Vectors, Matrices and Uniform Variables

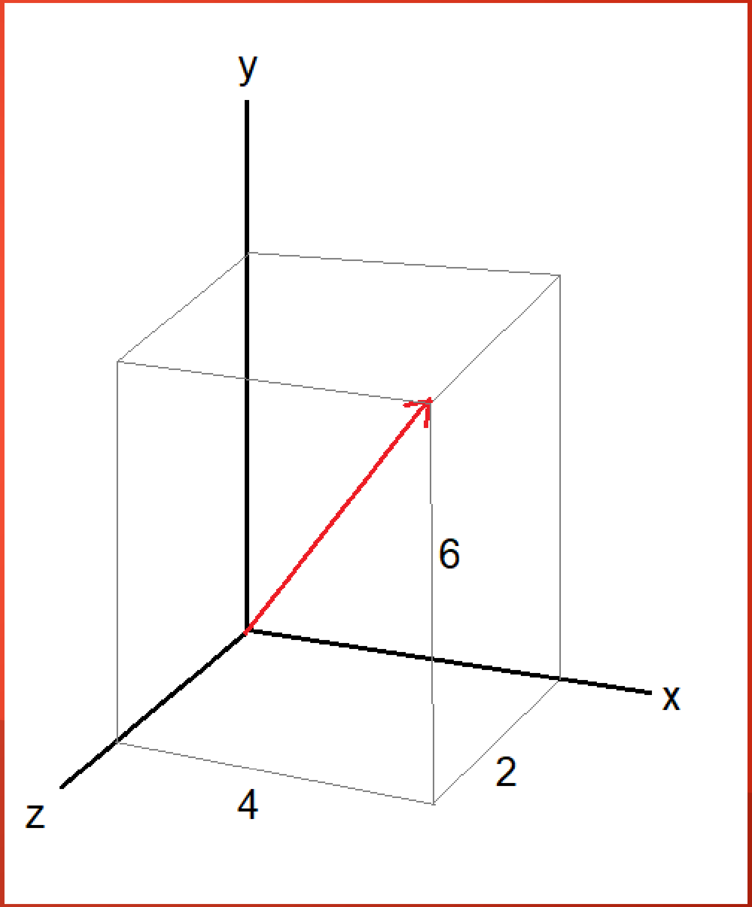
Vector Overview

- A quantity with magnitude and direction.
- In other words: How far something is and in what direction.
- Can be used for lots of things, normally to represent a direction, or something's position (e.g. how far and in what direction something is, relative to a certain point).

Vector Overview

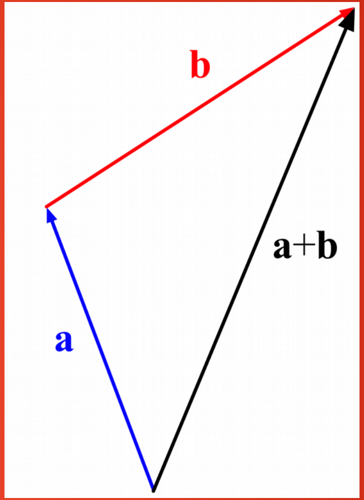
- x = 4, y = 6, z = 2
- v = [4, 6, 2]

$$v = \begin{bmatrix} 4 \\ 6 \\ 2 \end{bmatrix}$$

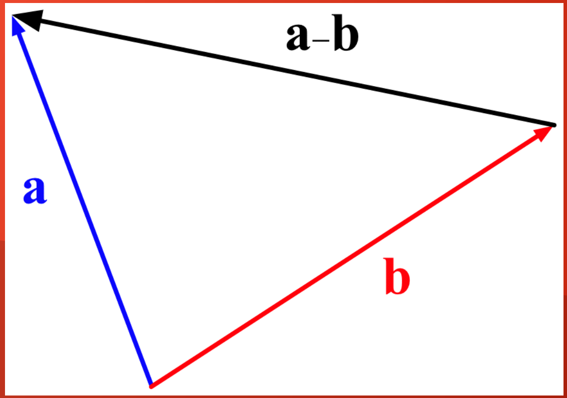


Vector Overview: Operations

- Addition: [1, 2, 3] + [2, 4, 6]
  = [1+2, 2+4, 3+6]
  = [3, 6, 9]



- Subtraction: [1, 2, 3] – [2, 4, 6]
  = [1-2, 2-4, 3-6]
  = [-1, -2, -3]



