Animating With Math

Natalie Burke

Shader-Driven Vertex Animation

Vertex animation, or vertex manipulation, is the process of manipulating vertex positions on a mesh at runtime using the vertex shader assigned to the 3D model

The Vertex Shader

The Vertex Shader portion of an assets shader handles the processing of individual vertices at render time

Vertex shaders are able to make changes to a model's existing vertices

Allows the position of the vertices to be significantly modified in the render process

[Videos from *The Vanishing of Ethan Carter*]

DESTINY
BUNGIE

BUNGIE

BUNGIE

BUNGIE

Tool Kit

NODE EDITOR

World Space Offset

Normal Offset

Local Space Offset

MESH AND GAMEPLAY PROPERTIES

Vertex Color

Gameplay Properties

Animated Channels

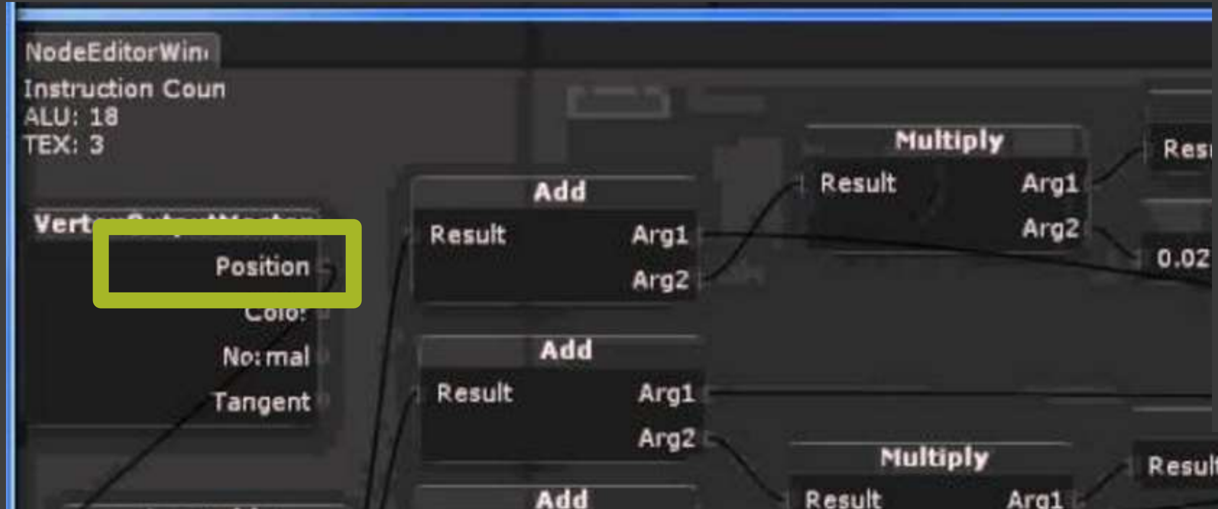
- ✓ Sin or Cosine
- ✓ Gameplay properties
- ✓ Vertex Color

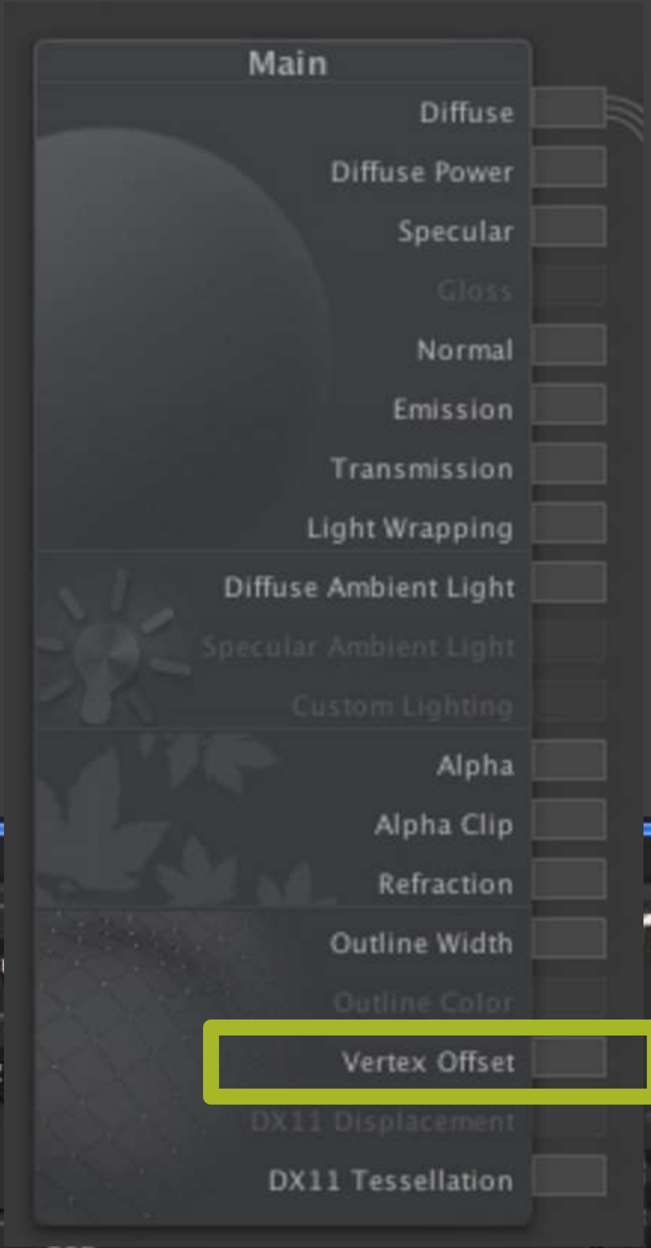
Try It Yourself

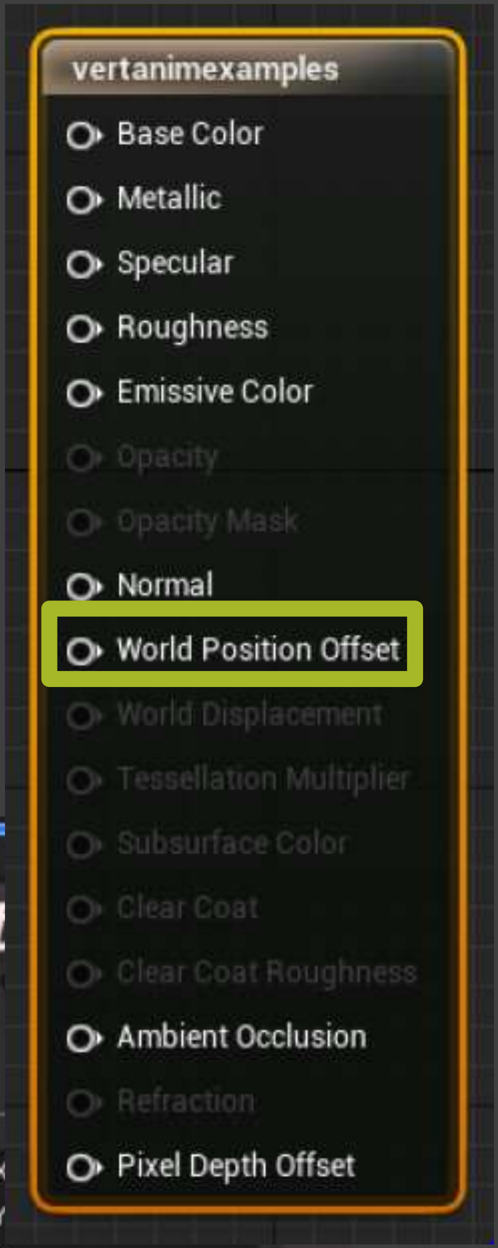
Strumpy Shader Editor, Unity - Vertex Graph

Shader Forge, Unity - Vertex Offset

Unreal Material Editor - World Position Offset

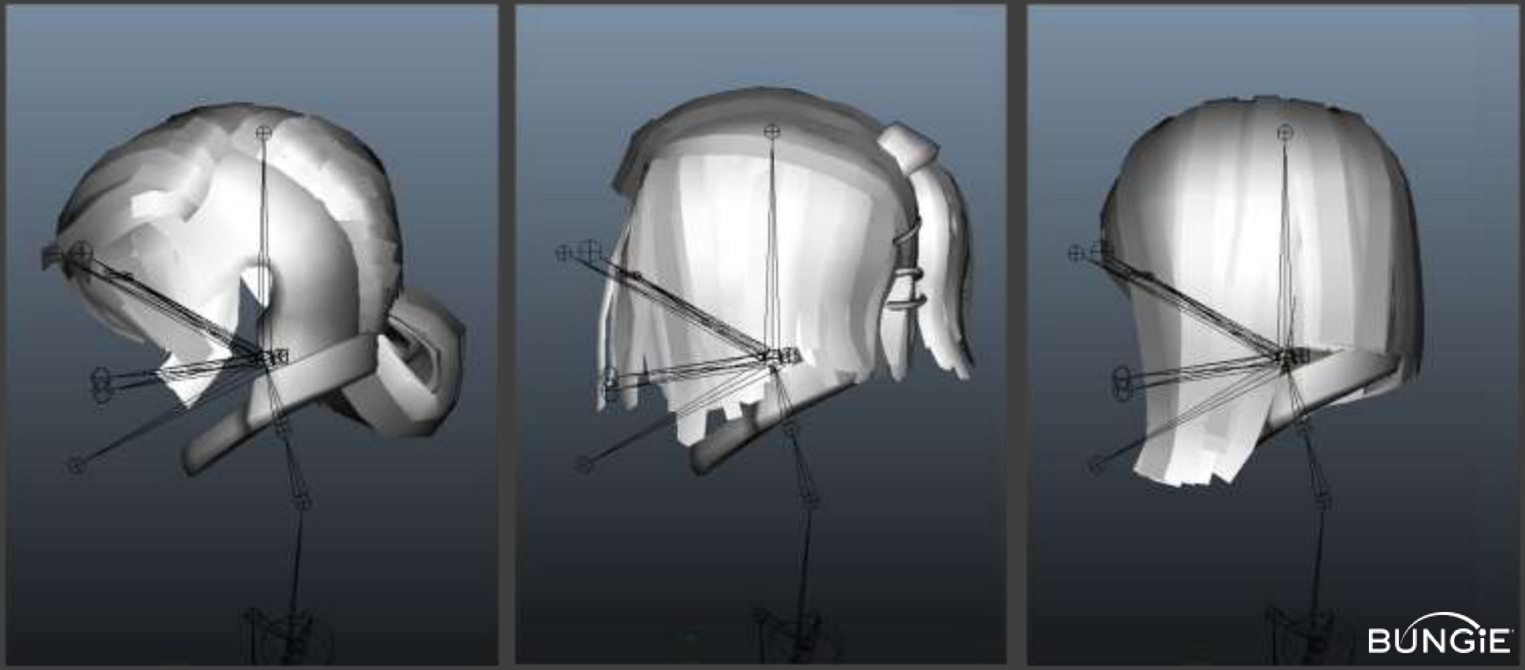






Why Use Vertex Animation?

- Player Customization (armor, weapons, heads)
  - Shared skeletons
  - Shared animation data
  - Unique silhouettes



Why Use Vertex Animation?

Reduce Joint Count

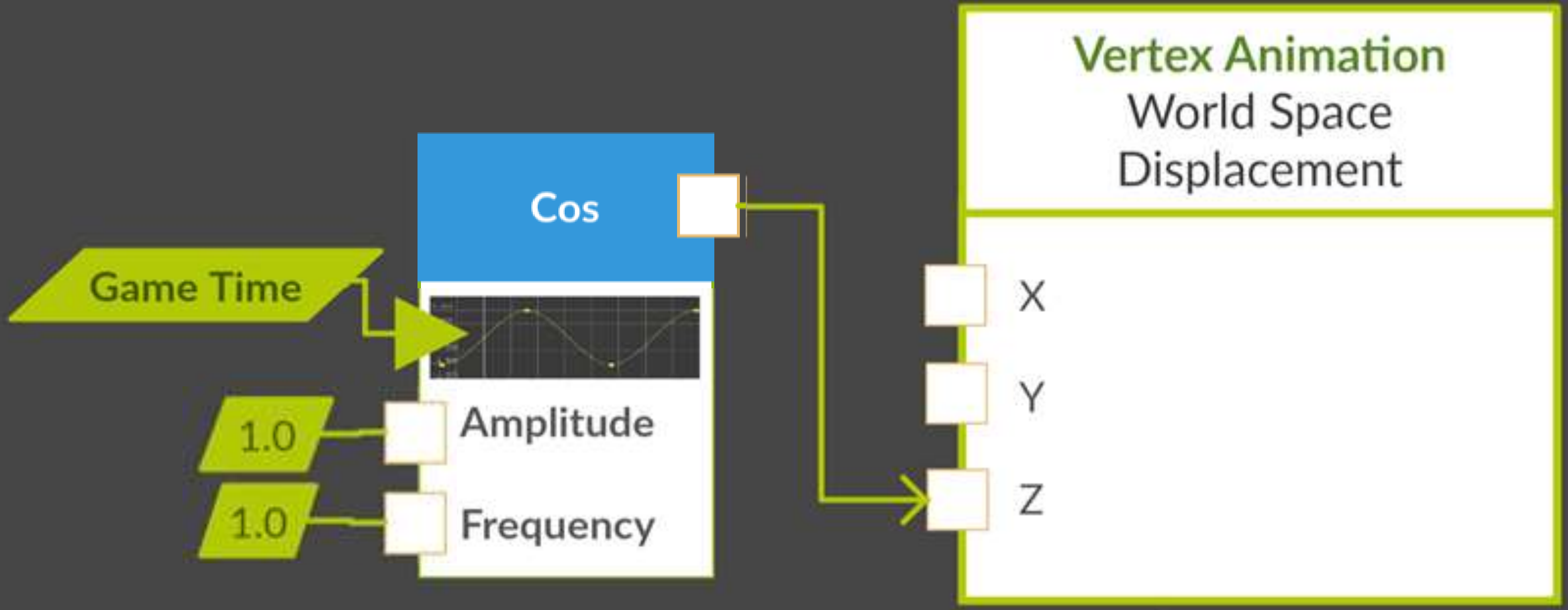
- Similar movement would require A LOT of joints

