

# Introduction to Shader development

h\_da WS2020/21

Paul Nasdalack

[info@paul-nasdalack.com](mailto:info@paul-nasdalack.com)

[!\[\]\(c3d993ca47bfe2a953c700506ce31fa0\_img.jpg\) @littleBugHunter](https://twitter.com/littleBugHunter)



# Vectors and Matrices





# Vectors and Matrices



# Vectors and Matrices

A **vector space** (also called a **linear space**) is a collection of objects called **vectors**, which may be [added](#) together and [multiplied](#) ("scaled") by numbers, called [scalars](#) in this context. Scalars are often taken to be [real numbers](#), but there are also vector spaces with scalar multiplication by [complex numbers](#), [rational numbers](#), or generally any [field](#). The operations of vector addition and scalar multiplication must satisfy certain requirements, called [axioms](#), listed [below](#).

*([https://en.wikipedia.org/wiki/Vector\\_space](https://en.wikipedia.org/wiki/Vector_space))*

# Vectors and Matrices

What are Vectors to us?

# What are Vectors to us?

- A Vector is:
  - Multiple numbers, that somehow belong together
- Popular Vectors are:
  - Positions (2 or 3 Numbers XY[Z])
  - Colors (3 or 4 Numbers RGB[A])
  - Directions (2 or 3 Numbers XY[Z])
- Basically everything in Shaders is represented by a Vector or a Scalar

What are Vectors to us?

Scalar?

What are Vectors to us?

Scalar?

Just a fancy term for single Number



What are Vectors to us?

What can we do with Vectors?

# What can we do with Vectors?

- Add/Subtract
- Multiply
  - Dot product
  - Cross product
  - Scalar product
- Divide???

# Add/Subtract

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 6 \end{bmatrix}$$

# Add/Subtract

3	+	1	=	4
2	+	5	=	7
4	+	2	=	6

# What are Vectors to us?

- Vector

HLSL/CG Syntax:

```
float a = 3.41; //normal scalar  
float3 b = float3(0.2,0.32,0.4) //vector
```

# Add/Subtract

HLSL/CG Syntax:

$a+b$

$a-b$

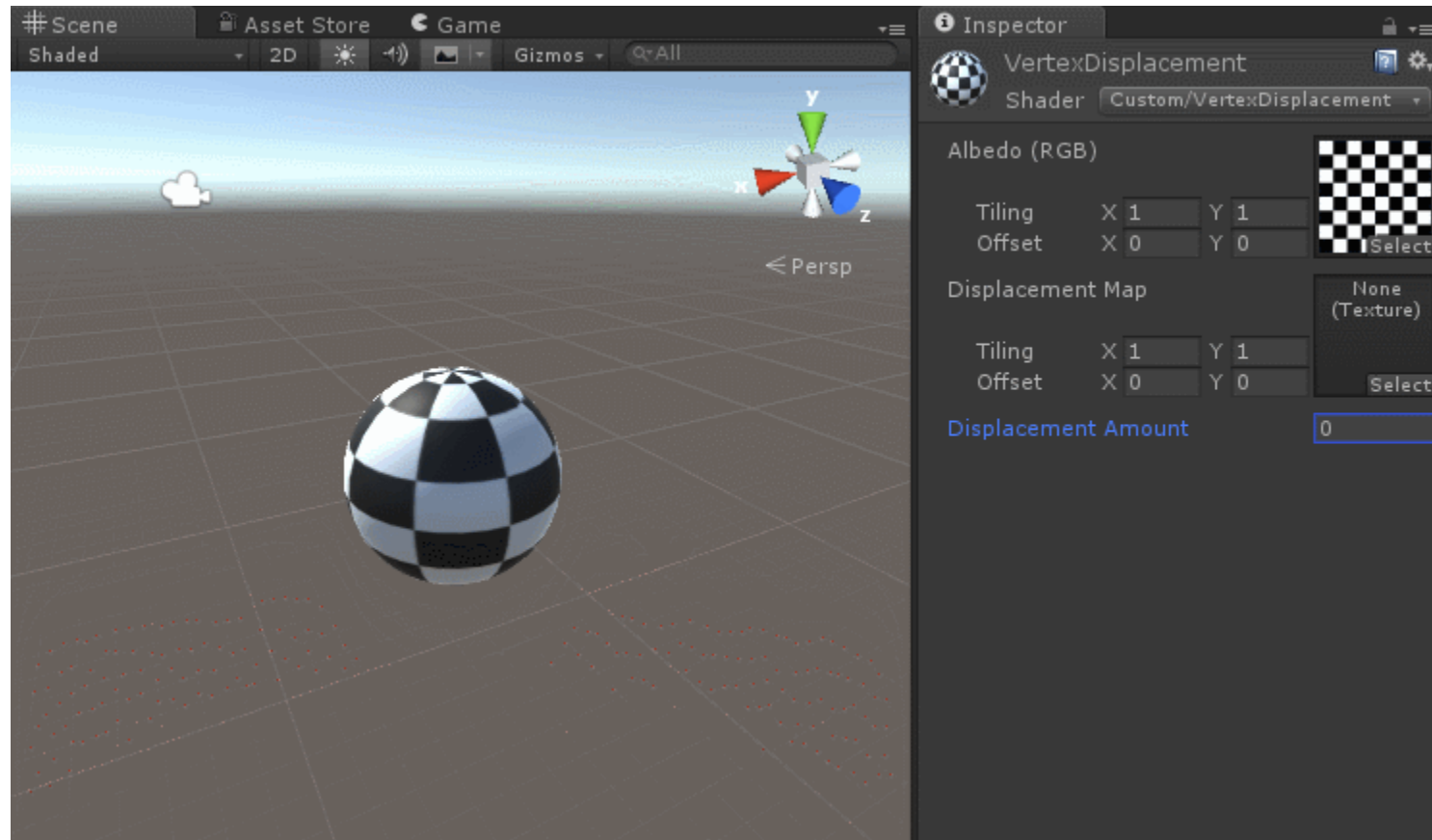


Add/Subtract

What can we do with Add/Subtract?

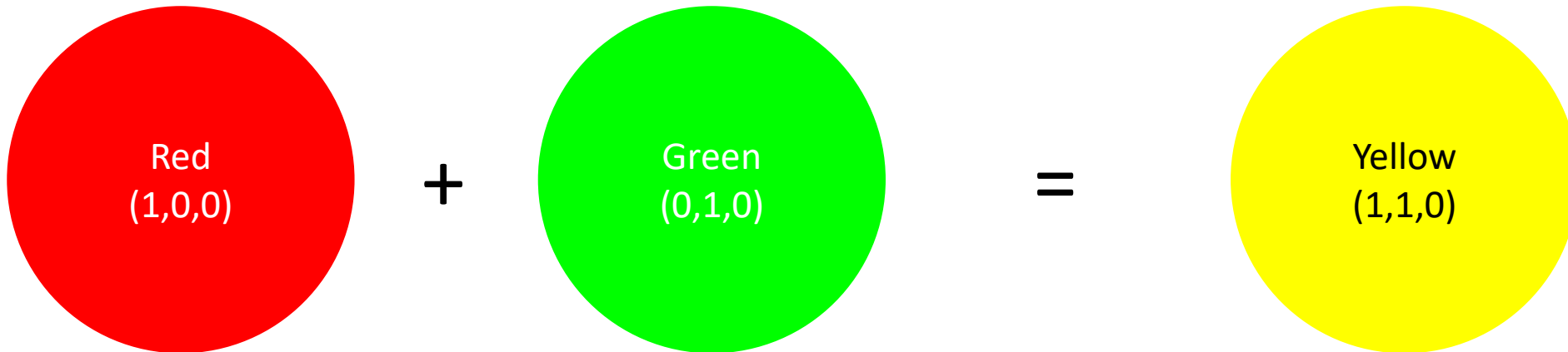
# What can we do with Add/Substract?