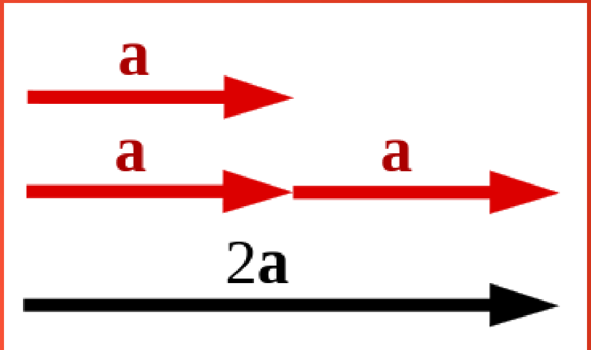
Vector Overview: Operations

- Multiplication by Scalar:

  [1, 2, 3] x 2

  = [1 x 2, 2 x 2, 3 x 2]

  = [2, 4, 6]



- Multiplication by Vector?
- Hard to define visually and not really used.
- Instead, use Dot Product!

Vector Overview: Dot Product

- Also called “Scalar Product” because it returns a scalar value (single value) as opposed to a vector.
- Can be done in two ways:
  - $[a, b, c] \cdot [d, e, f] = a \times d + b \times e + c \times f$

    $[1, 2, 3] \cdot [4, 5, 6] = 1 \times 2 + 2 \times 5 + 3 \times 6 = 2 + 10 + 18 = \mathbf{30}$

  - $v_1 \cdot v_2 = |v_1| \times |v_2| \times \cos(\theta)$

    $|v_1|$ is the “magnitude” or “length” of v_1

    θ is the angle between v_1 and v_2

Vector Overview: Magnitude

- Vectors form right-angle triangles.
- So we can calculate magnitude with a variation of the Pythagorean Theorem!
- In 3D, it's just: $|v| = \text{sqrt}(v_x^2 + v_y^2 + v_z^2)$
- $v = [1, 2, 2]$

  $|v| = \text{sqrt}(1^2 + 2^2 + 2^2)$

  $= \text{sqrt}(1 + 4 + 4) = \text{sqrt}(9)$

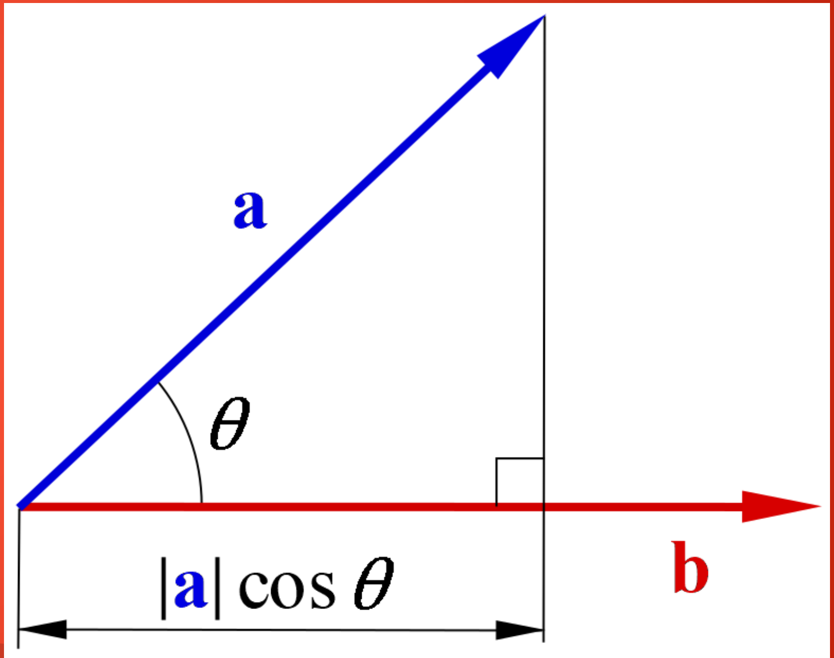
  $= 3$

Vector Overview: Dot Product

- This allows for some interesting scenarios…
- $v_1 \cdot v_2 = |v_1| \times |v_2| \times \cos(\theta)$
- If we know $v_1 \cdot v_2$ from alternative method…
- And we calculate the two magnitudes…
- $(v_1 \cdot v_2) / (|v_1| \times |v_2|) = \cos(\theta)$
- $\cos^{-1}((v_1 \cdot v_2) / (|v_1| \times |v_2|)) = \theta$
- More on this when we get to lighting!

Vector Overview: Dot Product Visualised

- Scalar Projection
- Dot Product with assumption that ‘b’ is a ‘unit vector’…
- Unit vector is a vector with magnitude ‘1’.
- If **a** and **b** are at right angles then projected length will be 0.
- Makes sense, because:
  |a| x cos(90) = |a| x 0 = 0
- So we can check for angles relative to this!
- Again, will be important in lighting.