Reduce Animation Work

- Animation authored by a technical artist

Building Movement Functions

Techniques for creating animation



Game Time

1.0

1.0

Cos

Amplitude

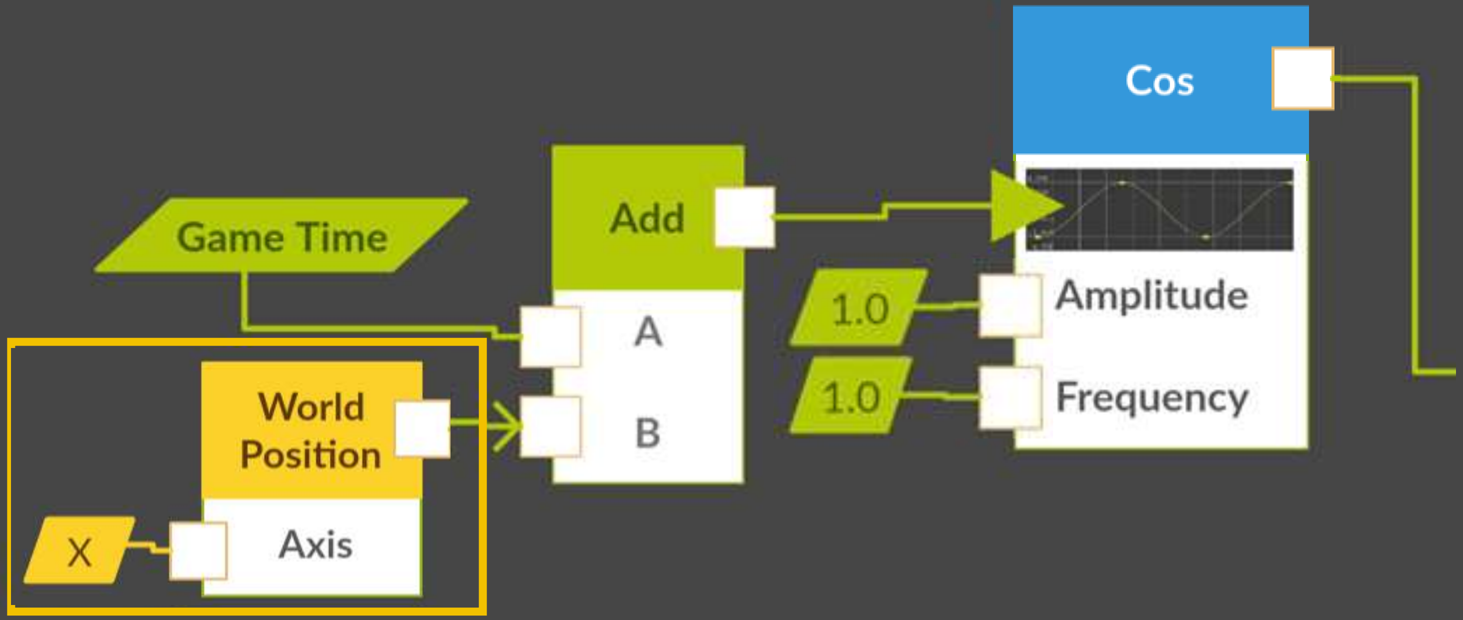
Frequency

Vertex Animation
World Space
Displacement

X

Y

Z



Game Time

World Position

Axis

X

Add

A

B

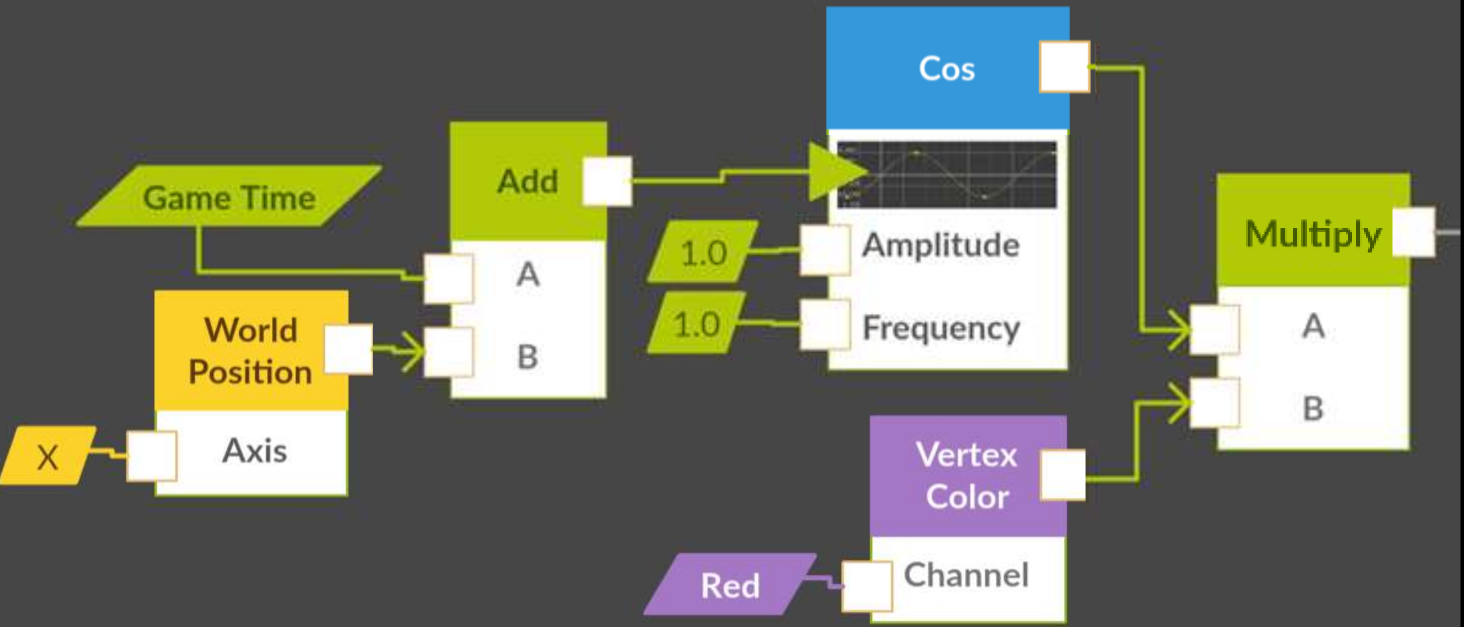
Cos

Amplitude

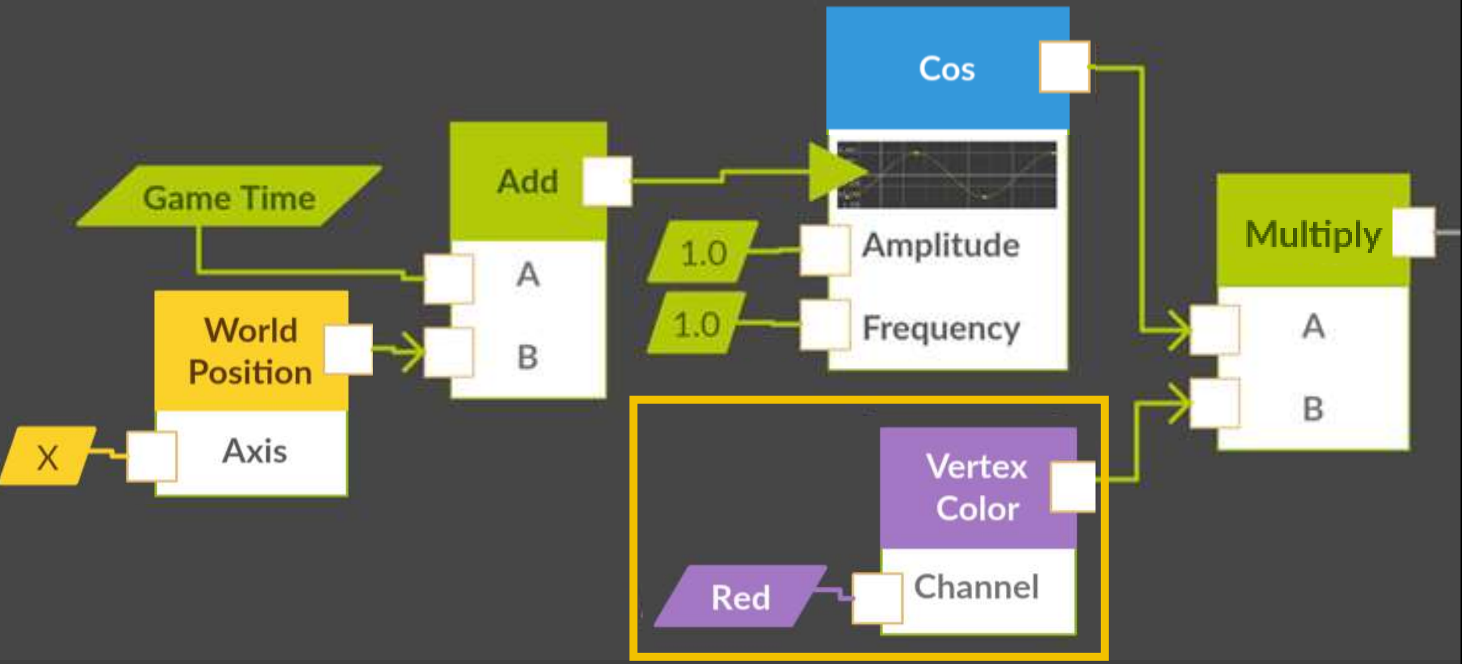
Frequency

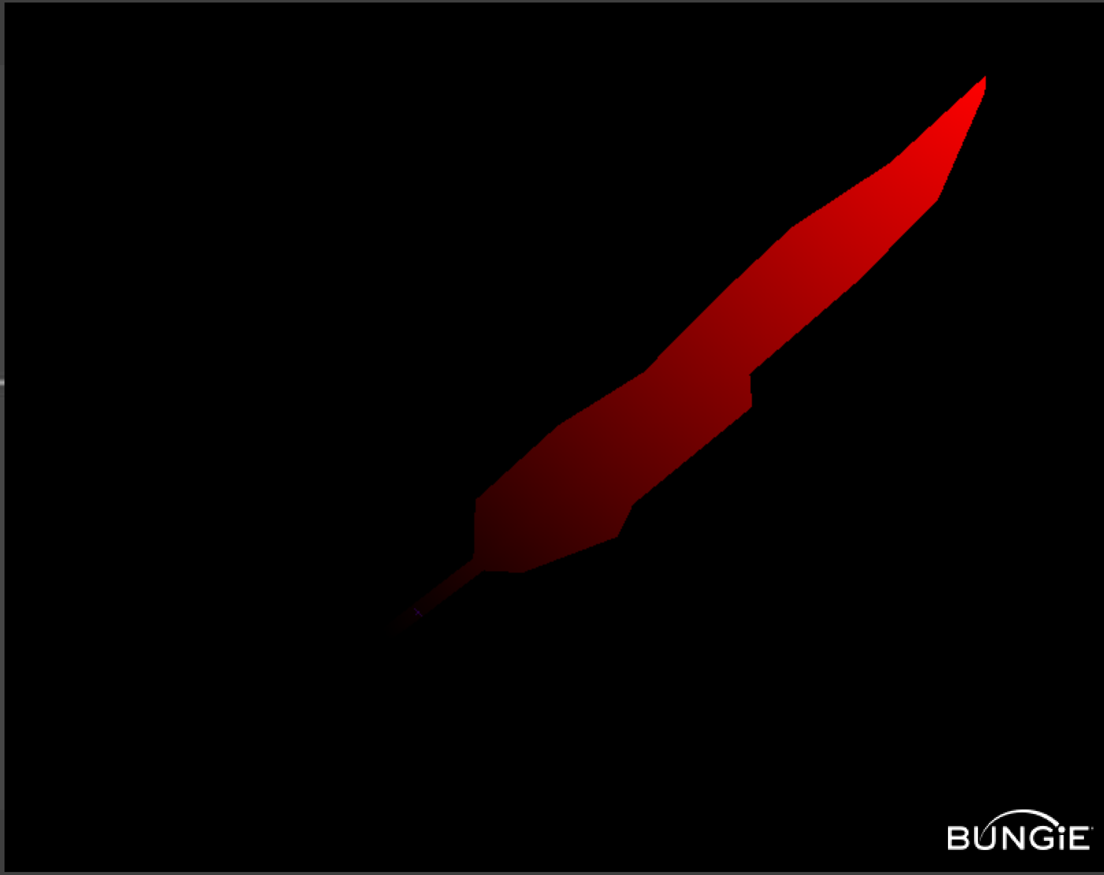
1.0

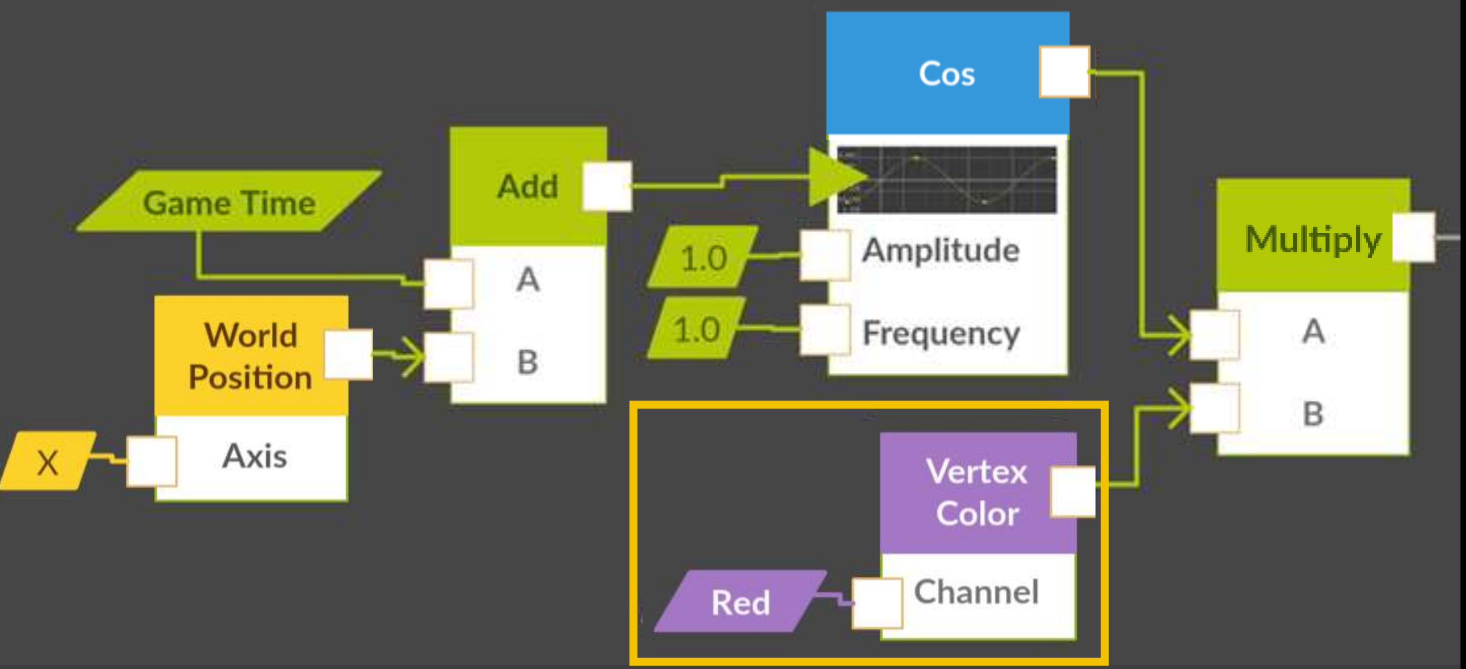
1.0


Game Time
World Position
Axis
X
Add
A
B
Cos
1.0
Amplitude
1.0
Frequency
Vertex Color
Channel
Red
Multiply
A
B

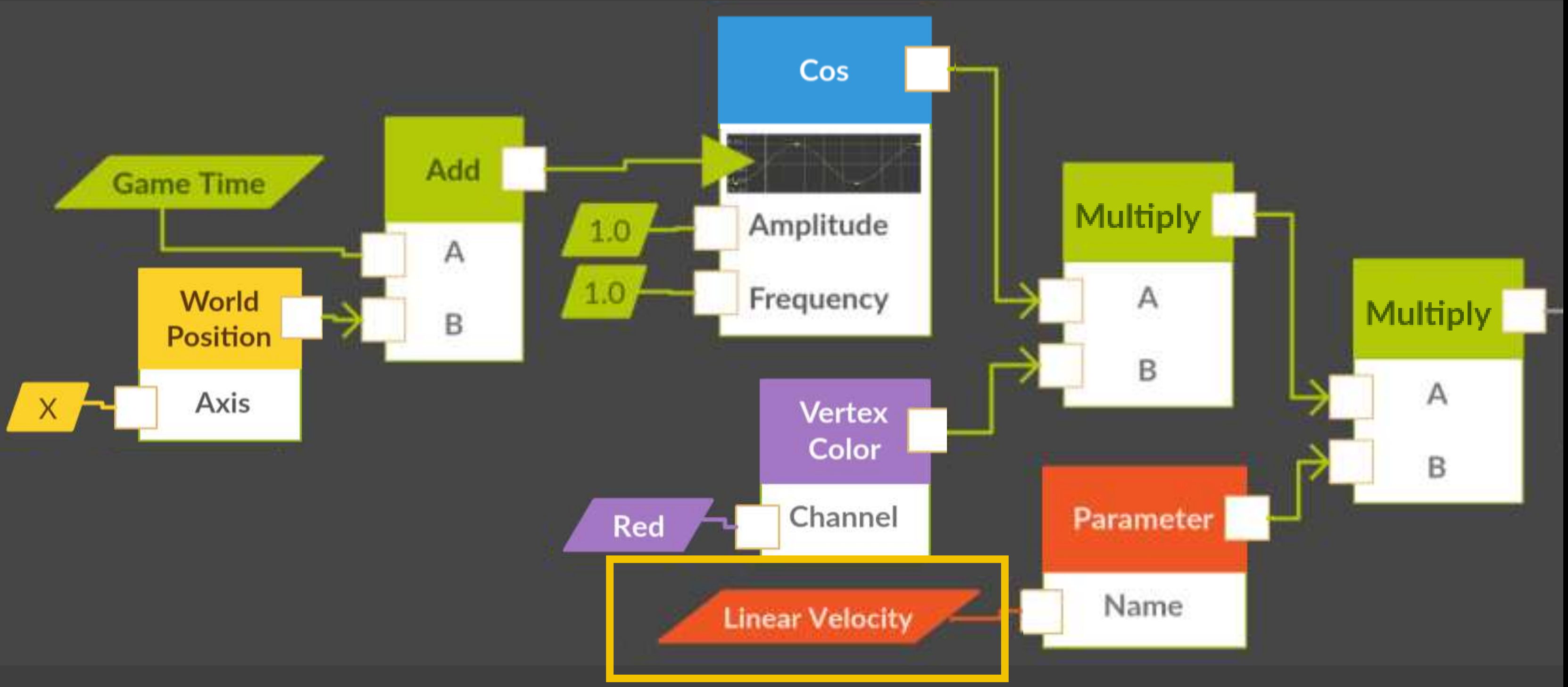
BUNGIE


Cos
Game Time
Add
A
B
1.0
Amplitude
1.0
Frequency
Multiply
A
B
World Position
Axis
X
Vertex Color
Red
Channel