Move Positions around (aka Vertex Displacement)



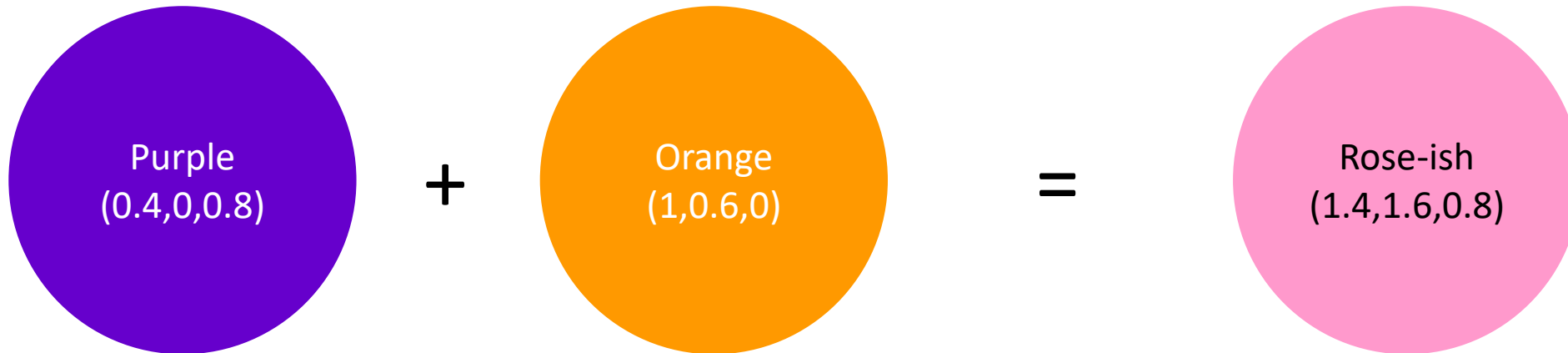
# What can we do with Add/Subtract?

Add Colors together



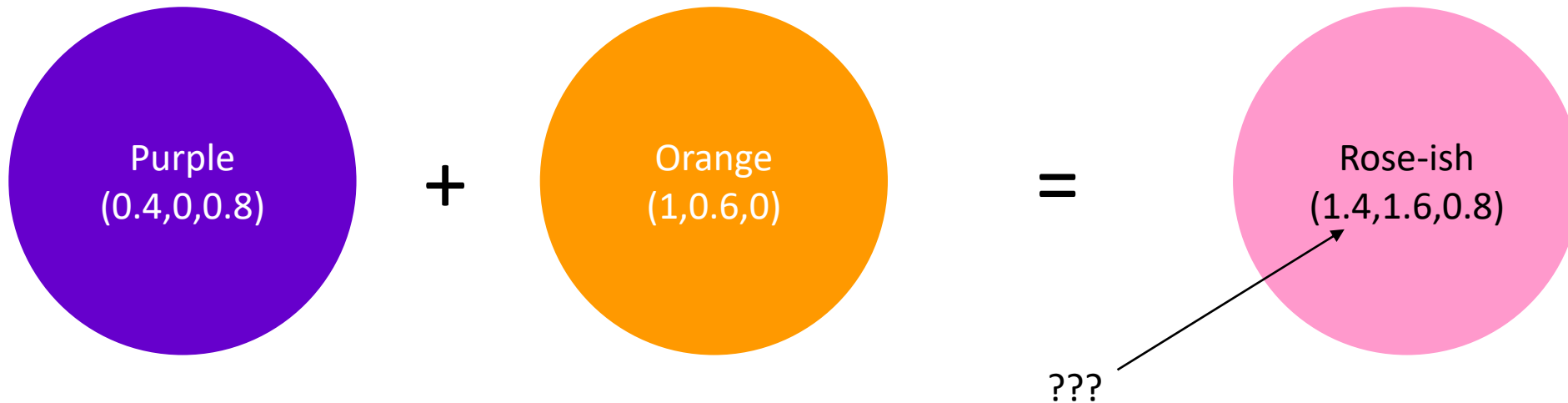
# What can we do with Add/Substract?

Add Colors together



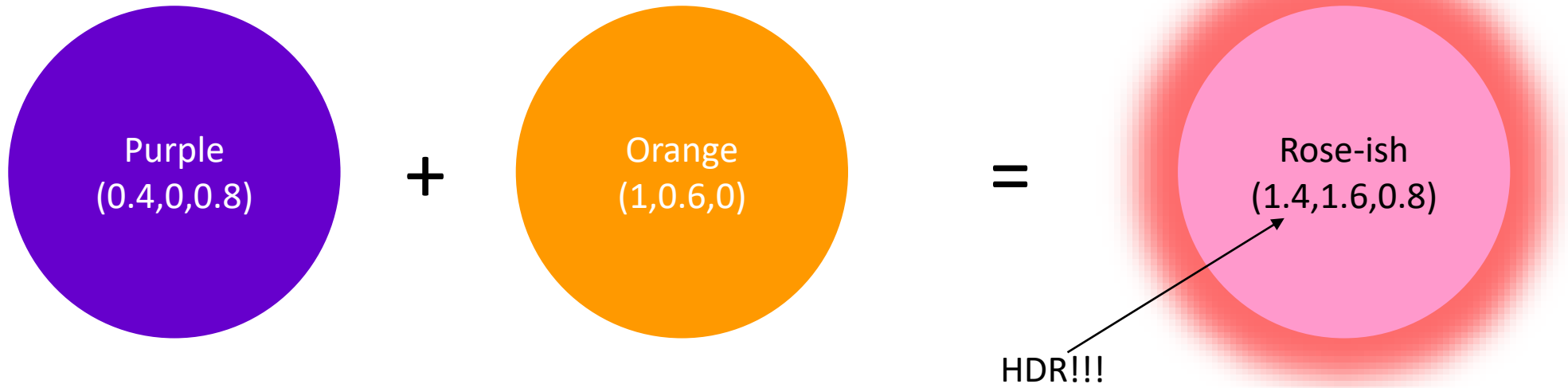
# What can we do with Add/Substract?

