to-R?

Guidelines

- Get a 3d maths library for C/C++ or make your own
  - Christophe Riccio's GLM http://glm.g-truc.net/
  - I made a simple one (Blackboard)
- Make a cheat-sheet (or grab mine off Blackboard)
- Know how the maths work for all of these operations
- If unusure – textbooks and online sources!
- This stuff definitely comes up in job interview tests, especially certain famous companies starting with 'H'

Next

- Extending the **transformation pipeline** to add a virtual camera
  - viewing position and angle
  - using perspective

Notes

- I deliberately skipped some things
  - Vector addition
  - Unit vectors and normalisation
  - Dot product of 2 vectors
  - Cross product of 2 vectors
- I plan to introduce this where we actually use it (lighting)
- ...and because I ramble too much