Game Time
World Position
X
Axis
Add
A
B
Cos
1.0
Amplitude
1.0
Frequency
Vertex Color
Red
Channel
Multiply
A
B

BUNGiE


Game Time
Add
A
B
World Position
Axis
X
Cos
1.0
Amplitude
1.0
Frequency
Multiply
A
B
Vertex Color
Channel
Red
Linear Velocity
Parameter
Name
Multiply
A
B

BUNGiE

Add Follow-Through Motion

Original

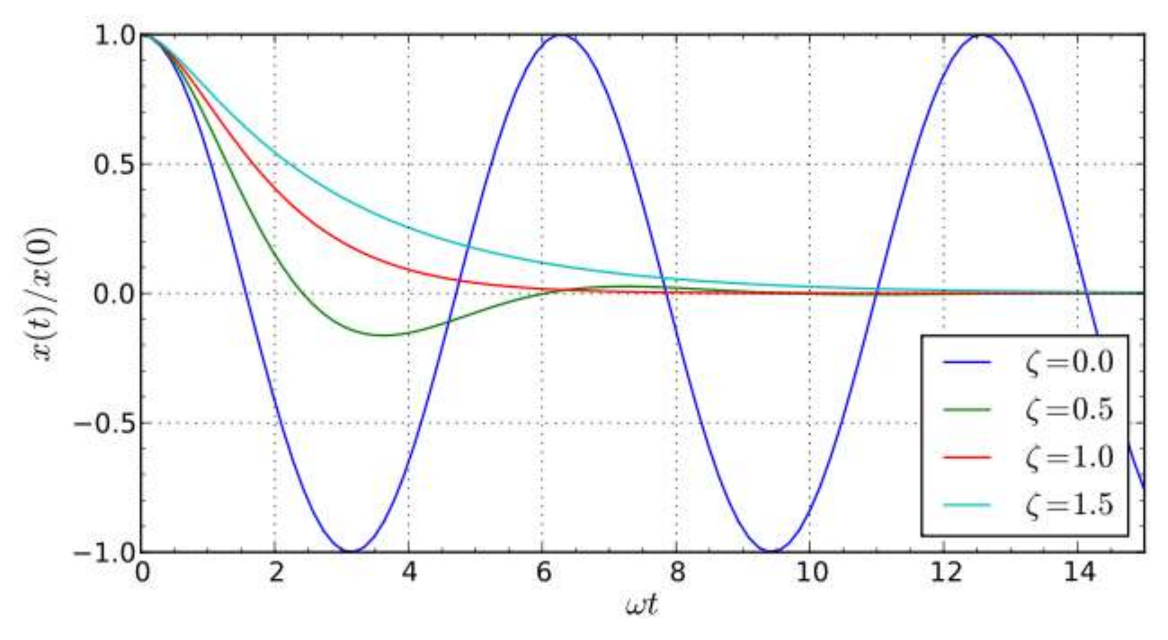
With Follow-Through

Channel Interpolation

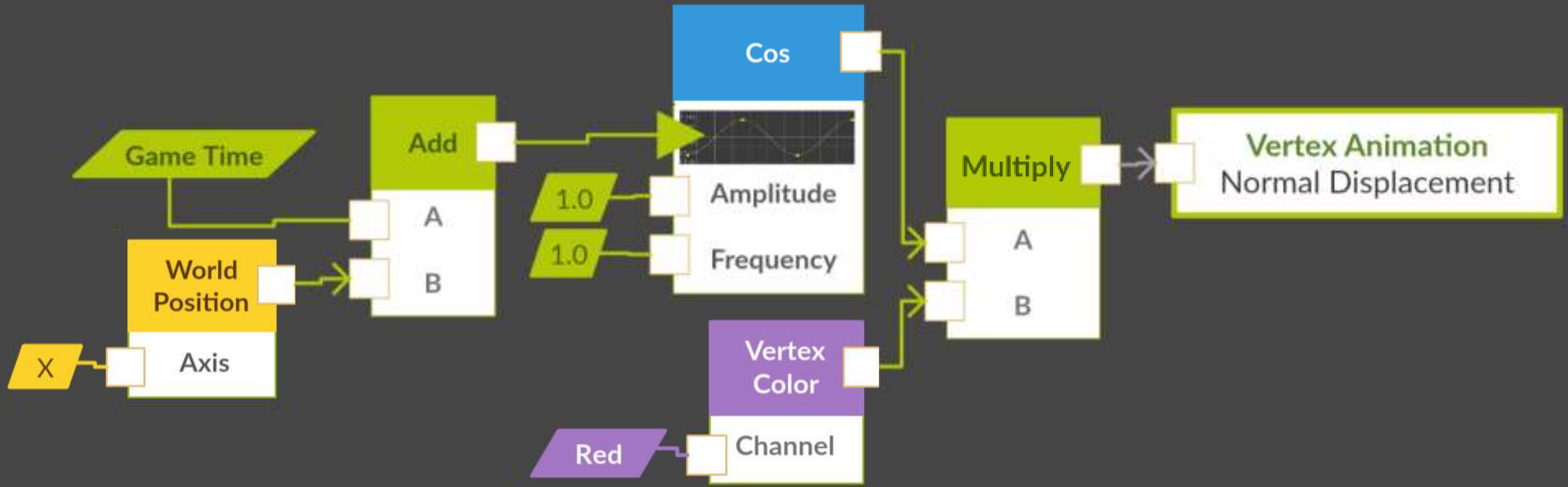
Create a new channel: Dampened_Linear_Velocity

length(float4(linear_velocity.x, linear_velocity.y, 0,0)) * exceptions

The damping ratio: damping/2*sqrt(strength)