Add Colors together



# What can we do with Add/Subtract?

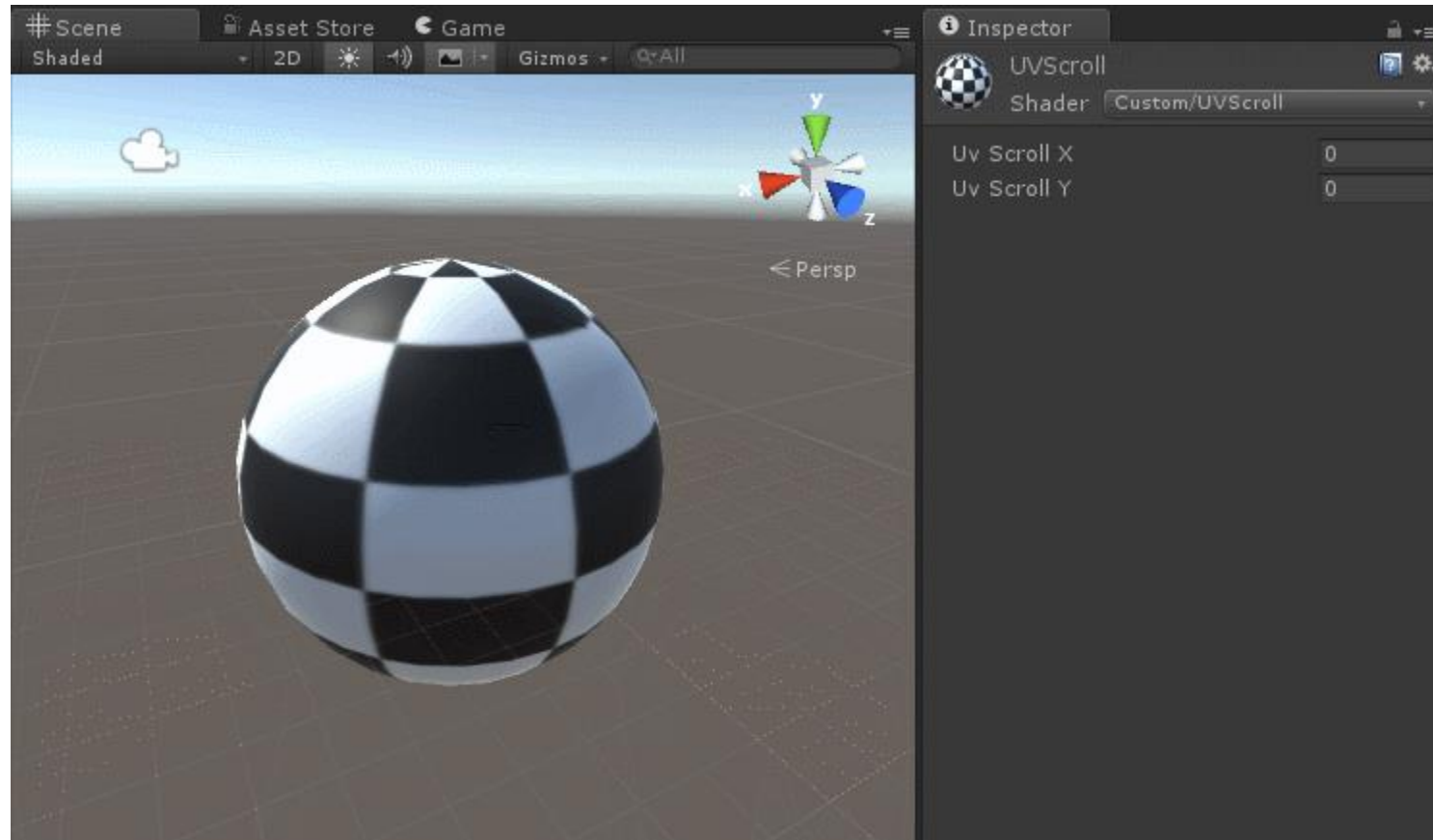
Add Colors together





# What can we do with Add/Subtract?

Scroll UVs around



# Multiplication (Scalar Product)

$$3 * \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 15 \\ 6 \end{bmatrix}$$

# Multiplication (Scalar Product)

	$\begin{pmatrix} 3 \end{pmatrix}$	*	$\begin{pmatrix} 1 \end{pmatrix}$		$\begin{pmatrix} 3 \end{pmatrix}$	
	3	*	5	$\underline{\hspace{1cm}}$ $\underline{\hspace{1cm}}$	15	
	3	*	2		6	

# Multiplication (Scalar Product)

$$\begin{bmatrix} 2 \\ 4 \\ 5 \end{bmatrix} \begin{matrix} * \\ * \\ * \end{matrix} \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 20 \\ 10 \end{bmatrix}$$

DISCLAIMER!

This is not very mathematical, but it's just too darn useful, so we've built it into Shaders anyway

# Multiplication (Scalar Product)

HLSL/CG Syntax:

$a * b$

$3 * b$

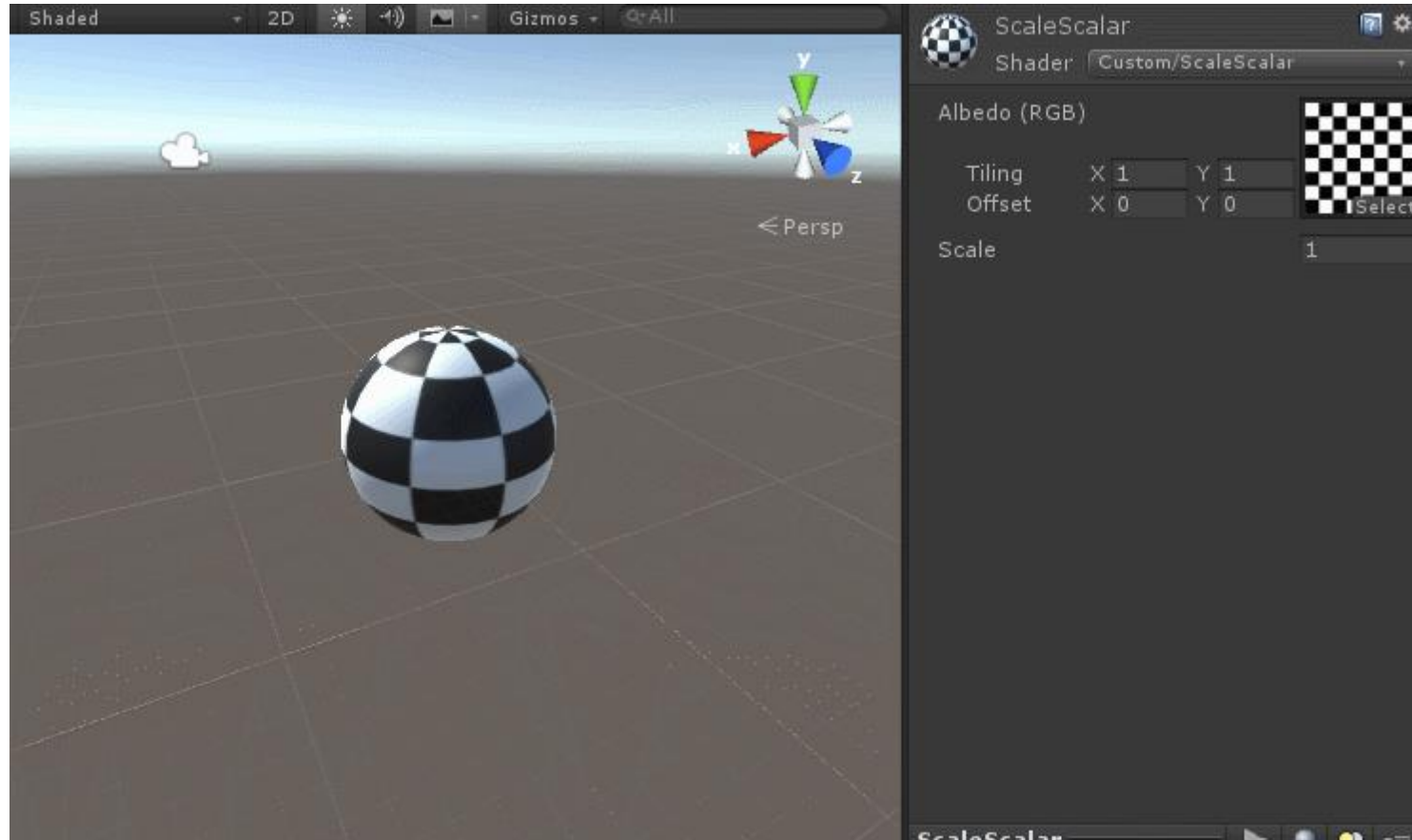
# Multiplication (Scalar Product)

What can we do with Scalar Products?