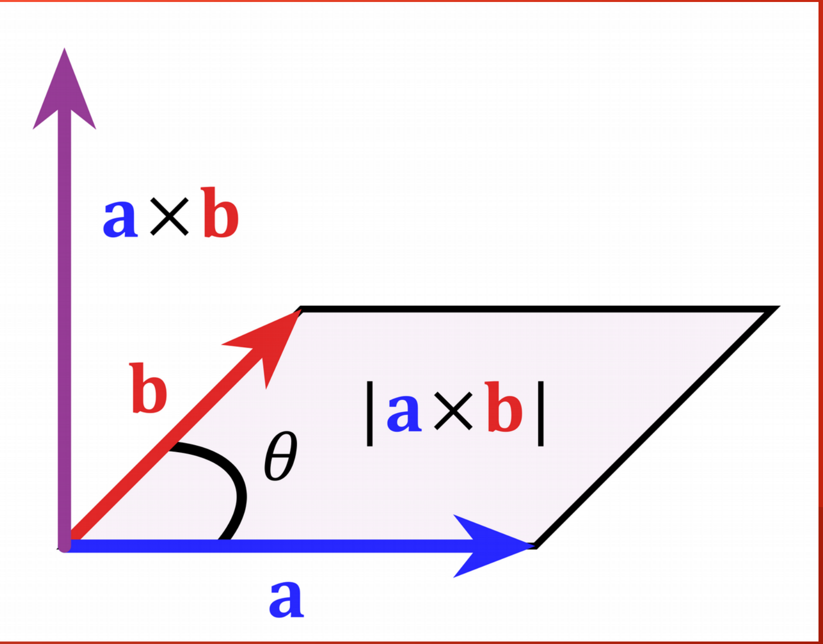
Vector Overview: Unit Vector

- Sometimes we only want to know a direction and how to advance in that direction.
- A unit vector is a vector with magnitude (length) of ‘1’.
- $u = v/|v|$
- $v = [1, 2, 2]$
- $|v| = \mathrm{sqrt}(1^2 + 2^2 + 2^3) = \mathrm{sqrt}(1 + 4 + 4) = \mathrm{sqrt}(9) = 3$
- $u = [1, 2, 2]/3 = [1/3, 2/3, 2/3]$
- **u** has same direction as **v** but is only one unit in magnitude!

Vector Overview: Cross Product

- Only really works in 3D.
- Creates a vector at right angles to two other vectors.

$$\begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} \times \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} = \begin{pmatrix} A_y \cdot B_z - A_z \cdot B_y \\ A_z \cdot B_x - A_x \cdot B_z \\ A_x \cdot B_y - A_y \cdot B_x \end{pmatrix}$$



Matrix Overview

- Group of values in an i x j grid.
- Example is a 2x3 matrix.
- i = rows

  j = columns

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

- Can be used for all sorts of things across graphics, game development and scientific fields.
- We will use them to handle model transforms (translation, rotation, scaling), projections and views.

Matrix Overview: Addition and Subtraction

- Scalar: Just add/subtract value from each element, much like with vectors.
- Matrix: Add the values on a per-element basis. Each one matches to its own position in the other matrix.
- This means the dimensions of the two matrices must match!

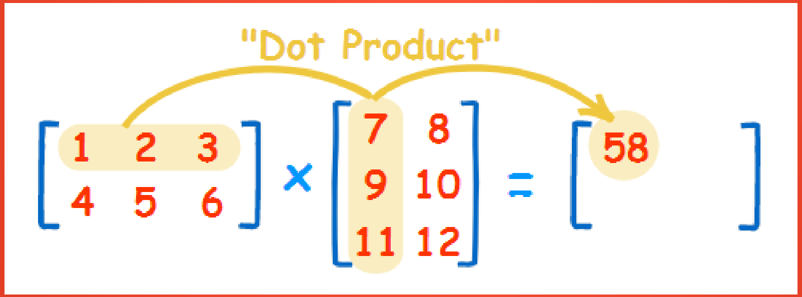
$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

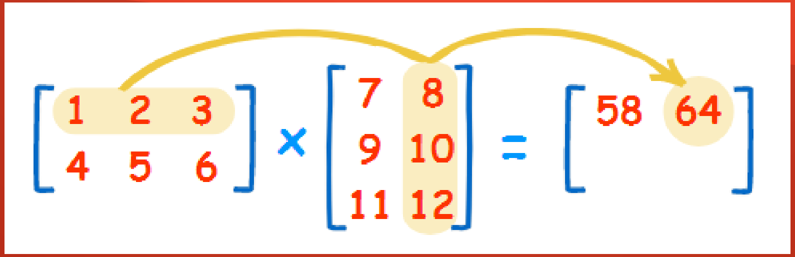
Matrix Overview: Multiplication

- Scalar: Just multiply value with each element, much like with vectors.
- Matrix: Things are a bit more complex…

Matrix Overview: Multiplication

- **ORDER MATTERS.**
- Columns on left-hand matrix **MUST** equal he number of Rows on the right-hand matrix.