Further Cosine Customization



Game Time

World Position

X

Axis

Add

A

B

1.0

1.0

Cos

Amplitude

Frequency

Vertex Color

Red

Channel

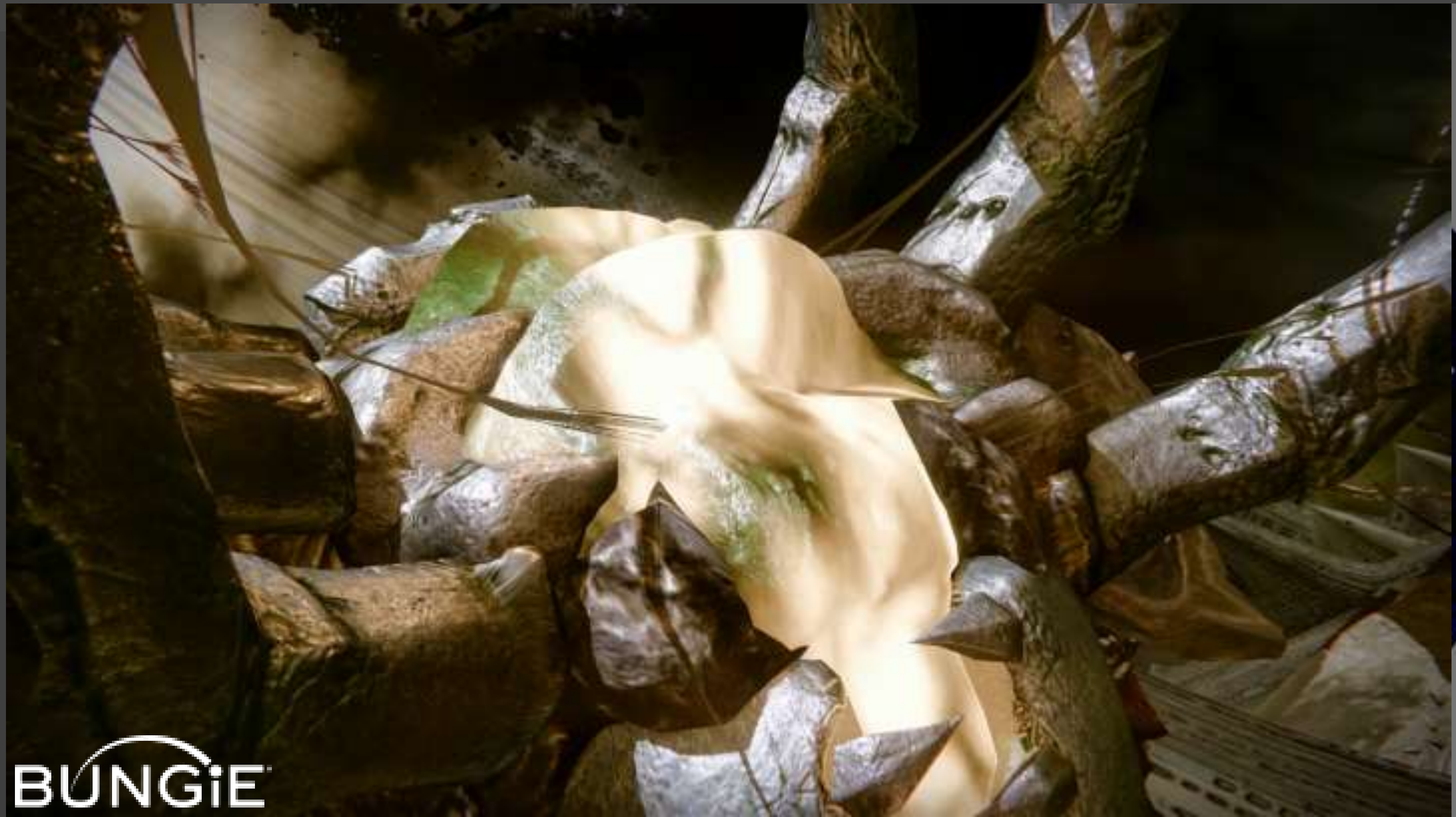
Multiply

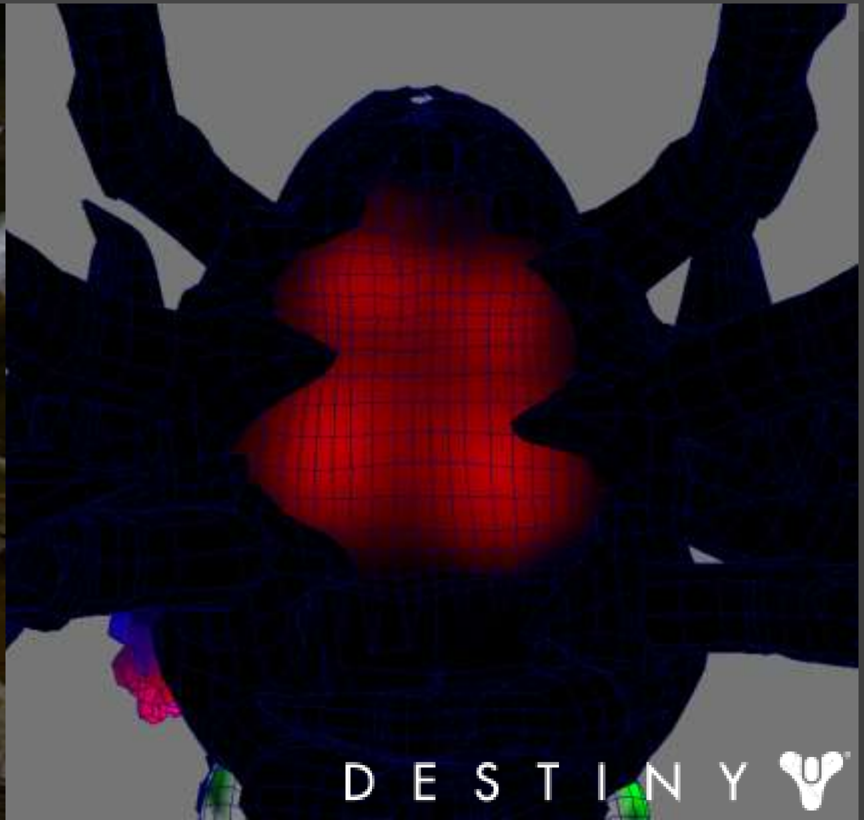
A

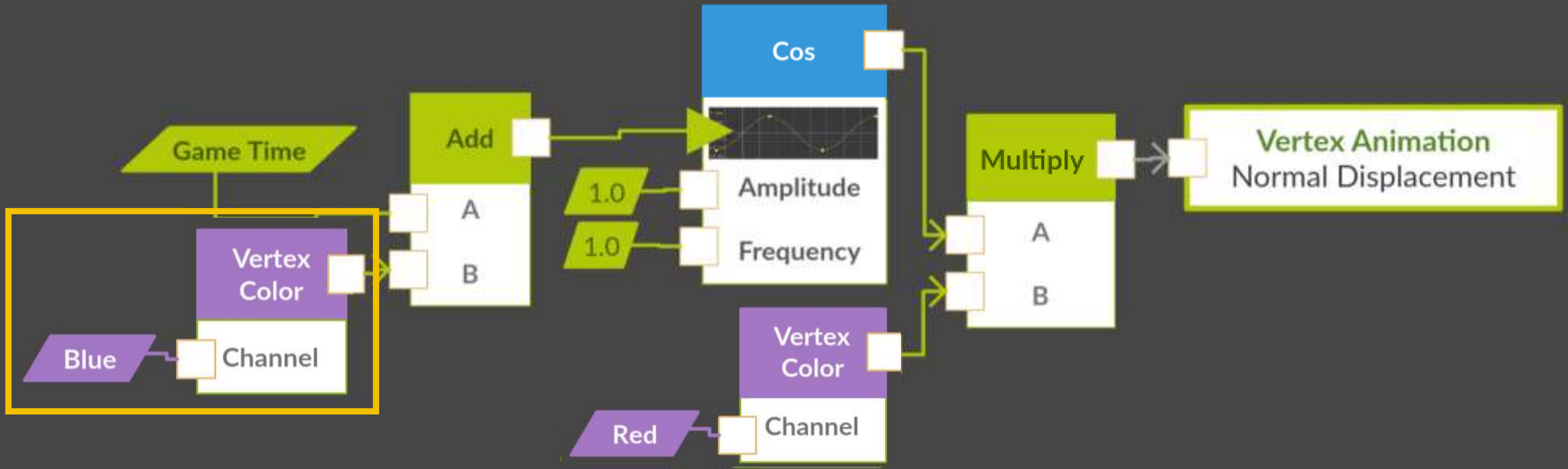
B

Vertex Animation

Normal Displacement







Game Time

Add

A

B

Vertex Color

Blue

Channel

Cos

1.0

Amplitude

1.0

Frequency

Vertex Color

Red

Channel

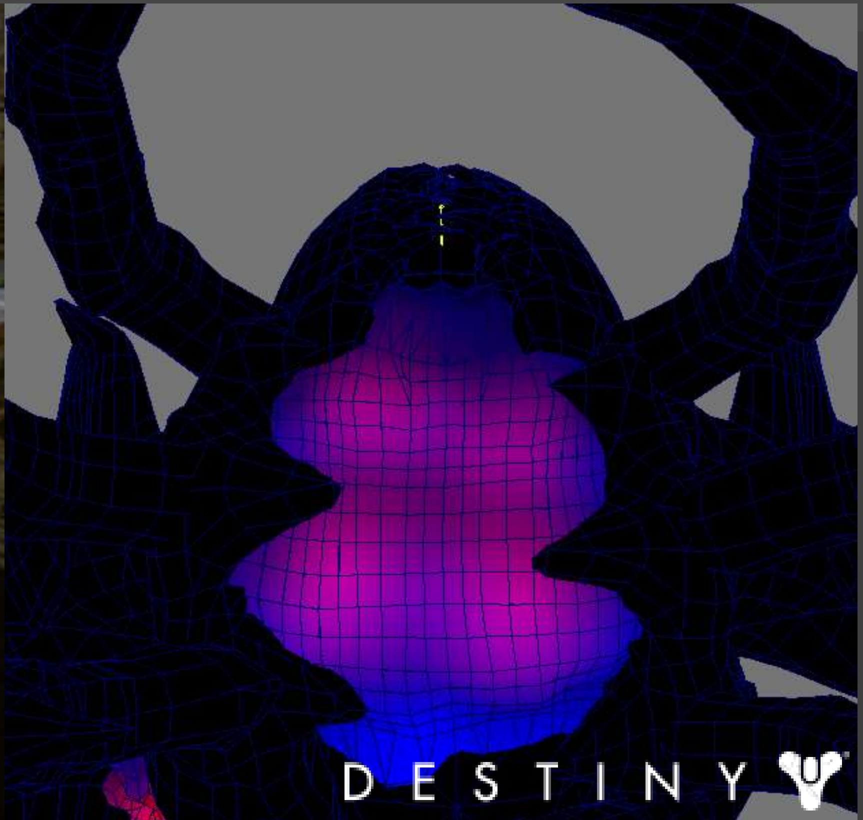
Multiply

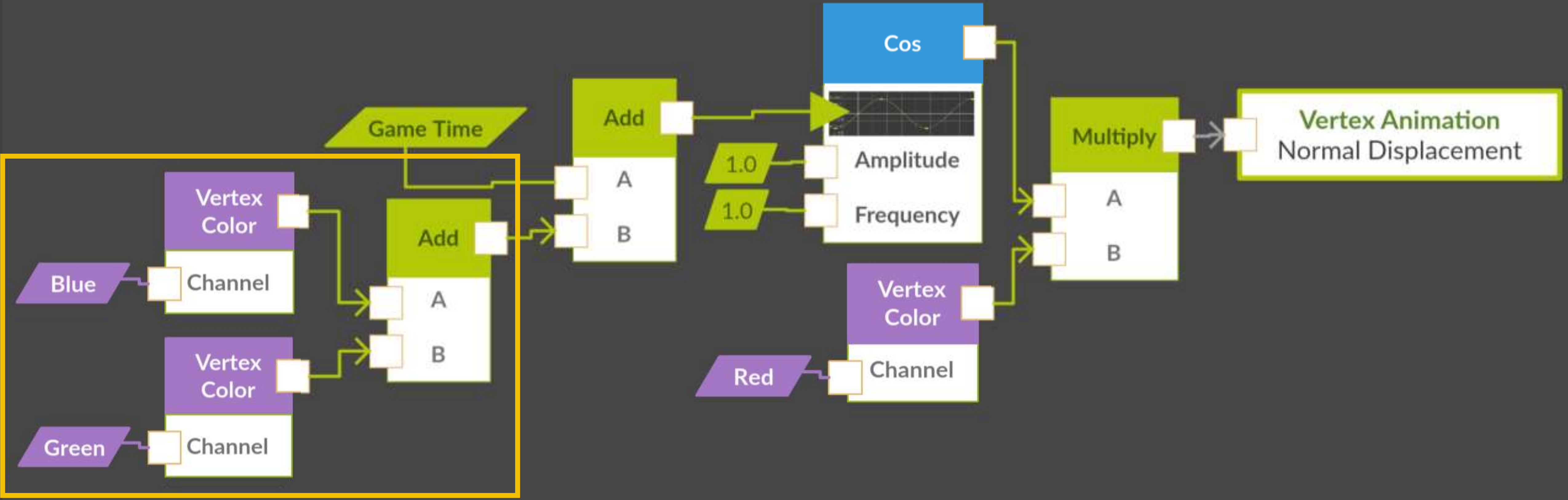
A

B

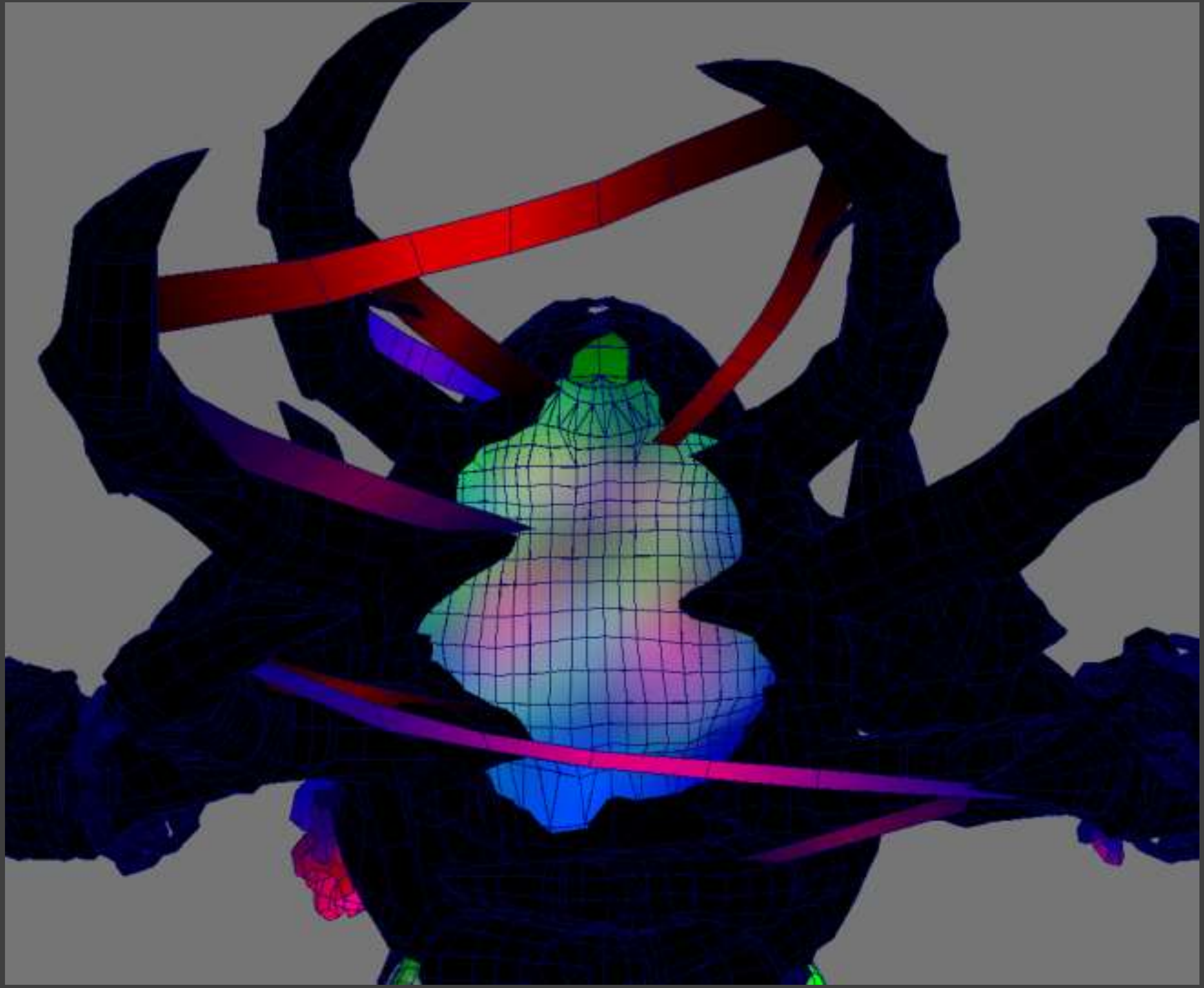
Vertex Animation

Normal Displacement



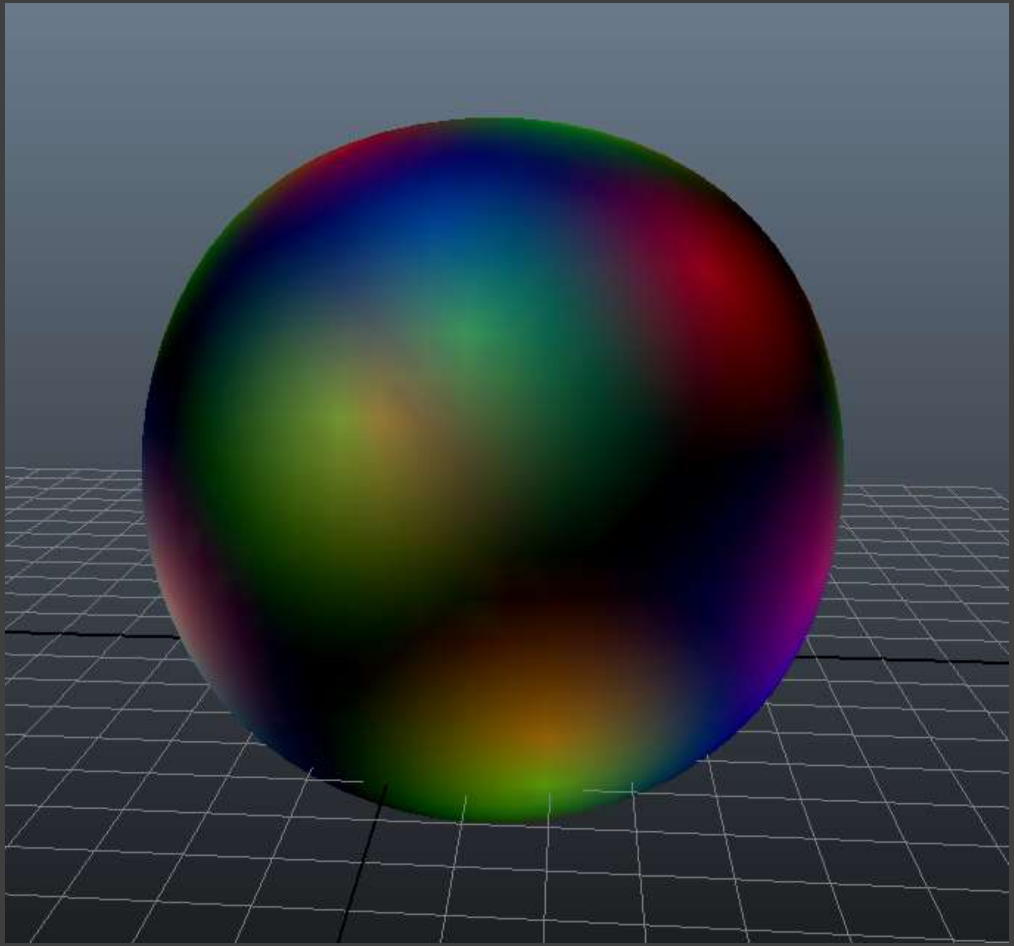

Game Time
Vertex Color
Blue
Channel
Vertex Color
Green
Channel
Add
A
B
Add
A
B
Cos
1.0
Amplitude
1.0
Frequency
Vertex Color
Red
Channel
Multiply
A
B
Vertex Animation
Normal Displacement