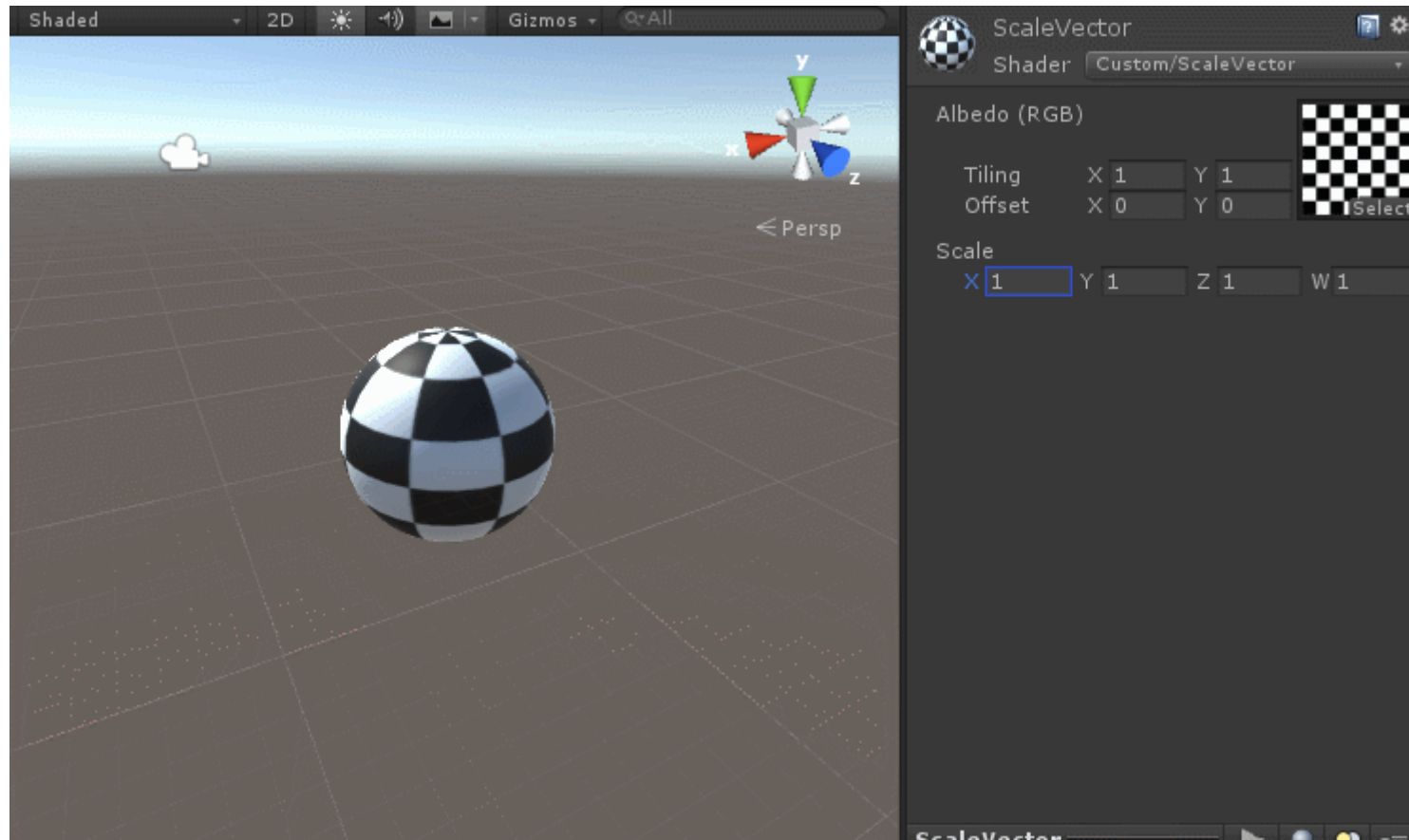
# What can we do with Scalar Products?

Scale things! (using a Scalar)



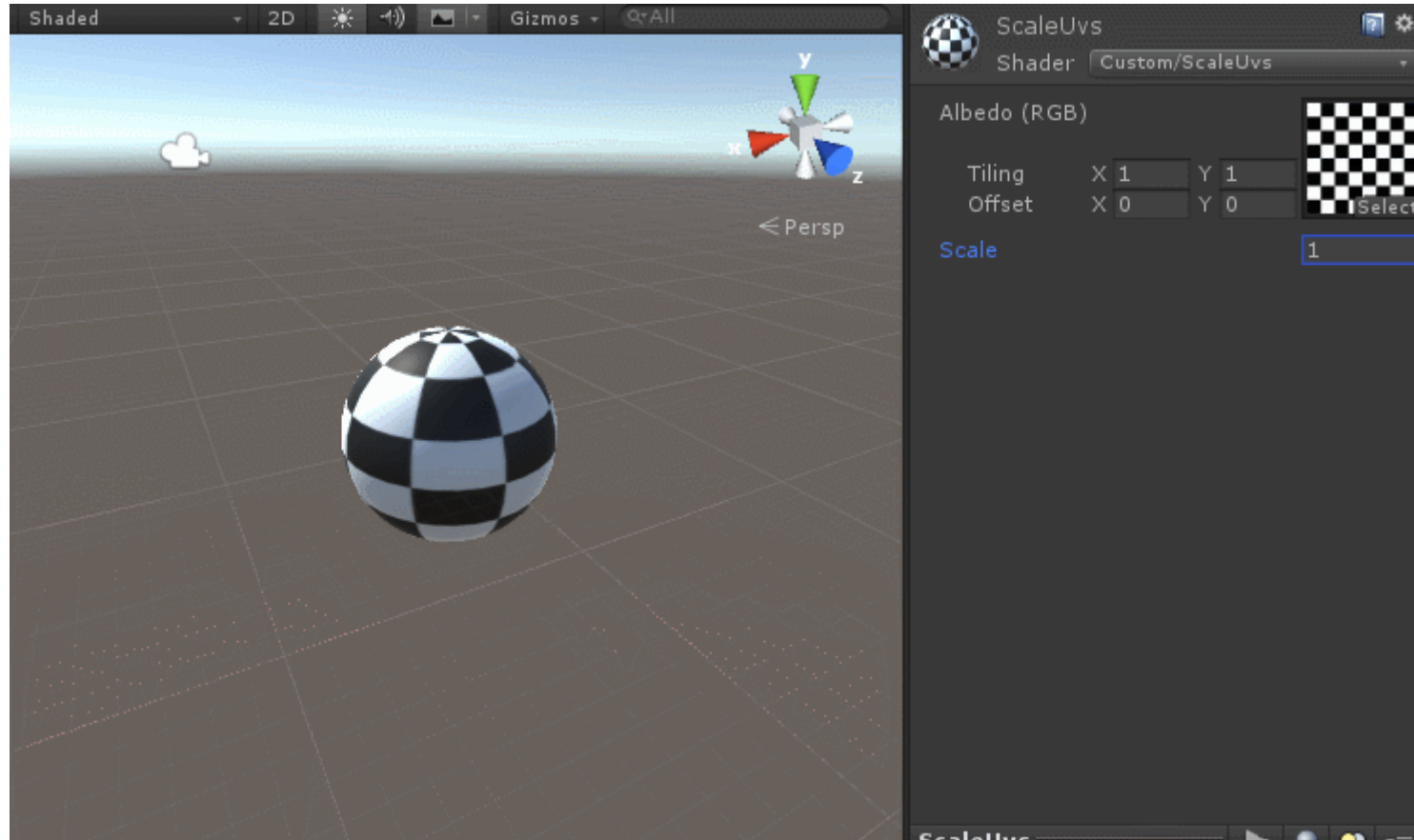
# What can we do with Scalar Products?

Scale things! (using a Vector)



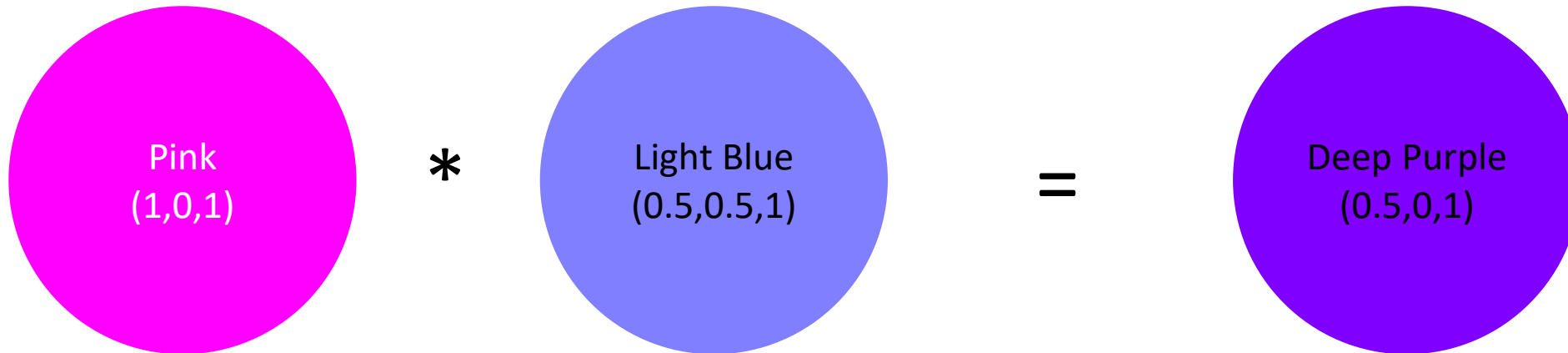
# What can we do with Scalar Products?