Matrix Overview: Multiplication

- I'm not going to go in to more detail on this because it will take too long!
- The GLM library will handle all the maths for us.

- However, if there's enough demand for a separate lesson on Matrices, I'll add one.

Matrix Overview: Vectors

- How do Matrices work with Vectors?
- Vectors are just matrices with a single column!
- Multiplying a vector by a matrix will create a modified version of that vector.

$$v = \begin{bmatrix} 4 \\ 6 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 5 \\ 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 4 \\ 47 \\ 5 \\ 68 \end{bmatrix}$$

- Vector will always be on the right of the matrix.

Matrix Transforms

- Matrices can be used with vectors to apply transforms to them (translation, rotation, scaling…).
- Most basic is Identity Matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Simply returns the given vector!
- Act as a “starting point” for applying other transforms.

Matrix Transforms: Translation

- Translation “moves” the vector.
- Use it for changing the position of something.

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + X \\ y + Y \\ z + Z \\ 1 \end{bmatrix}$$

Matrix Transforms: Scaling

- Scaling “resizes” a vector.
- Can be used to increase a distance by a factor, or more commonly, to make an object larger.

$$\begin{bmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} SX \cdot x \\ SY \cdot y \\ SZ \cdot z \\ 1 \end{bmatrix}$$

Matrix Transforms: Rotation

- Rotation rotates a vector.
- Should be thought of as rotating around its origin…
- So to choose a point of rotation, translate the vector so the point to rotate around is at the origin.

- Three different matrices for handling rotation!

Matrix Transforms: Rotation

- X Rotation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ \cos\theta \cdot y - \sin\theta \cdot z \\ \sin\theta \cdot y + \cos\theta \cdot z \\ 1 \end{bmatrix}$$

- Y Rotation:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{bmatrix}$$

- Z Rotation:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{bmatrix}$$

Matrix Transforms

- YOU DON'T HAVE TO REMEMBER ALL OF THIS!
- However, it helps to know how and why it's working.
- GLM (OpenGL Mathematics) will do most of the matrix maths for us.

Matrix Transforms: Combining

- To combine transforms, just multiply them.

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Then apply to the vector.

$$\begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 2x + 1 \\ 2y + 2 \\ 2z + 3 \\ 1 \end{bmatrix}$$

Matrix Transforms: Combining

- Remember: **ORDER MATTERS!**

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Transforms happen in reverse order: The scale is applied first, and then the translation.
- If you swapped them around, the translation would be applied, and then the scale would be applied…
- So the scale would also scale the transform!

GLM

- GLM is a free library for handling common mathematical operations used with OpenGL.
- Most importantly: Vectors and Matrices.
- Uses vec4 (vector with 4 values) and mat4 (4x4 matrix) types.
- Simple code:

```
glm::mat4 trans;
trans = glm::translate(trans, glm::vec3(1.0f, 2.0f, 3.0f));
```

Uniform Variables

- Type of variable in shader.
- Uniforms are values global to the shader that aren't associated with a particular vertex.

```
#version 330

in vec3 pos;

uniform mat4 model;

void main()
{
    gl_Position = model * vec4(pos, 1.0);
}
```

Uniform Variables

- Each uniform has a location ID in the shader.
- Need to find the location so we can bind a value to it.

```
int location = glGetUniformLocation(shaderID, "uniformVarName");
```

- Now we can bind a value to that location.

```
glUniform1f(location, 3.5f);
```

- Make sure you have set the appropriate shader program to be in use!

Uniform Variables

- Different variable types:

  glUniform1f – Single floating value.

  glUniform1i – Single integer value.

  glUniform4f – vec4 of floating values.

  glUniform4fv – vec4 of floating values, value specified by pointer.

  glUniformMatrix4fv – mat4 of floating values, value specified by pointer.

  etc...

Summary

- Vectors are directions and positions in space.
- Matrices are 2-dimensional arrays of data used for calculating transforms and various other functions.
- Vectors are a type of matrix and can have these functions applied to them.
- The order of transform operations matters!!
- Last matrix operation applied happens first.
- GLM is used to handle matrix calculations.
- Uniform variables pass global data to shaders.
- Need to obtain a uniform’s location then bind data to it.

See you next video!