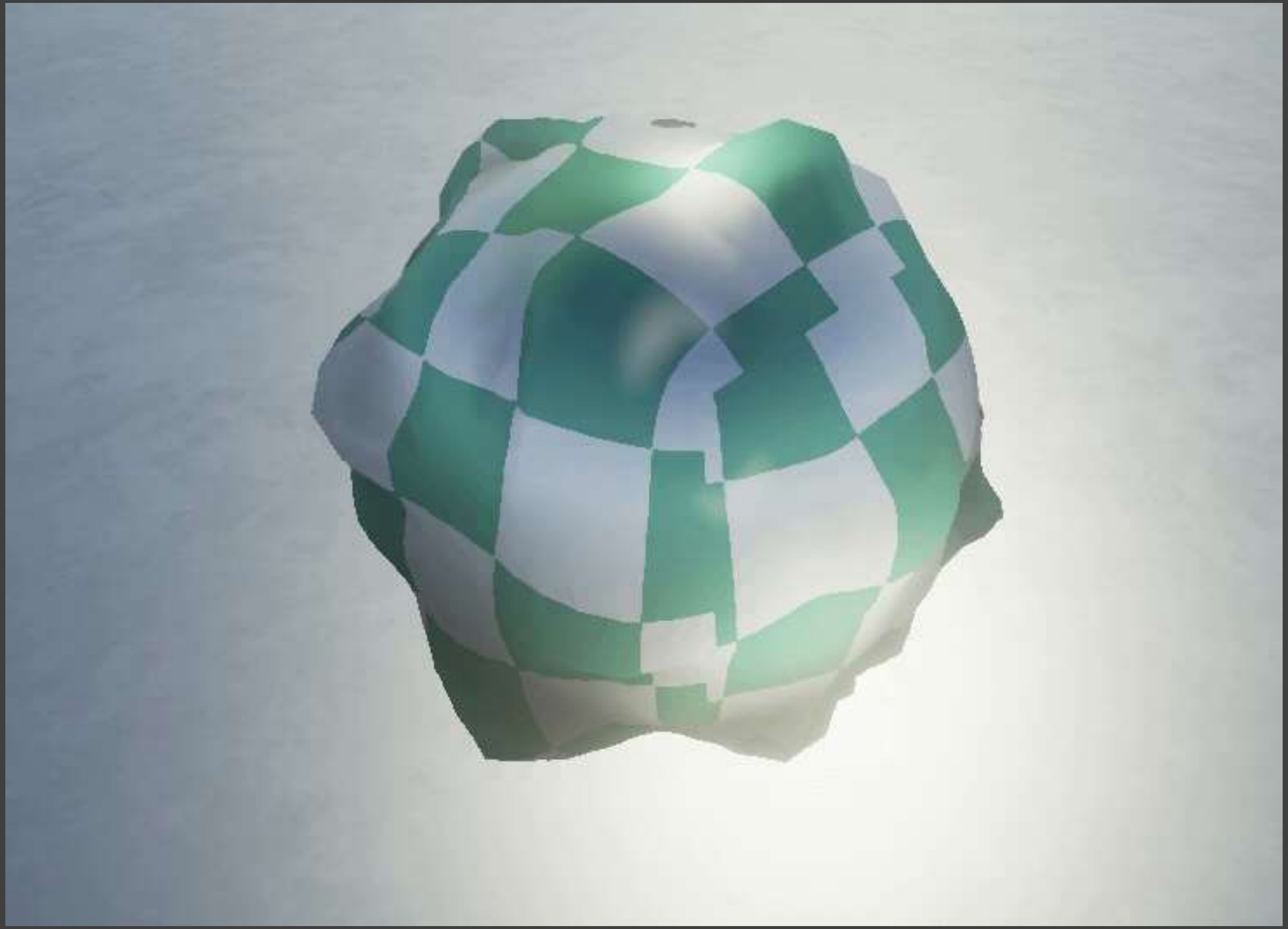
BUNGIE



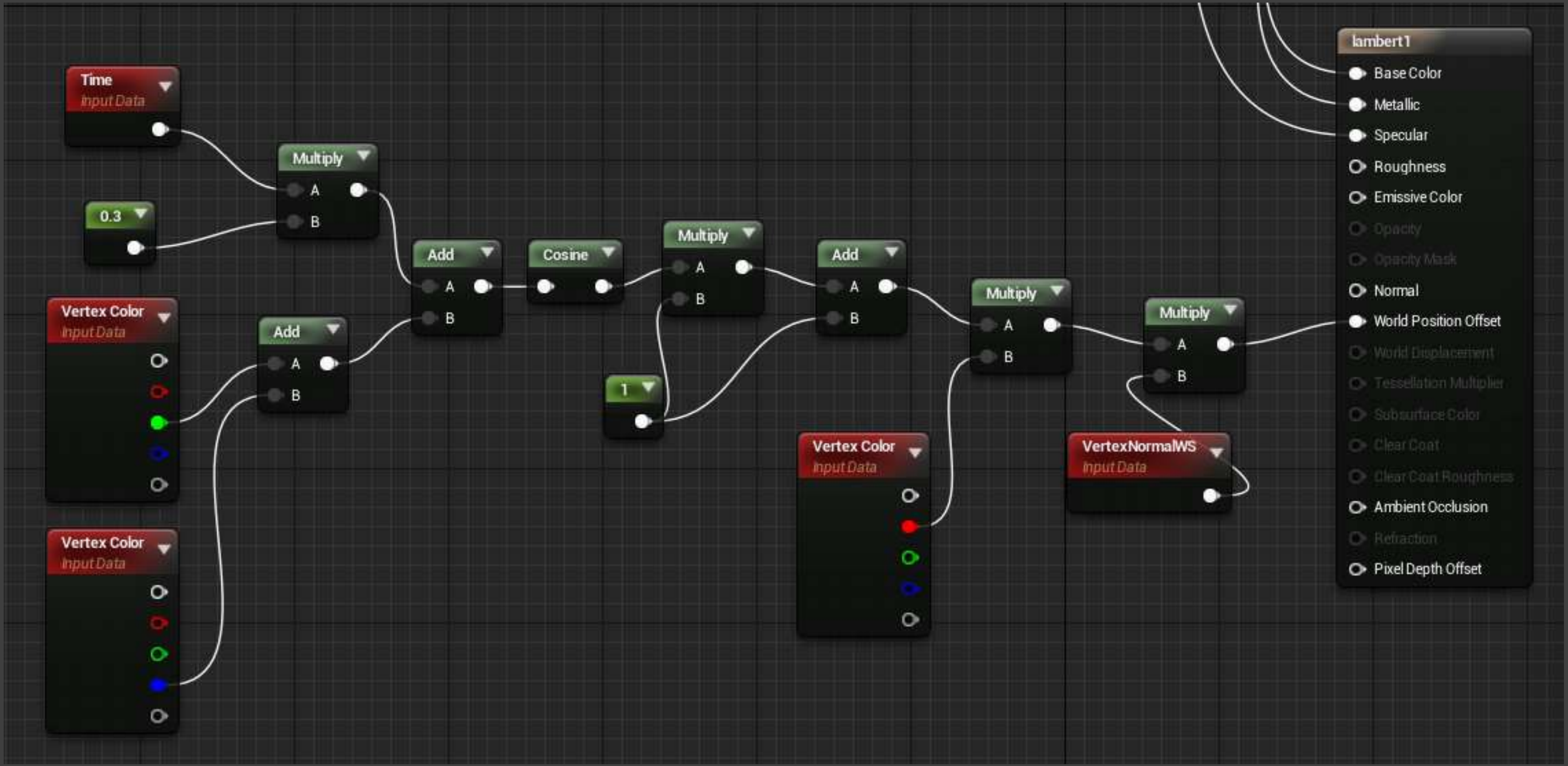
BUNGiE

Unreal Engine Example

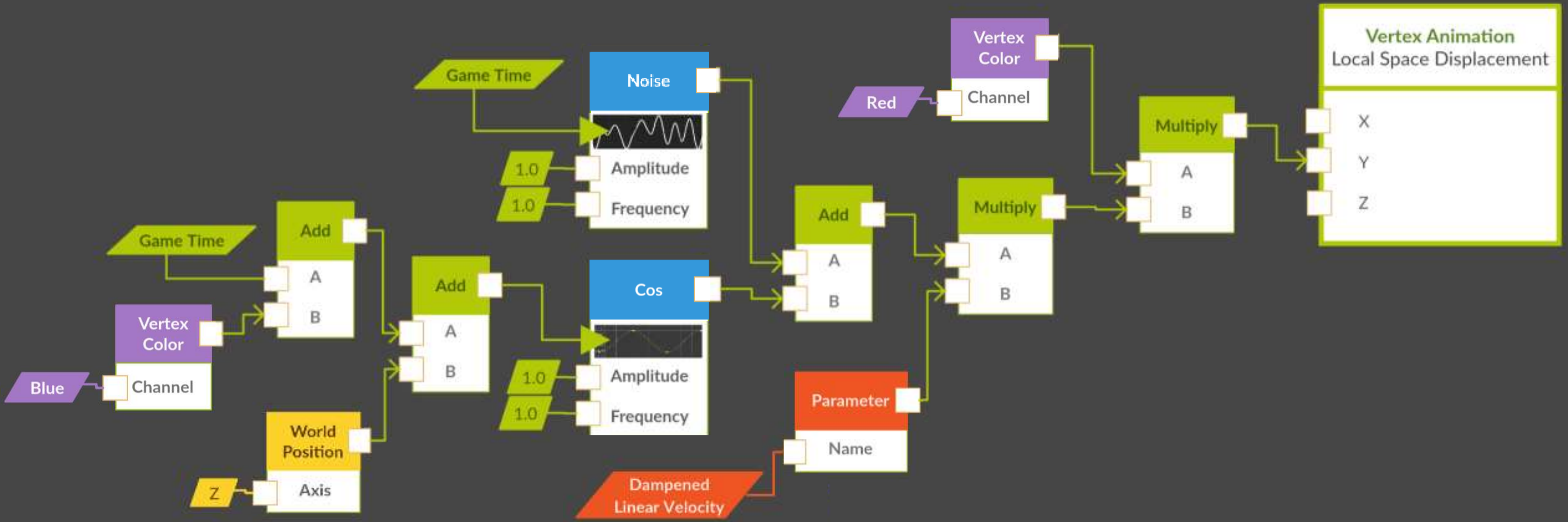




Unreal Engine Example

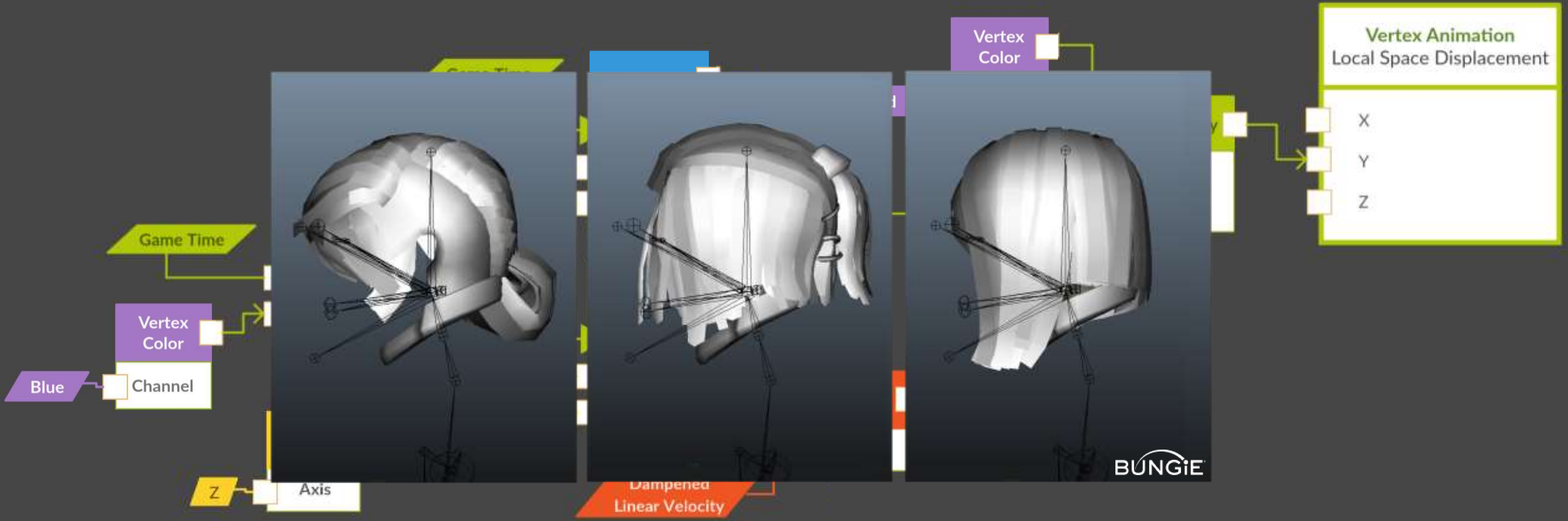


Player Hair

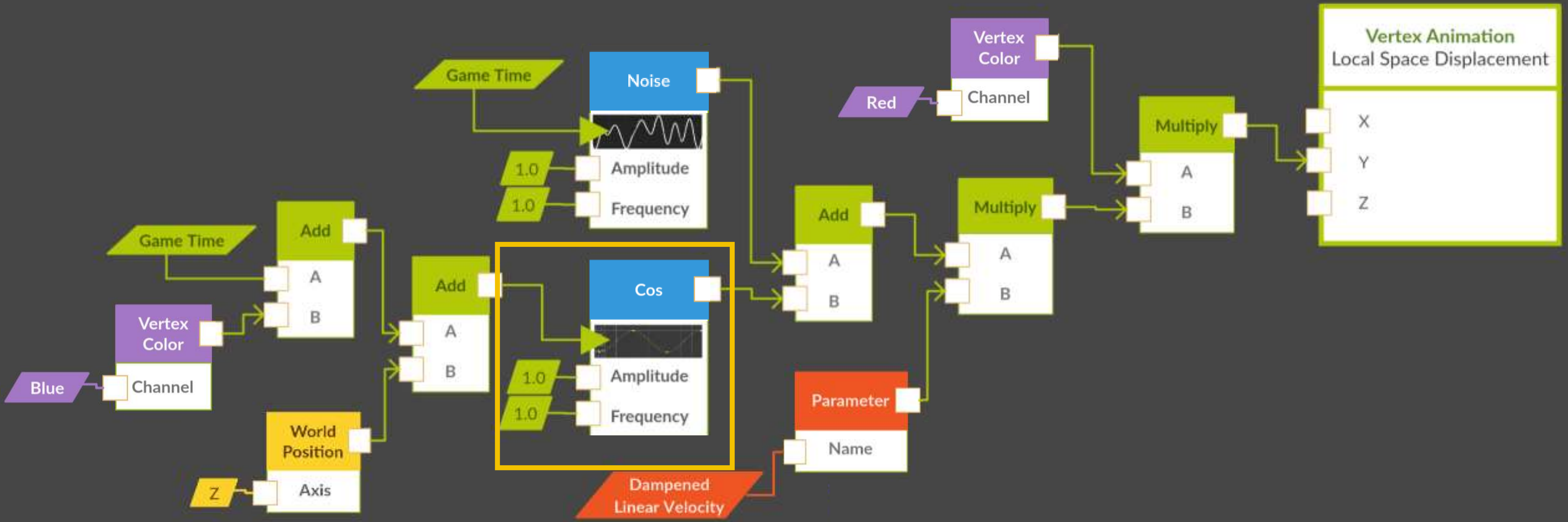


Game Time
Noise
1.0
Amplitude
1.0
Frequency
Vertex Color
Red
Channel
Vertex Animation
Local Space Displacement
X
Y
Z
Multiply
A
B
Game Time
Add
A
B
Add
A
B
Add
A
B
Multiply
A
B
Vertex Color
Blue
Channel
Cos
1.0
Amplitude
1.0
Frequency
Parameter
Name
World Position
Z
Axis
Dampened Linear Velocity

Player Hair

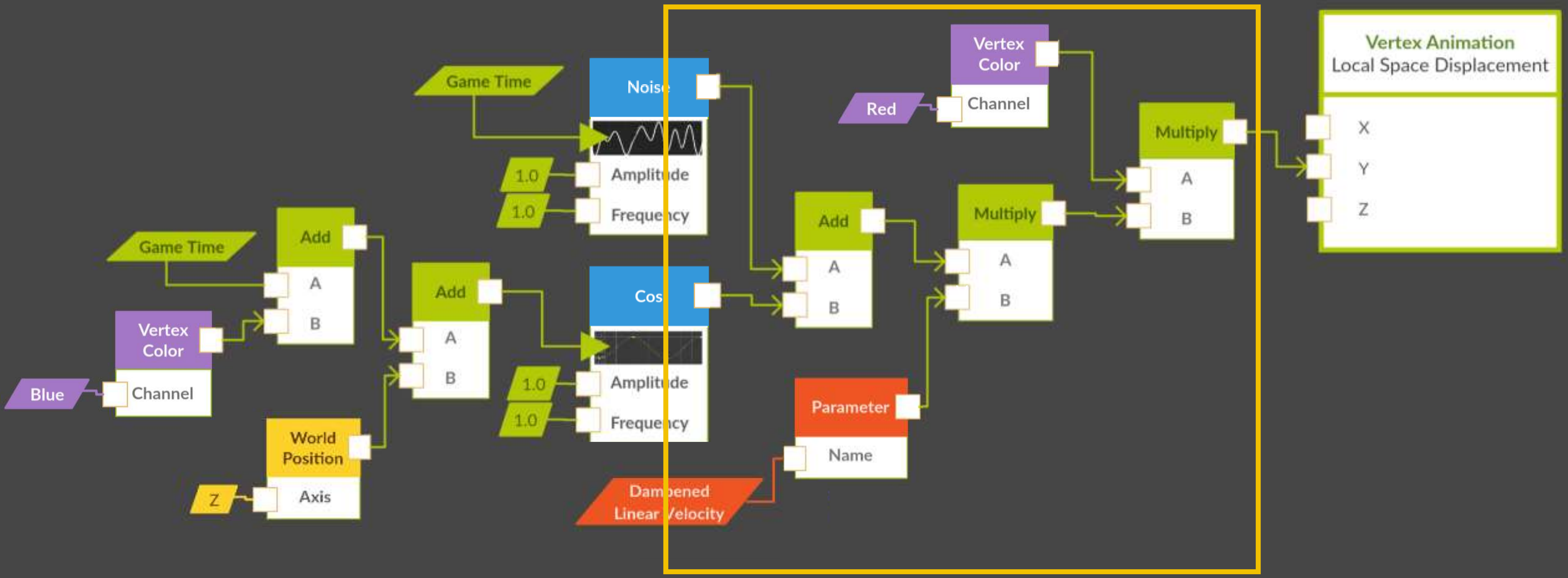

Vertex Color
Vertex Animation
Local Space Displacement
X
Y
Z
Game Time
Vertex Color
Blue
Channel
Z
Axis
Dampened
Linear Velocity