Scale things! (UVs)



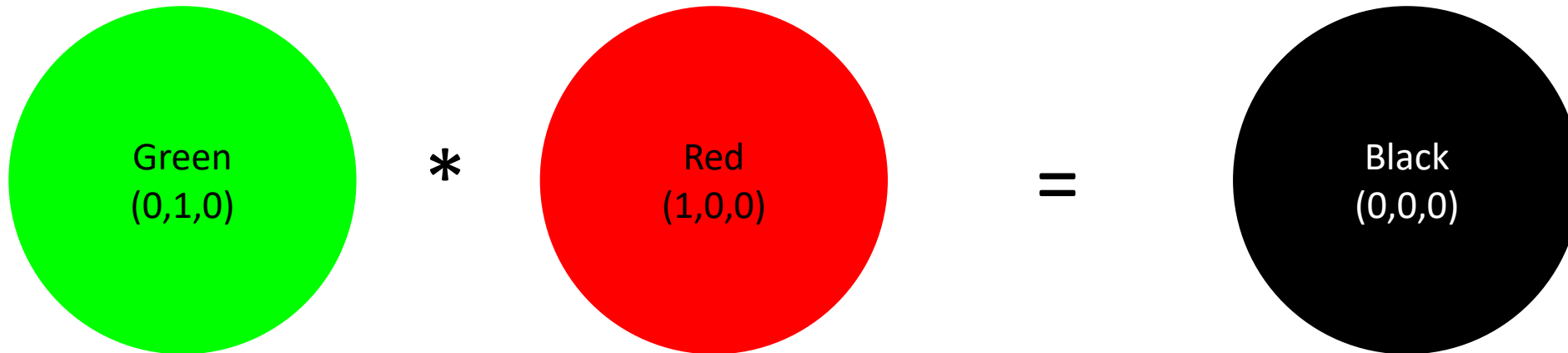
# What can we do with Scalar Products?

Multiply Colors



# What can we do with Scalar Products?

Multiply Colors



# What can we do with Scalar Products?

Multiply Colors



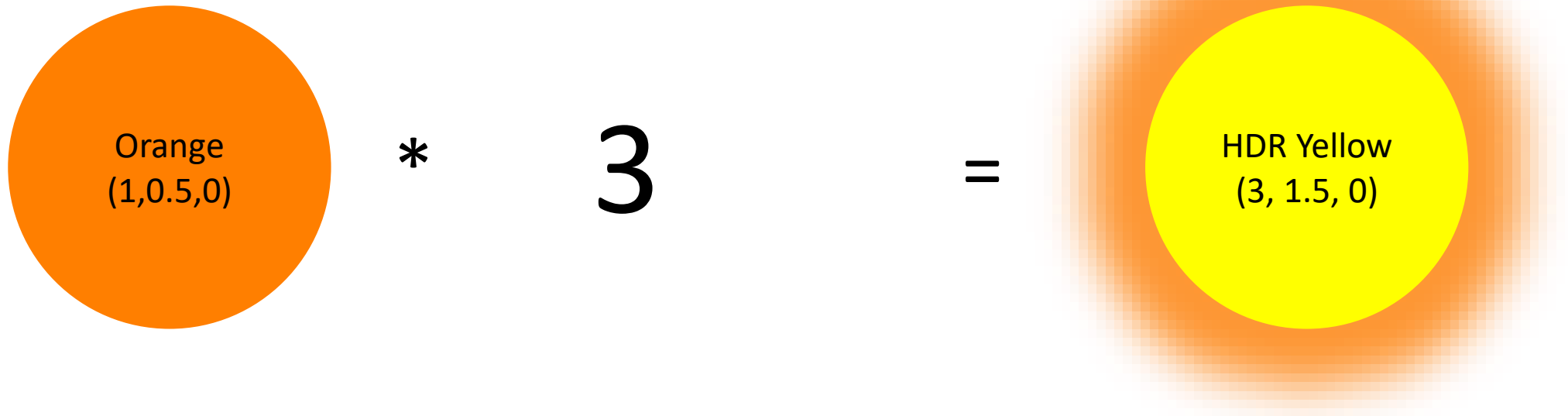
The diagram illustrates the scalar multiplication of a color vector. On the left, an orange circle contains the text "Orange" and the vector  $(1, 0.5, 0)$ . This is followed by a multiplication symbol  $*$ , the scalar  $0.5$ , an equals sign  $=$ , and a dark orange circle on the right containing the text "Dark Orange" and the resulting vector  $(0.5, 0.25, 0)$ .

$$\begin{matrix} \text{Orange} \\ (1, 0.5, 0) \end{matrix} * 0.5 = \begin{matrix} \text{Dark Orange} \\ (0.5, 0.25, 0) \end{matrix}$$



# What can we do with Scalar Products?

Multiply Colors



# Multiplication (Dot Product)

*(aka: the proper way to multiply Vectors)*

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = 21$$

# Multiplication (Dot Product)

*(aka: the proper way to multiply Vectors)*

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = 21?$$

# Multiplication (Dot Product)

*(aka: the proper way to multiply Vectors)*

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = (3*1)+(2*5)+(4*2)$$

# Multiplication (Dot Product)

*(aka: the proper way to multiply Vectors)*

HLSL/CG Syntax:

```
dot(a, b);
```

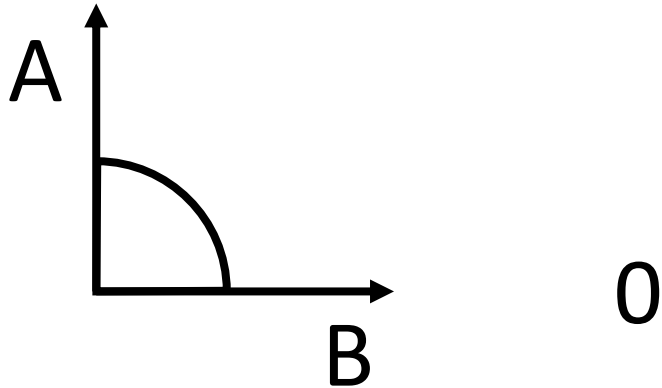
# Multiplication (Dot Product)

*(aka: the proper way to multiply Vectors)*

## What can we do with Dot Products?

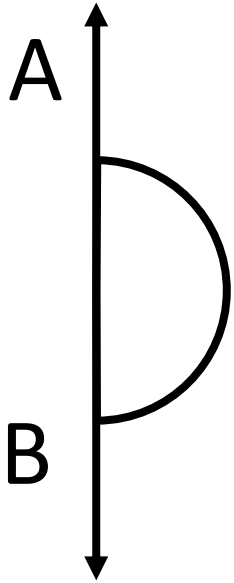
# What can we do with Dot Products?

Tell the angle between two (normalized) Vectors



# What can we do with Dot Products?

Tell the angle between two (normalized) Vectors

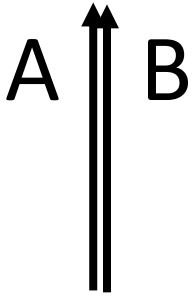


-1



# What can we do with Dot Products?

Tell the angle between two (normalized) Vectors

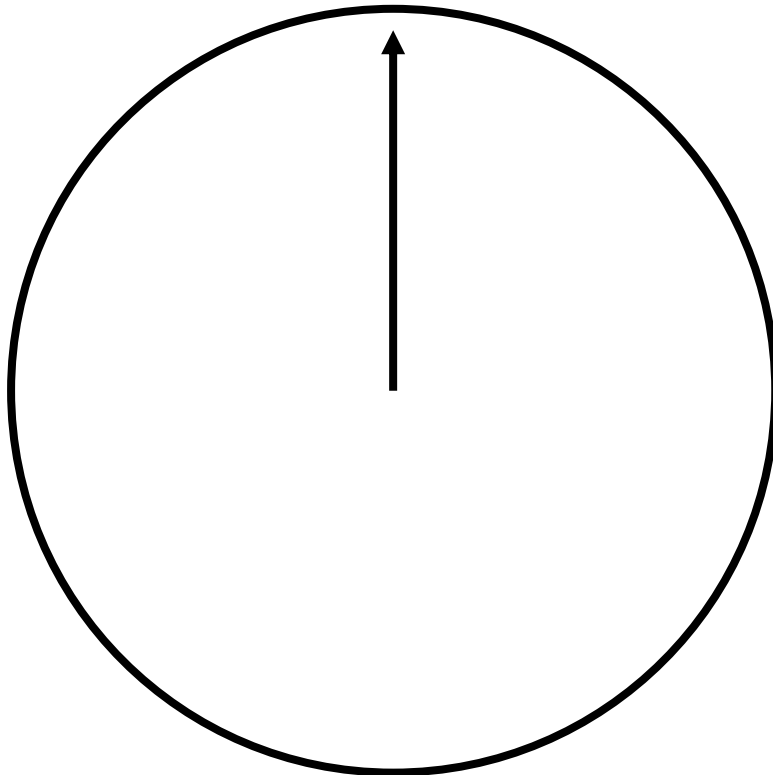


1

# What can we do with Dot Products?

## Normalized Vector?

A Vector with a length of 1 (usually directional Vectors are normalized)



# What can we do with Dot Products?

Normalized Vector?

HLSL/CG Syntax:

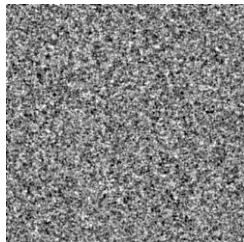
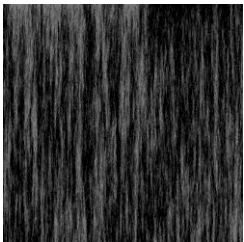
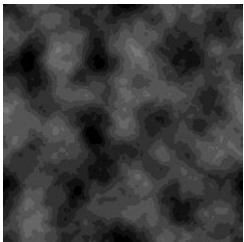
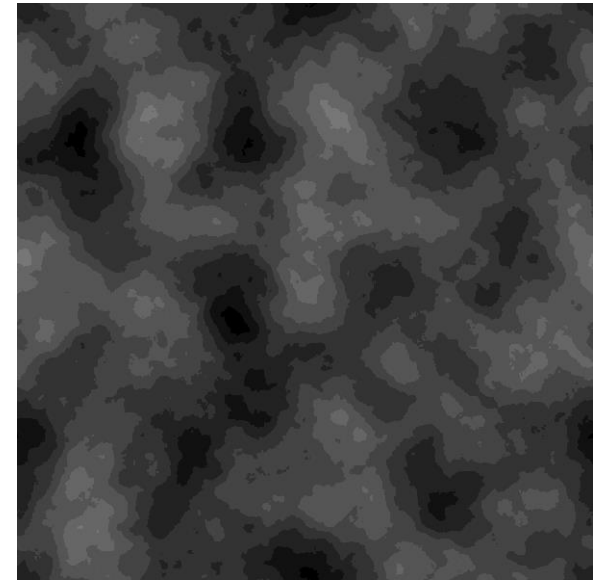
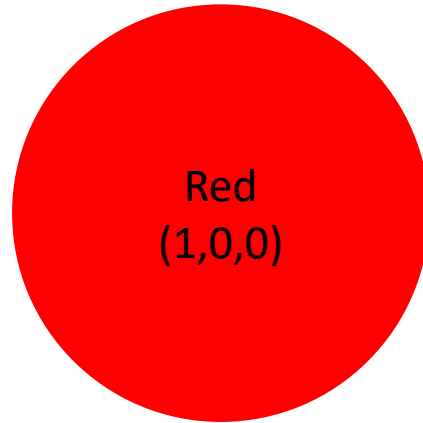
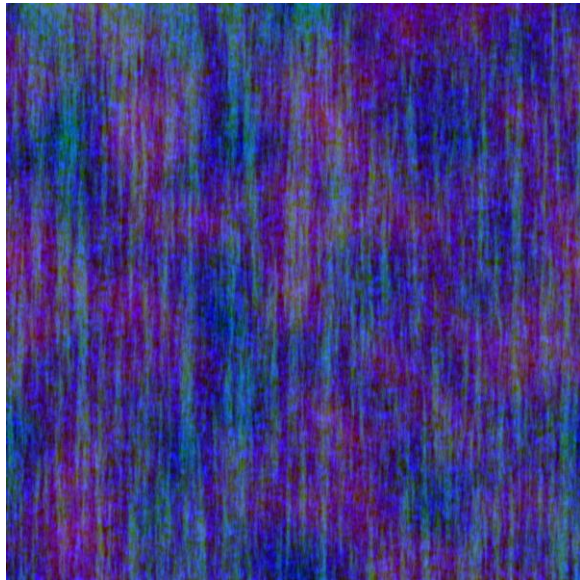
```
normalize(a);
```

What can we do with Dot Products?

**Back to Dot Products!**

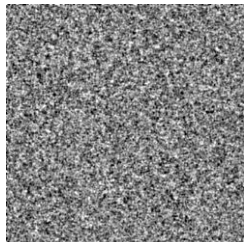
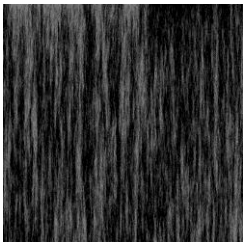
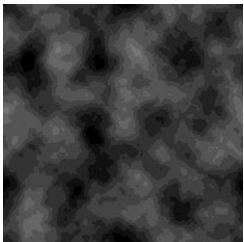
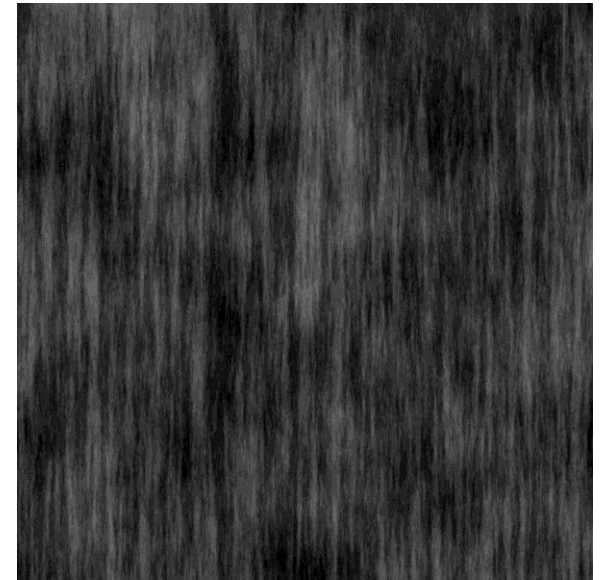
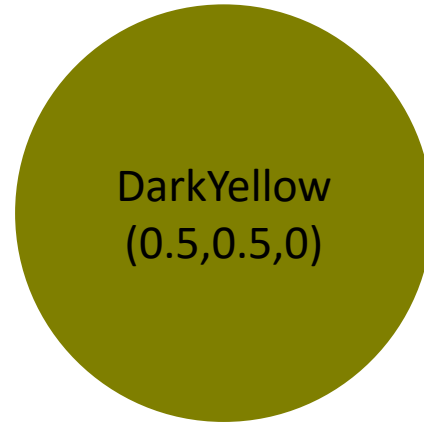
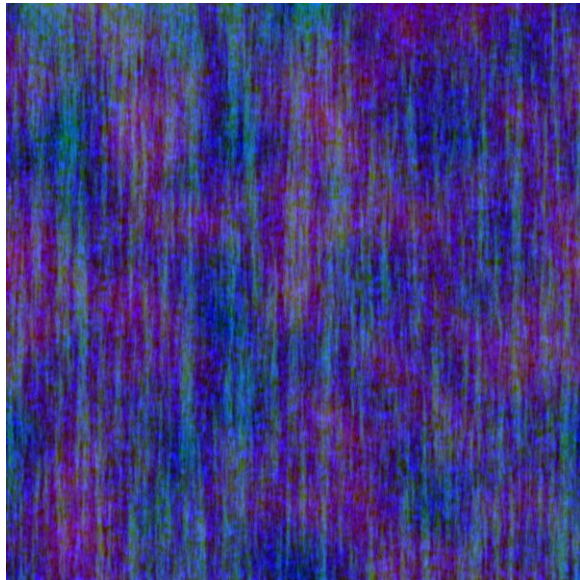
# What can we do with Dot Products?