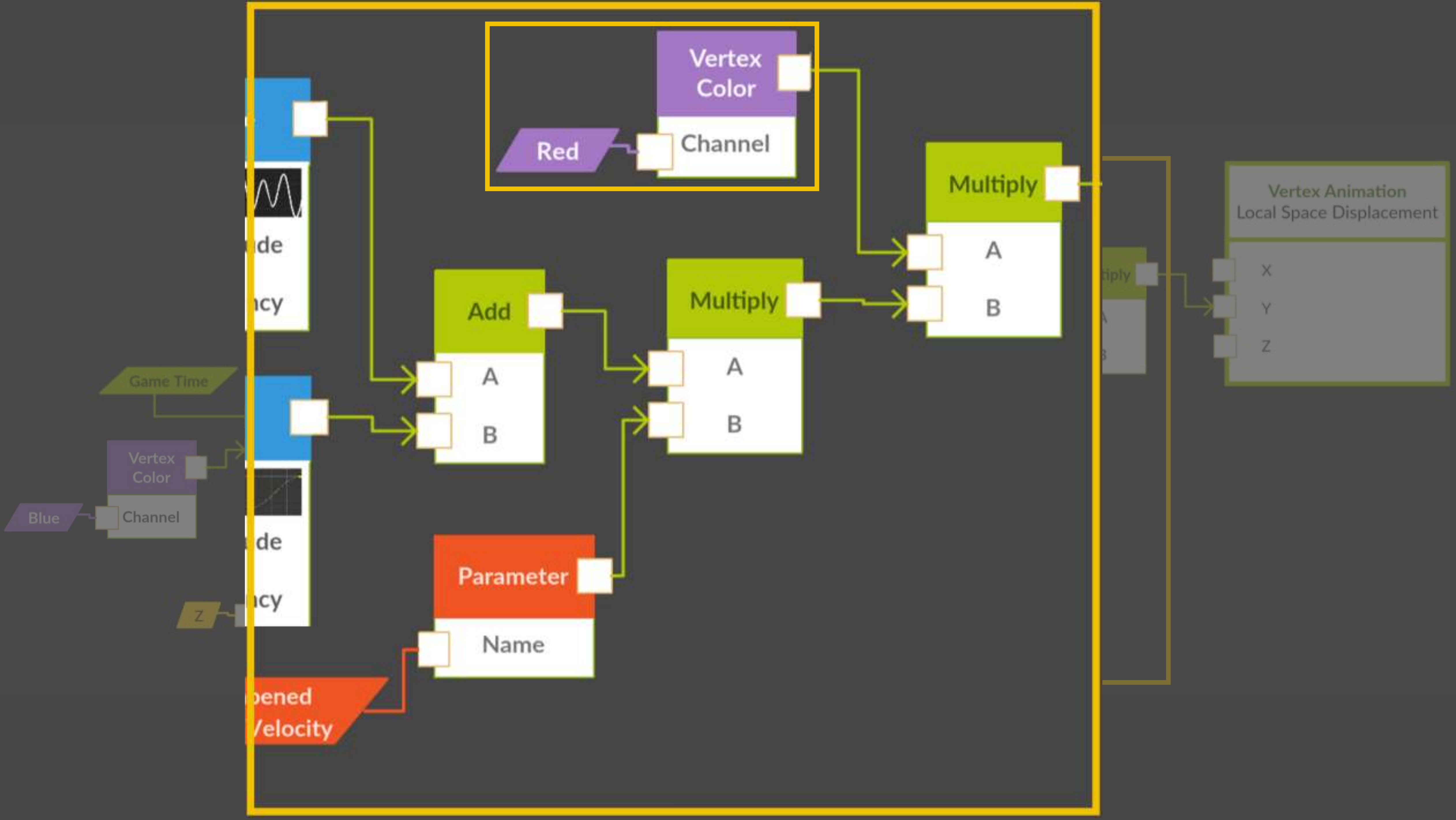
Player Hair



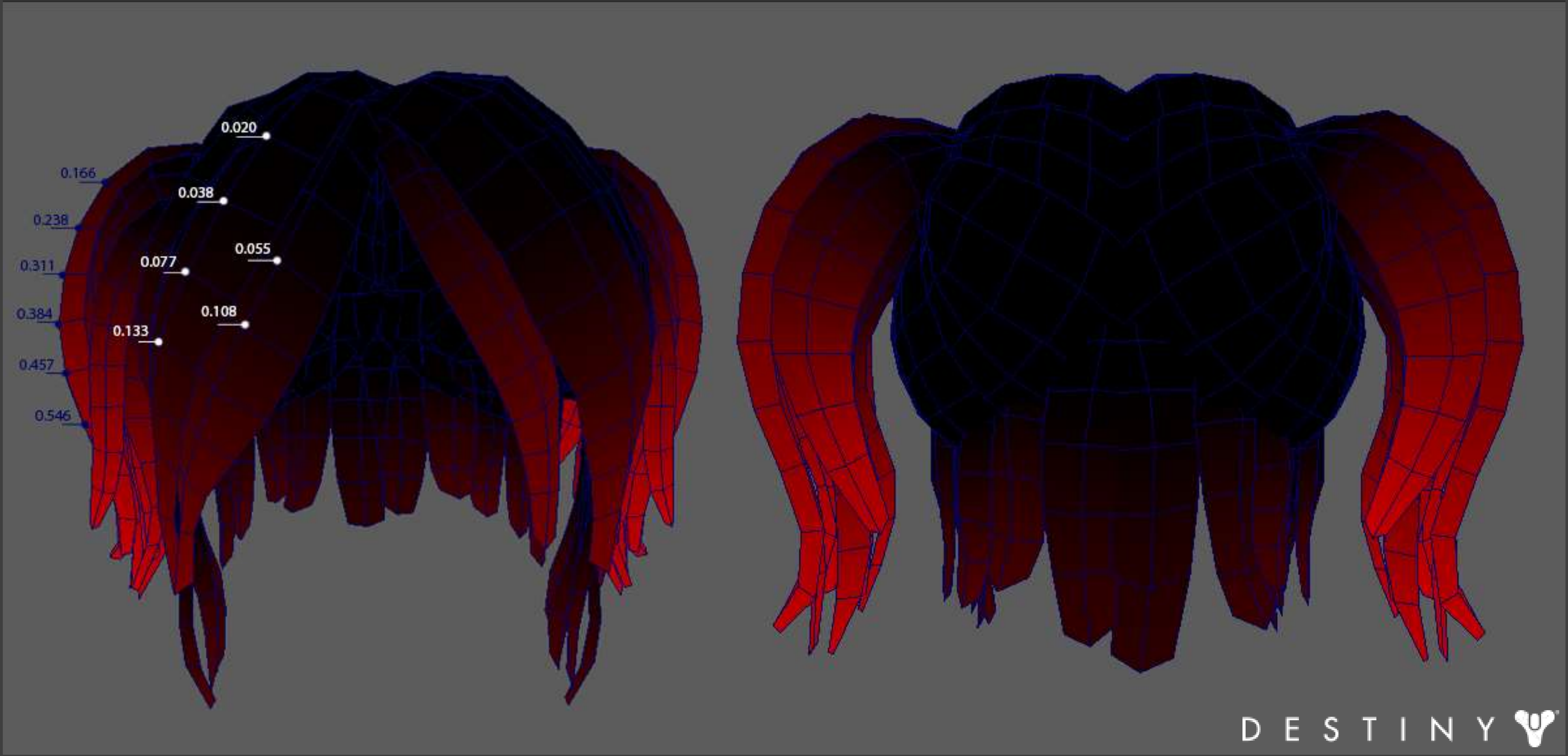
Game Time
Noise
1.0
Amplitude
1.0
Frequency
Vertex
Color
Red
Channel
Vertex Animation
Local Space Displacement
X
Y
Z
Multiply
A
B
Game Time
Add
A
B
Add
A
B
Multiply
A
B
Vertex
Color
Blue
Channel
Add
A
B
Cos
1.0
Amplitude
1.0
Frequency
Parameter
Name
World
Position
Z
Axis
Dampened
Linear Velocity

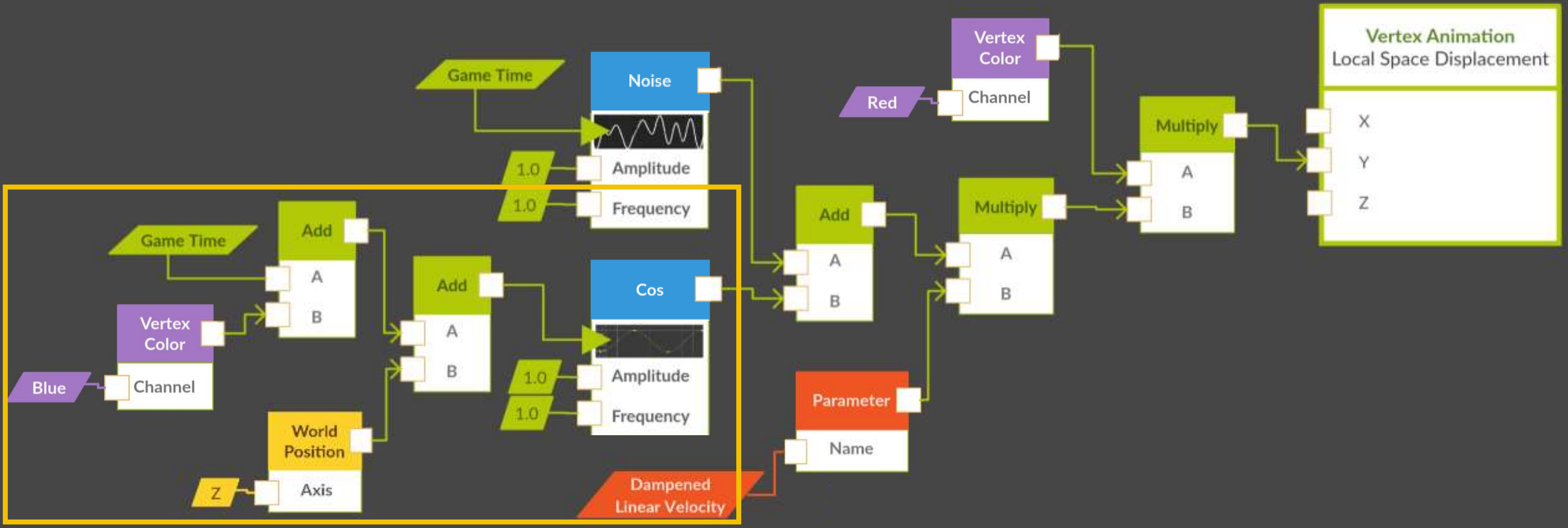
Player Hair

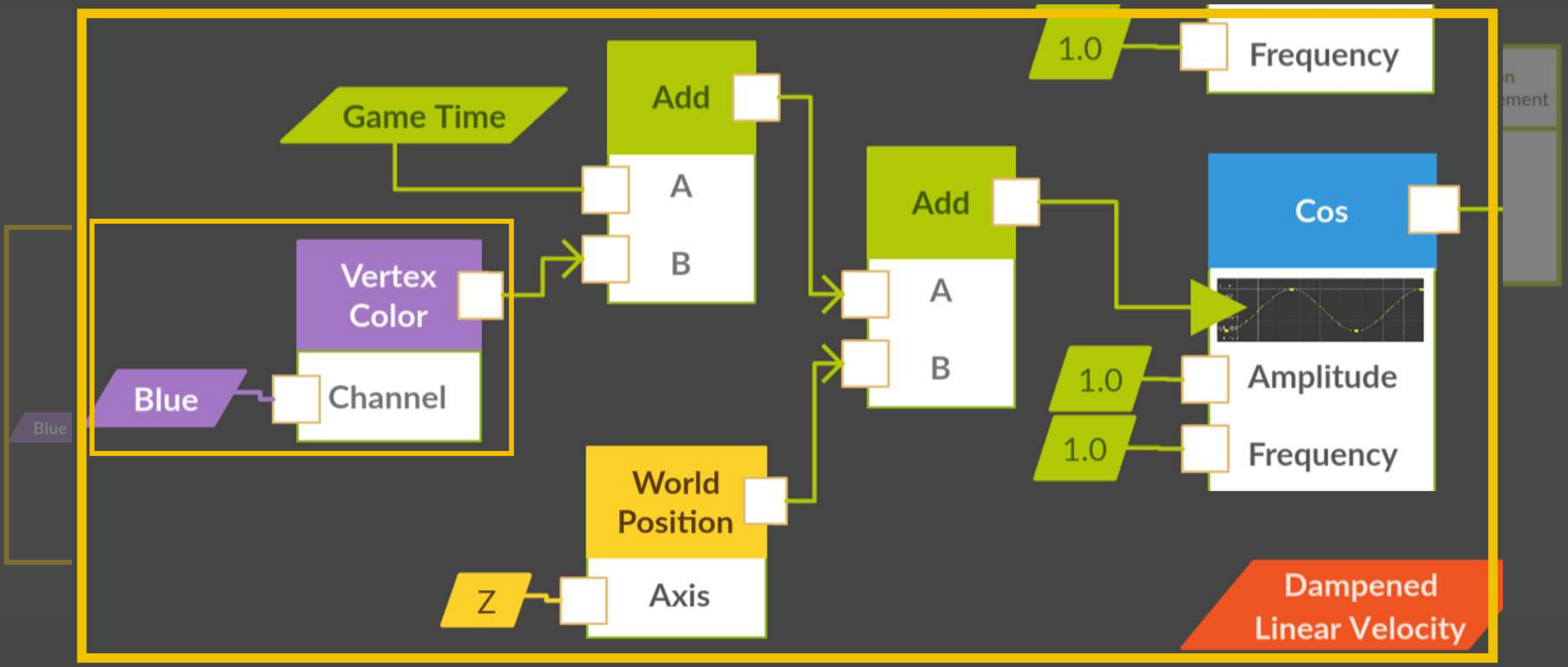

Game Time
Noise
1.0
Amplitude
1.0
Frequency
Game Time
Add
A
B
Vertex Color
Blue
Channel
Add
A
B
World Position
Z
Axis
Cos
1.0
Amplitude
1.0
Frequency
Vertex Color
Red
Channel
Add
A
B
Multiply
A
B
Parameter
Name
Dampened Linear Velocity
Multiply
A
B
Vertex Animation
Local Space Displacement
X
Y
Z


Vertex Color
Red
Channel
Multiply
A
B
ude
ncy
Add
A
B
Multiply
A
B
Parameter
Name
pened Velocity
ude
ncy
Game Time
Vertex Color
Blue
Channel
Z
tiply
A
B
Vertex Animation Local Space Displacement
X
Y
Z

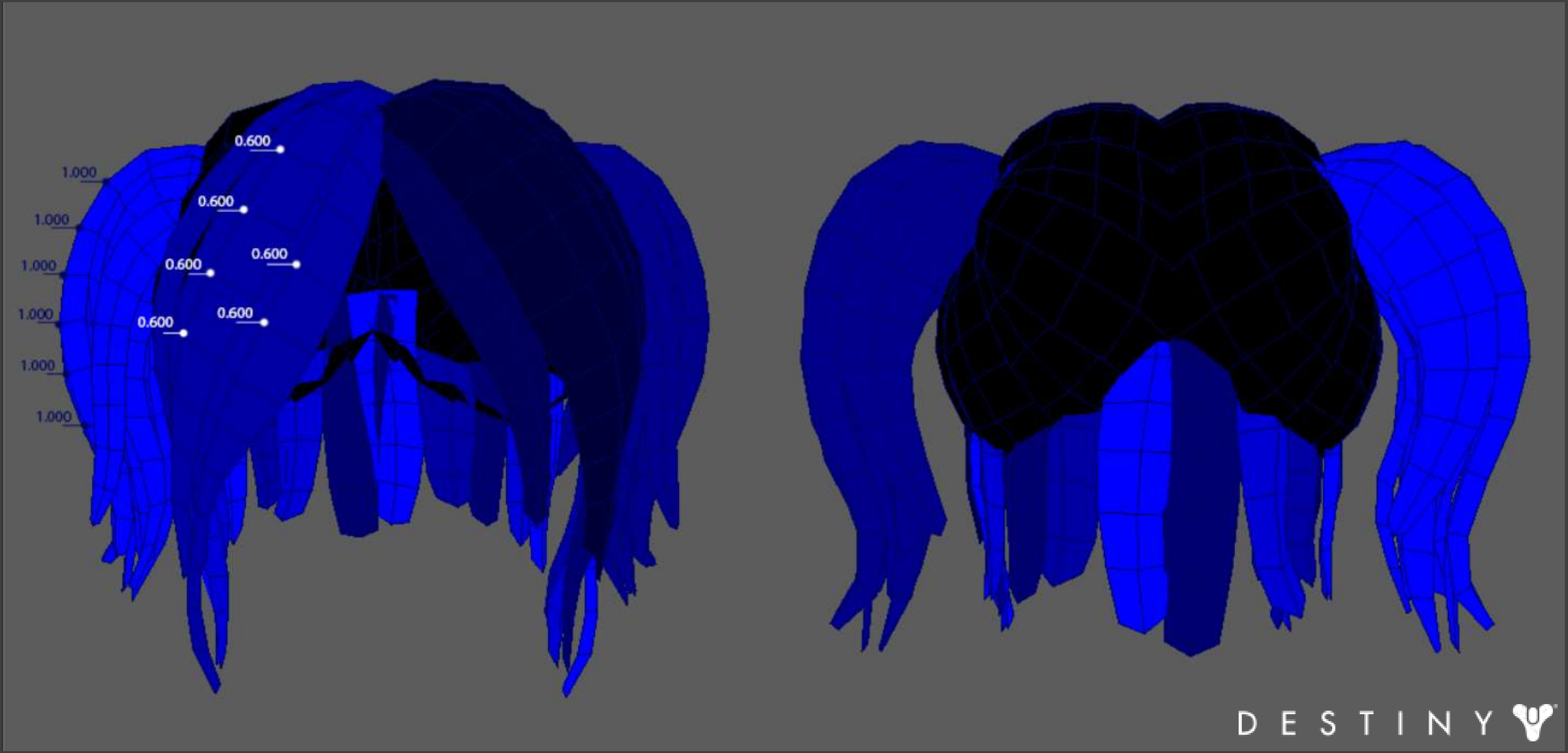
Red channel = strength of movement



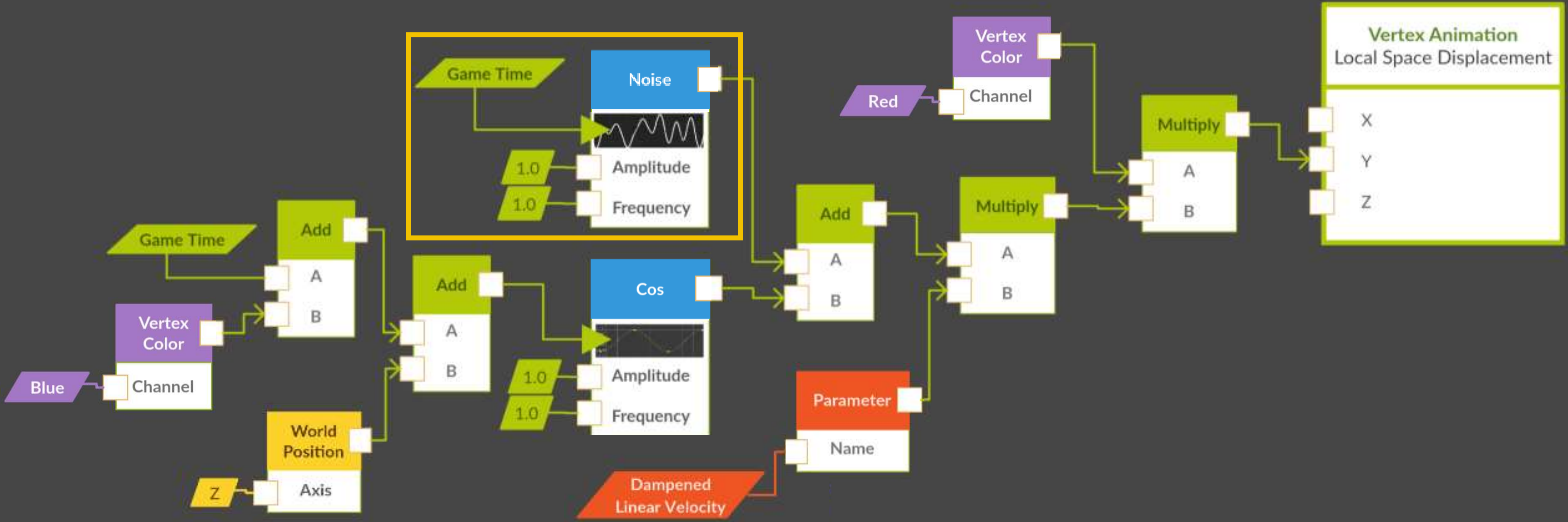

Game Time
Noise
1.0
Amplitude
1.0
Frequency
Vertex Color
Red
Channel
Vertex Animation
Local Space Displacement
X
Y
Z
Multiply
A
B
Add
A
B
Multiply
A
B
Game Time
Add
A
B
Vertex Color
Blue
Channel
Add
A
B
Cos
1.0
Amplitude
1.0
Frequency
World Position
Z
Axis
Parameter
Name
Dampened Linear Velocity

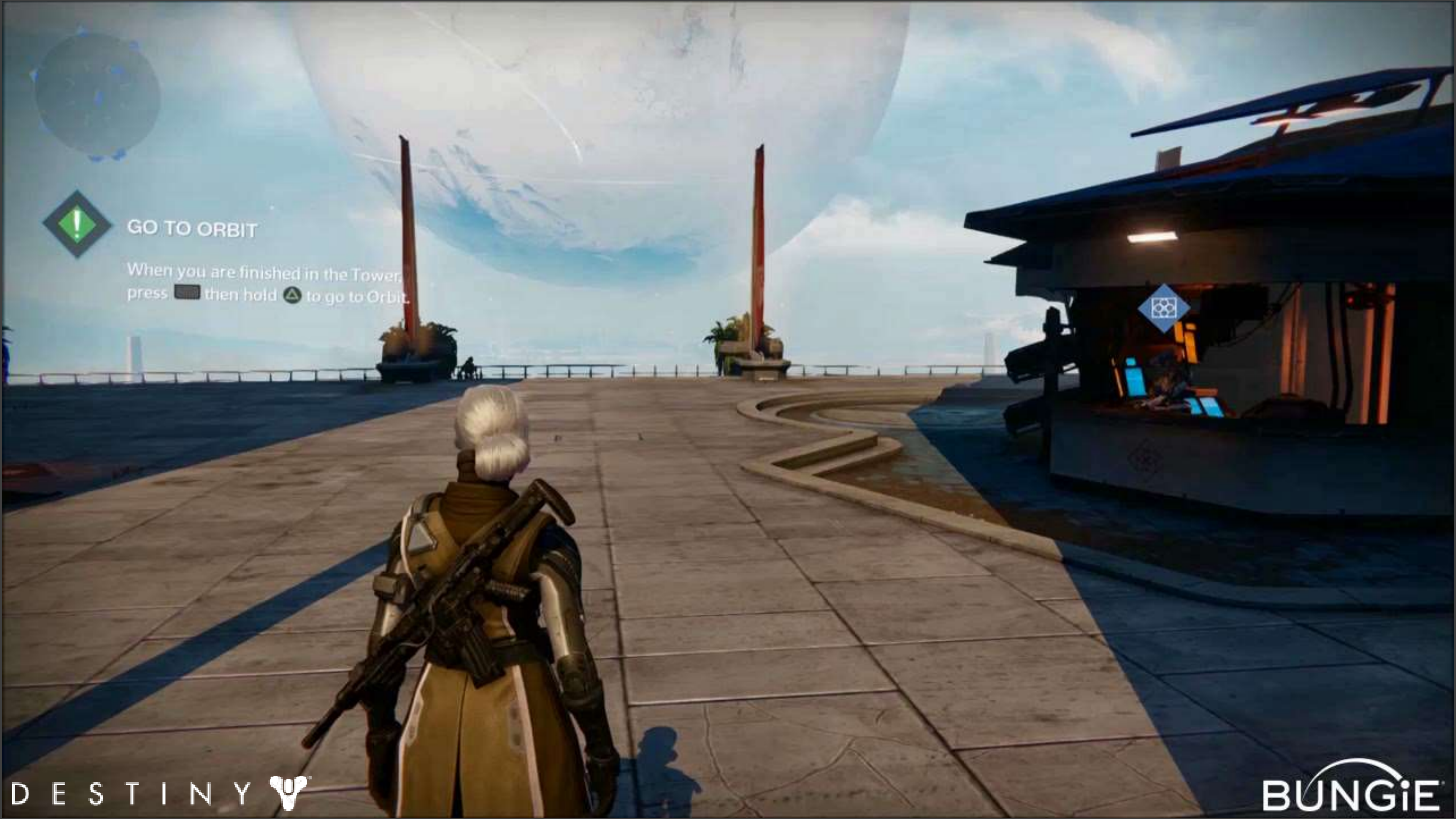

1.0
Frequency
Game Time
Add
A
B
Add
A
B
Cos
Vertex Color
Blue
Channel
1.0
Amplitude
1.0
Frequency
World Position
Z
Axis
Dampened Linear Velocity
Blue

Blue channel = timing of movement



DESTINY

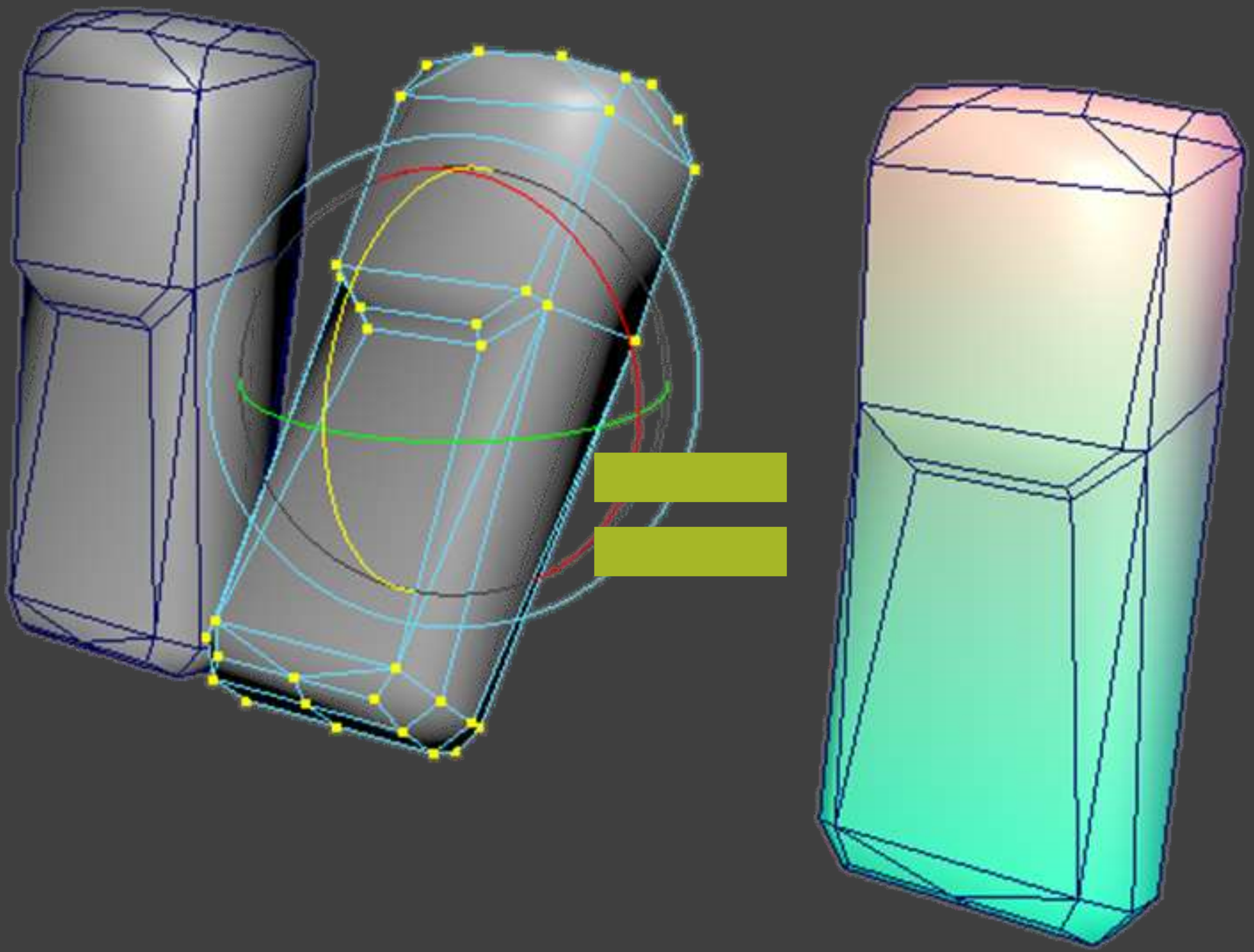

Game Time
Noise
Amplitude
Frequency
1.0
1.0
Vertex Color
Red
Channel
Vertex Animation
Local Space Displacement
X
Y
Z
Multiply
A
B
Multiply
A
B
Add
A
B
Game Time
Add
A
B
Vertex Color
Blue
Channel
Add
A
B
Cos
Amplitude
Frequency
1.0
1.0
World Position
Z
Axis
Parameter
Name
Dampened Linear Velocity



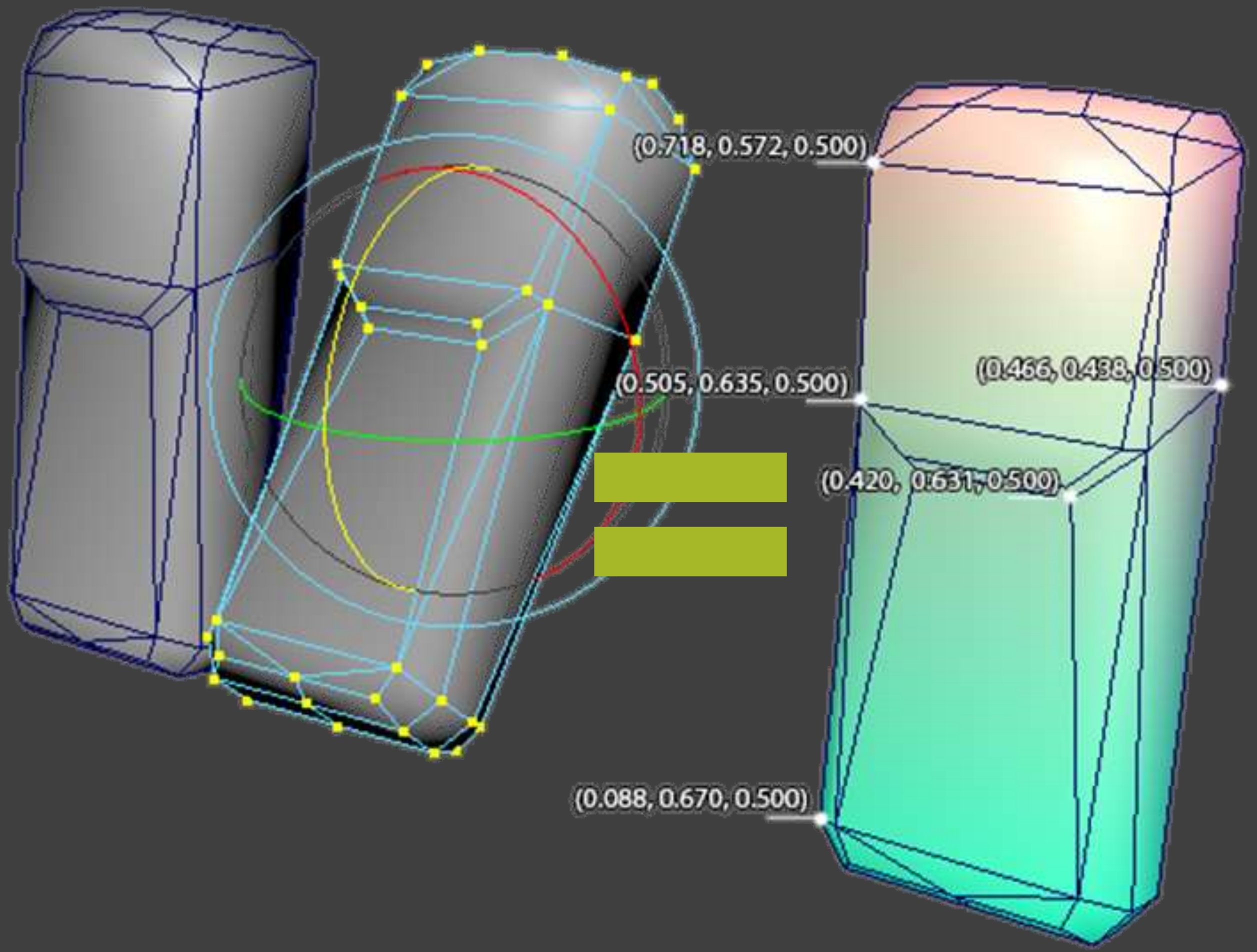
GO TO ORBIT

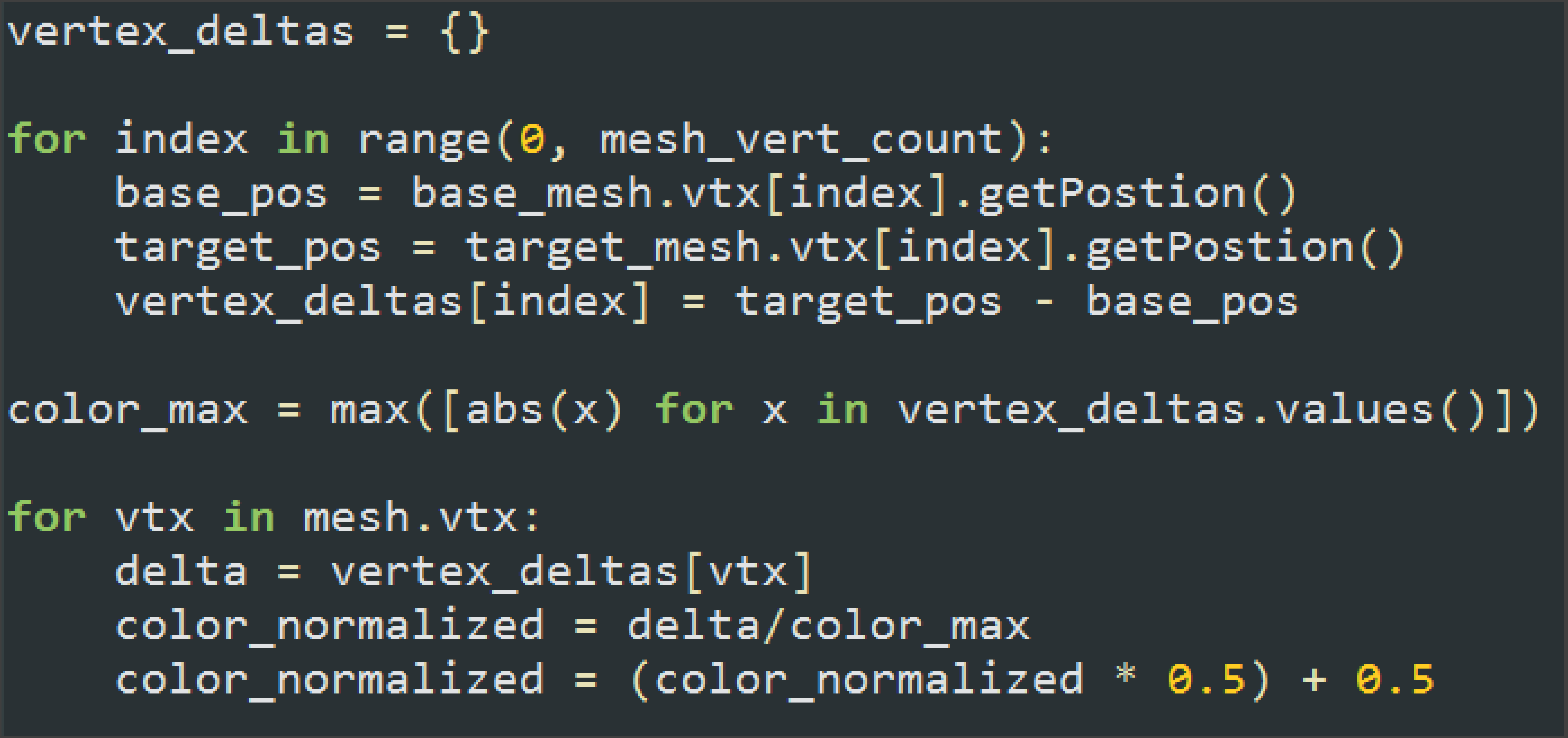
When you are finished in the Tower, press ▭ then hold △ to go to Orbit.