## Colormask Textures



# What can we do with Dot Products?

## Colormask Textures



# Multiplication (Cross Product)

$$\begin{bmatrix} 1 \\ -7 \\ 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -30 \\ 1 \\ 37 \end{bmatrix}$$

# Multiplication (Cross Product)

$$\begin{bmatrix} 1 \\ -7 \\ 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -7*4-1*2 \\ ? \\ ? \end{bmatrix}$$



# Multiplication (Cross Product)

$$\begin{bmatrix} 1 \\ -7 \\ 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -7*4-1*2 \\ 1*5-1*4 \\ ? \end{bmatrix}$$

*Note: The first two vectors and the cross product symbol are shown in a faded gray color in the original image.*

# Multiplication (Cross Product)

$$\begin{bmatrix} 1 \\ -7 \\ 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -7*4-1*2 \\ 1*5-1*4 \\ 1*2- -7*5 \end{bmatrix}$$

# Multiplication (Cross Product)

$$\begin{bmatrix} 1 \\ -7 \\ 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -30 \\ 1 \\ 37 \end{bmatrix}$$

# Multiplication (Cross Product)

HLSL/CG Syntax:

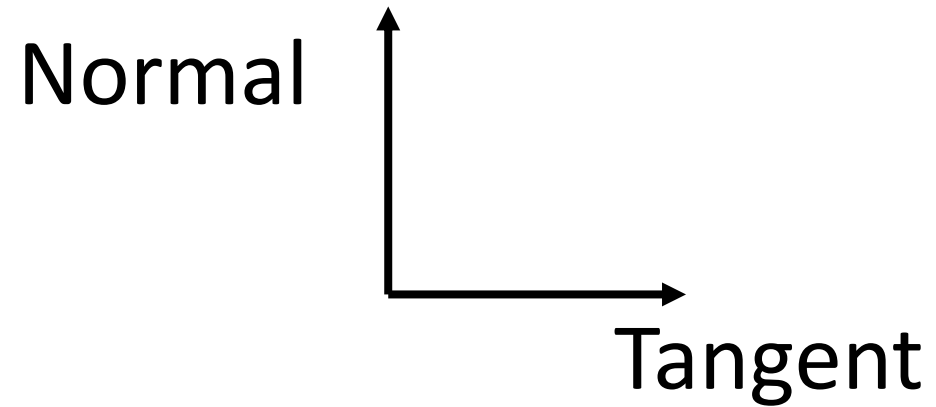
```
cross(a, b);
```

# Multiplication (Cross Product)

What can we do with Cross Products?

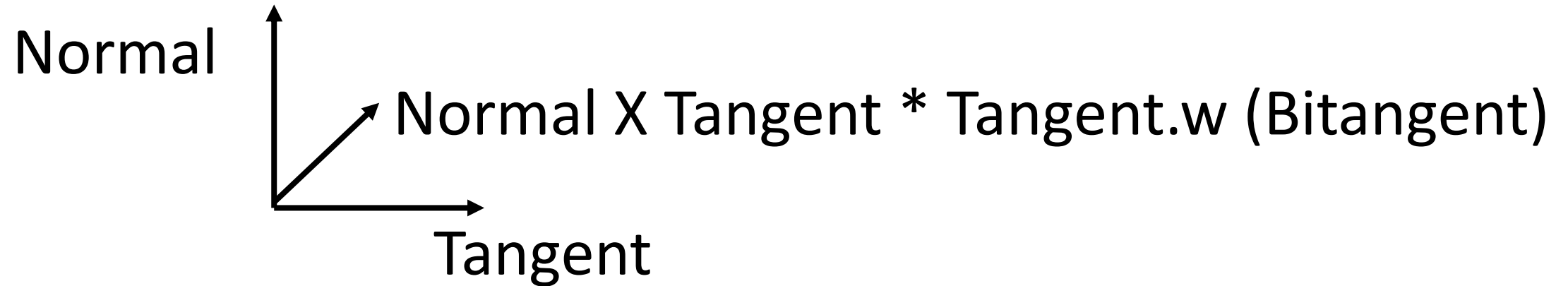
# What can we do with Cross Products?

Encode the Bitangent



# What can we do with Cross Products?

Encode the Bitangent



# Division?

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} / \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = ?$$

DISCLAIMER!

This is not very mathematical, but it's just too darn useful, so we've built it into Shaders anyway



# Division

$$\begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} / \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 0.4 \\ 0.2 \end{bmatrix}$$

# Division

3	/	1		3
2	/	5	— —	0.4
4	/	2		0.2

## DISCLAIMER!

This is not very mathematical, but it's just too darn useful, so we've built it into Shaders anyway

# Division

HLSL/CG Syntax:

```
a/b;
```

Swizzle!!!

# Swizzle

HLSL/CG Syntax:

```
a.xyz = b.xxy;  
a.rgb = b.aaa;  
a.rgb = b.xyzw;
```

# Vectors and Matrices



# Matrices

***The Matrix*** is a 1999 American-Australian [science fiction action film](#) written and directed by [The Wachowskis](#), starring [Keanu Reeves](#), [Laurence Fishburne](#), [Carrie-Anne Moss](#), [Hugo Weaving](#), and [Joe Pantoliano](#). It depicts a [dystopian](#) future in which reality as perceived by most humans is actually a [simulated reality](#) called "the Matrix", created by sentient machines to subdue the human population, while their bodies' heat and electrical activity are used as an energy source. Computer programmer "[Neo](#)" learns this truth and is drawn into a rebellion against the machines, which involves other people who have been freed from the "dream world".

([https://en.wikipedia.org/wiki/The\\_Matrix](https://en.wikipedia.org/wiki/The_Matrix))

# Matrices

In [mathematics](#), a **matrix** (plural **matrices**) is a [rectangular array](#)<sup>[1]</sup> of [numbers](#), [symbols](#), or [expressions](#), arranged in [rows](#) and [columns](#).<sup>[2][3]</sup> For example, the dimensions of matrix ([1](#)) are  $2 \times 3$  (read "two by three"), because there are two rows and three columns.

$$\begin{bmatrix} 1 & 9 & -13 \\ 20 & 5 & -6 \end{bmatrix}$$

([https://en.wikipedia.org/wiki/Matrix\\_\(mathematics\)](https://en.wikipedia.org/wiki/Matrix_(mathematics)))



# Matrices

- Matrices come in all kinds of sizes
  - 3x3 Matrix
  - 4x4 Matrix
  - 4x3 Matrix
  - 3x4 Matrix
  - 2x2 Matrix

# Matrices

HLSL/CG Syntax:

```
float3x3 mat = {1,2,3,1,4,2,3,1,2};
```

# Matrices

What can we do with Matrices?

# What can we do with Matrices?

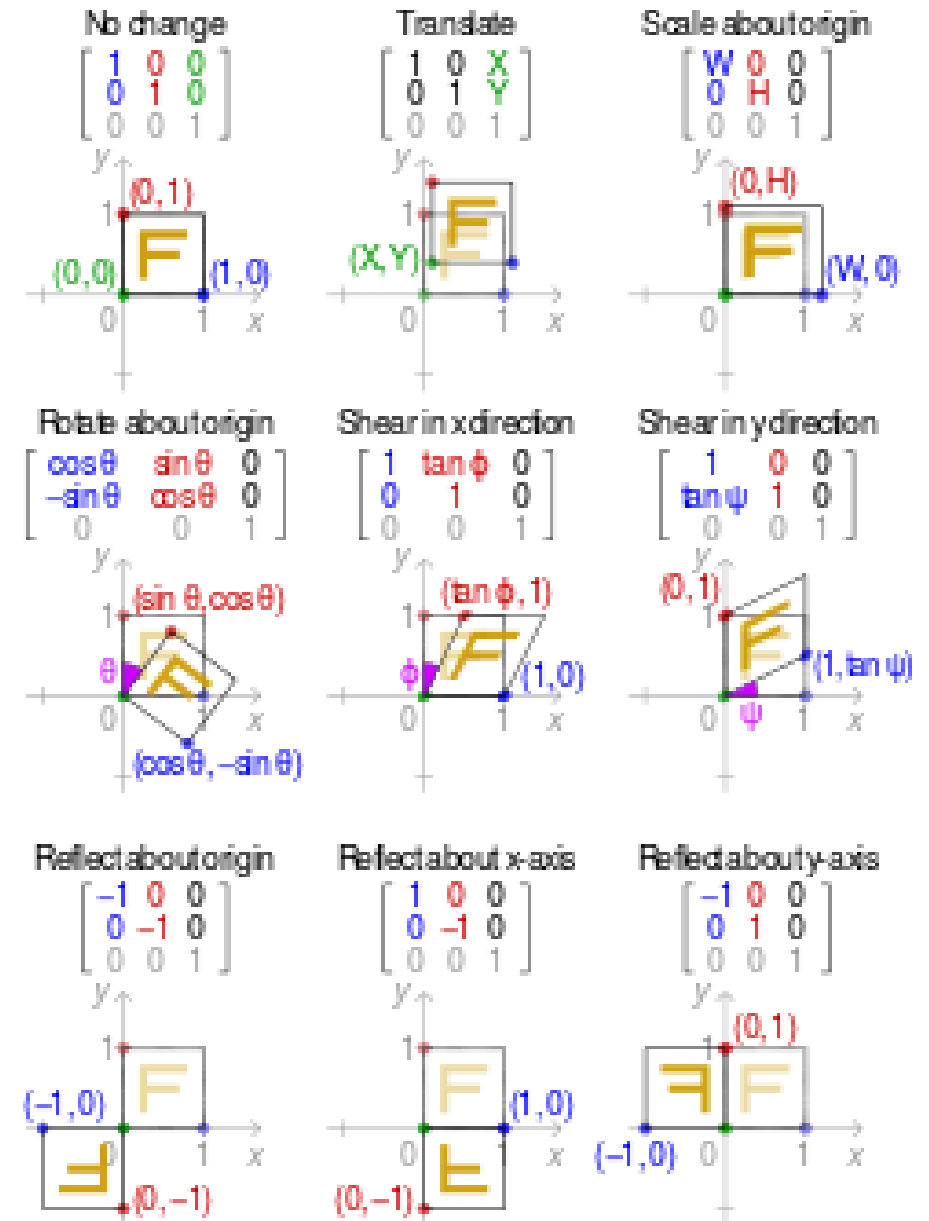
- Transform Positions
  - Move
  - Rotate
  - Scale
- Perspectively Distort Positions

# What can we do with Matrices?

- Multiply them with other Matrices
- Multiply them with Vectors

# How does it work?

- Transformation Matrices
- Multiply a Position with the Matrix



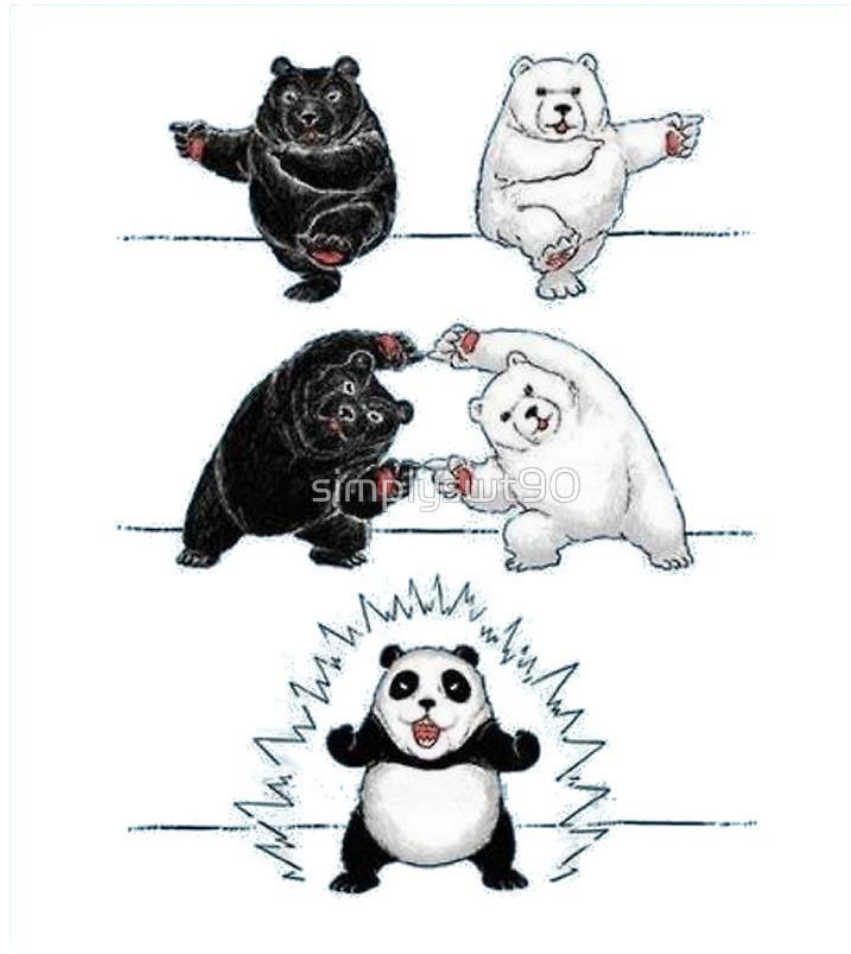
# Matrices

HLSL/CG Syntax:

```
mul(matA, vecB);
```

# How does it work?

- Multiply two matrices to combine their effects





# Matrices

HLSL/CG Syntax:

```
matC = mul(matA, matB);  
vecA = mul(matB, vecA);  
vecA = mul(matA, vecA);  
//multiplying with matC is the same as  
//first multiplying MatB and then MatA  
vecB = mul(matC, vecB);
```

# Popular Matrices (in Shaders)

- Model/World Matrix
  - Moves the model to its position
- View/Camera Matrix
  - Moves the „World“ in front of the camera
- Projection Matrix
  - Distorts the world perspectively
- MVP Matrix
  - all the above multiplied together


# Cheat Sheet!

Vector Math for Shader Programmers  
by Paul Nasdalack 26.10.2016


### Addition

$$\begin{pmatrix} 10 \\ 8 \\ 2 \end{pmatrix} + \begin{pmatrix} 6 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 16 \\ 12 \\ 7 \end{pmatrix}$$


Use Cases:



Color Addition



`vertex+=normal*displacement;`  
Vertex Displacement




`uv+=offset;`  
Texture Scrolling


### Scalar Product

$$3 * \begin{pmatrix} 6 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 18 \\ 12 \\ 15 \end{pmatrix}$$
$$\begin{pmatrix} 10 \\ 4 \\ 2 \end{pmatrix} * \begin{pmatrix} 6 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 60 \\ 16 \\ 10 \end{pmatrix}$$

Use Cases:



Color Multiplication




`vertex*=scale;`  
Scaling

### Dot Product

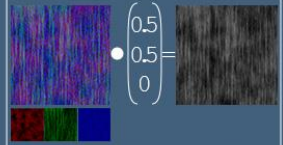
$$\begin{pmatrix} 10 \\ 4 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 6 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 16 \\ 12 \\ 15 \end{pmatrix}$$
$$10*6+4*4+2*5=86$$

Use Cases:



$\alpha$  is in range of -1 and 1

`lamert=dot(l,n);`  
Angle between two (normalized) Vectors




`fixed4 values=fixed4(1,0,0);`  
`fixed4 t=tex2D(tex,uv);`  
`fixed c=dot(t,values);`  
Texture channel decoding

### Cross Product

$$\begin{pmatrix} 1 \\ 7 \\ 1 \end{pmatrix} \times \begin{pmatrix} 5 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 26 \\ 1 \\ -33 \end{pmatrix}$$
$$\begin{pmatrix} 1 \\ 7 \\ 1 \end{pmatrix} \times \begin{pmatrix} 5 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 7*4-1*2 \\ 1*5-1*4 \\ 1*2-7*5 \end{pmatrix}$$

Use Cases:



`bitangent=cross(n,t)*t.w;`  
Get perpendicular vector to two other vectors

<http://bit.ly/2dQzwvD>