Blendshapes

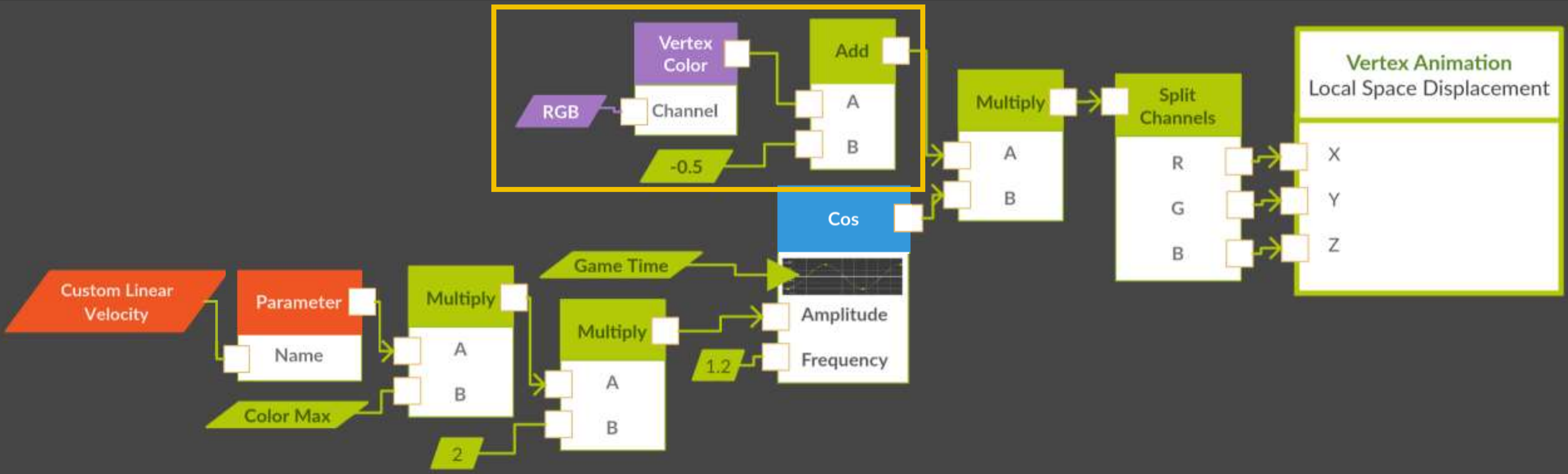


XYZ positions become RBG values

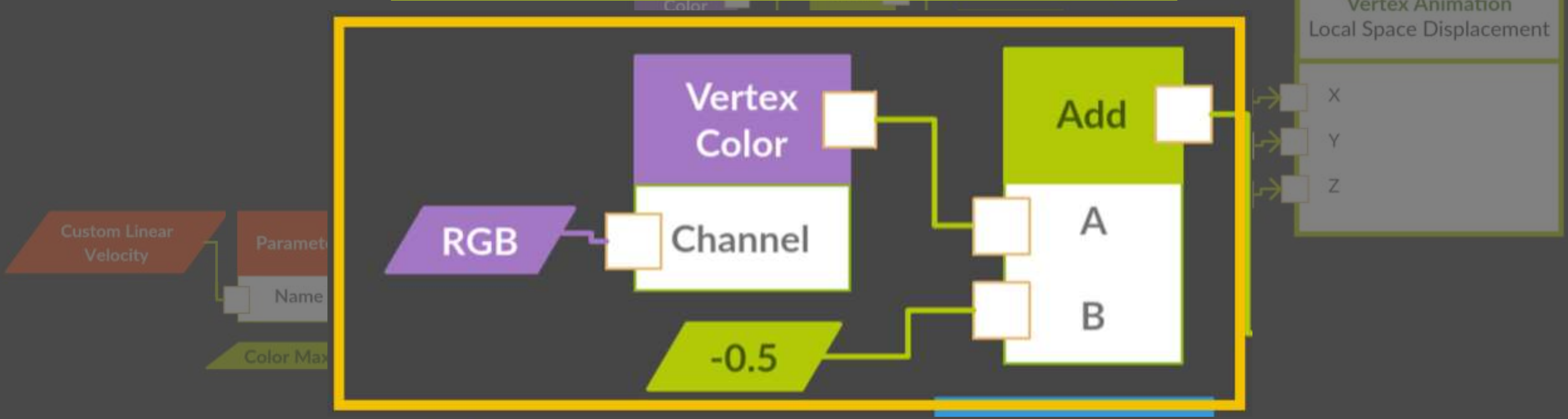


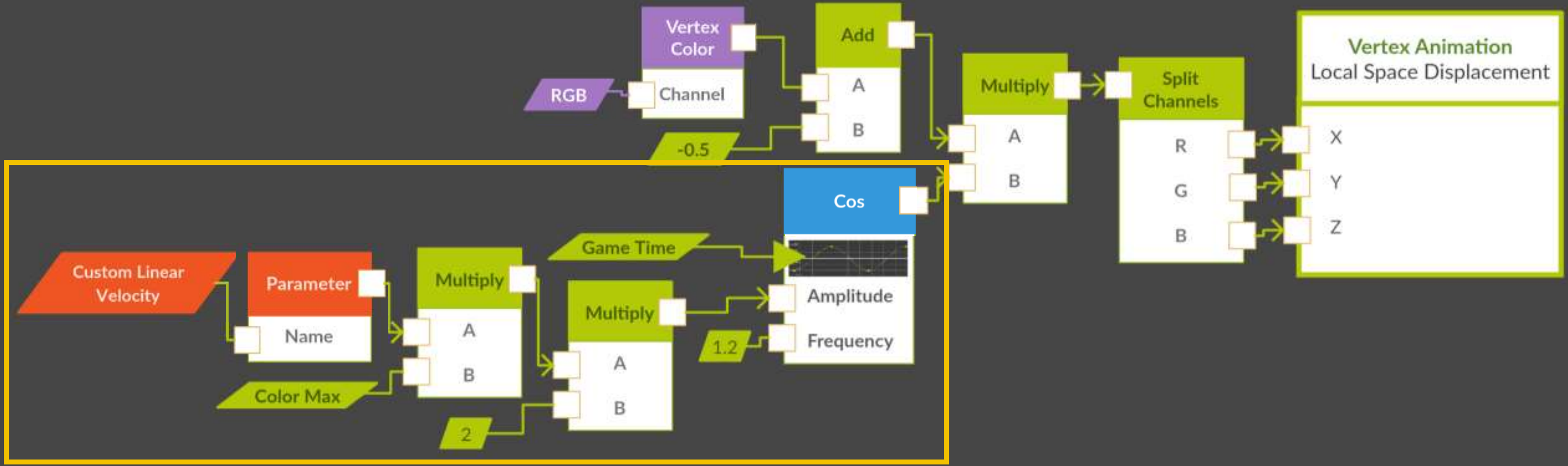


```
vertex_deltas = {}

for index in range(0, mesh_vert_count):
    base_pos = base_mesh.vtx[index].getPostion()
    target_pos = target_mesh.vtx[index].getPostion()
    vertex_deltas[index] = target_pos - base_pos

color_max = max([abs(x) for x in vertex_deltas.values()])

for vtx in mesh.vtx:
    delta = vertex_deltas[vtx]
    color_normalized = delta/color_max
    color_normalized = (color_normalized * 0.5) + 0.5
```


Vertex Color
RGB
Channel
Add
A
B
-0.5
Multiply
A
B
Split Channels
R
G
B
Vertex Animation
Local Space Displacement
X
Y
Z
Cos
Game Time
Amplitude
Frequency
1.2
Custom Linear Velocity
Parameter
Name
Multiply
A
B
Color Max
Multiply
A
B
2

(Red – 0.5, Green – 0.5, Blue – 0.5)





Vertex Color

RGB

Channel

-0.5

Add

A

B

Multiply

A

B

Split Channels

R

G

B

Vertex Animation

Local Space Displacement

X

Y

Z

Cos

Game Time

Amplitude

Frequency

1.2

Custom Linear Velocity

Parameter

Name

Multiply

A

B

Color Max

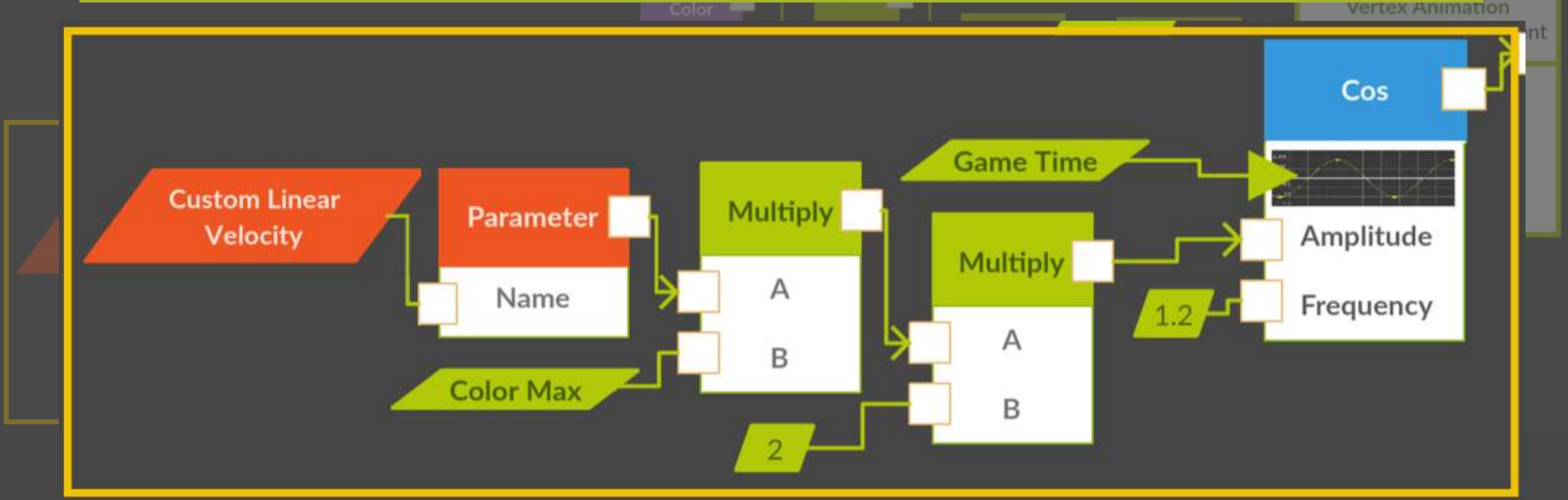
Multiply

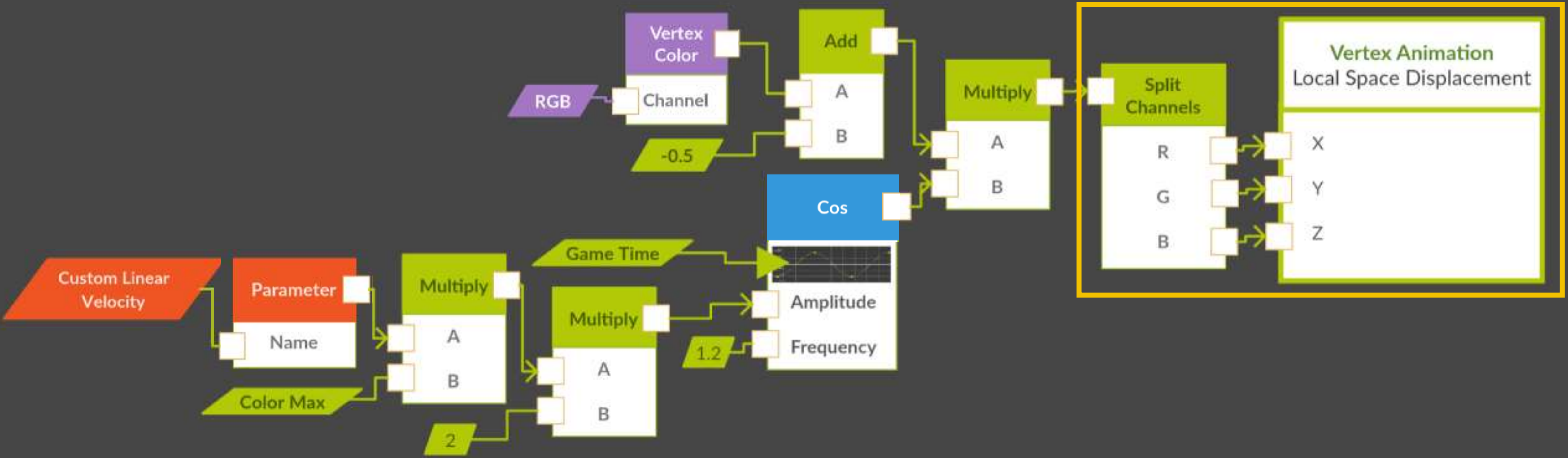
A

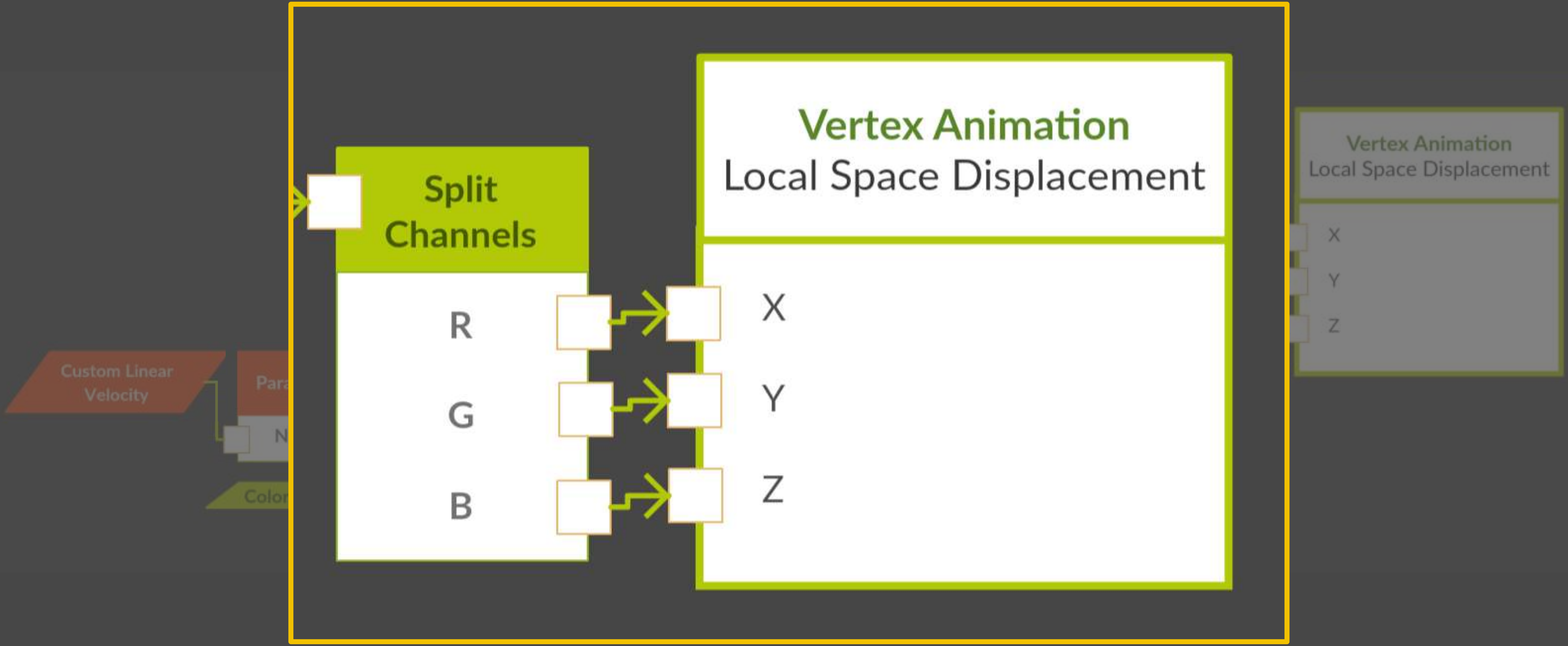
B

2

Custom Linear Velocity = clamp(dampened_linear_velocity, 0, 1)
Custom Linear Velocity * Color Max * 2




Vertex Color
RGB
Channel
Add
A
B
-0.5
Multiply
A
B
Split Channels
R
G
B
Vertex Animation
Local Space Displacement
X
Y
Z
Cos
Game Time
Amplitude
Frequency
1.2
Custom Linear Velocity
Parameter
Name
Multiply
A
B
Color Max
Multiply
A
B
2



Custom Linear Velocity

Para

N

Color

Split Channels

R

G

B

Vertex Animation

Local Space Displacement

X

Y

Z

Vertex Animation

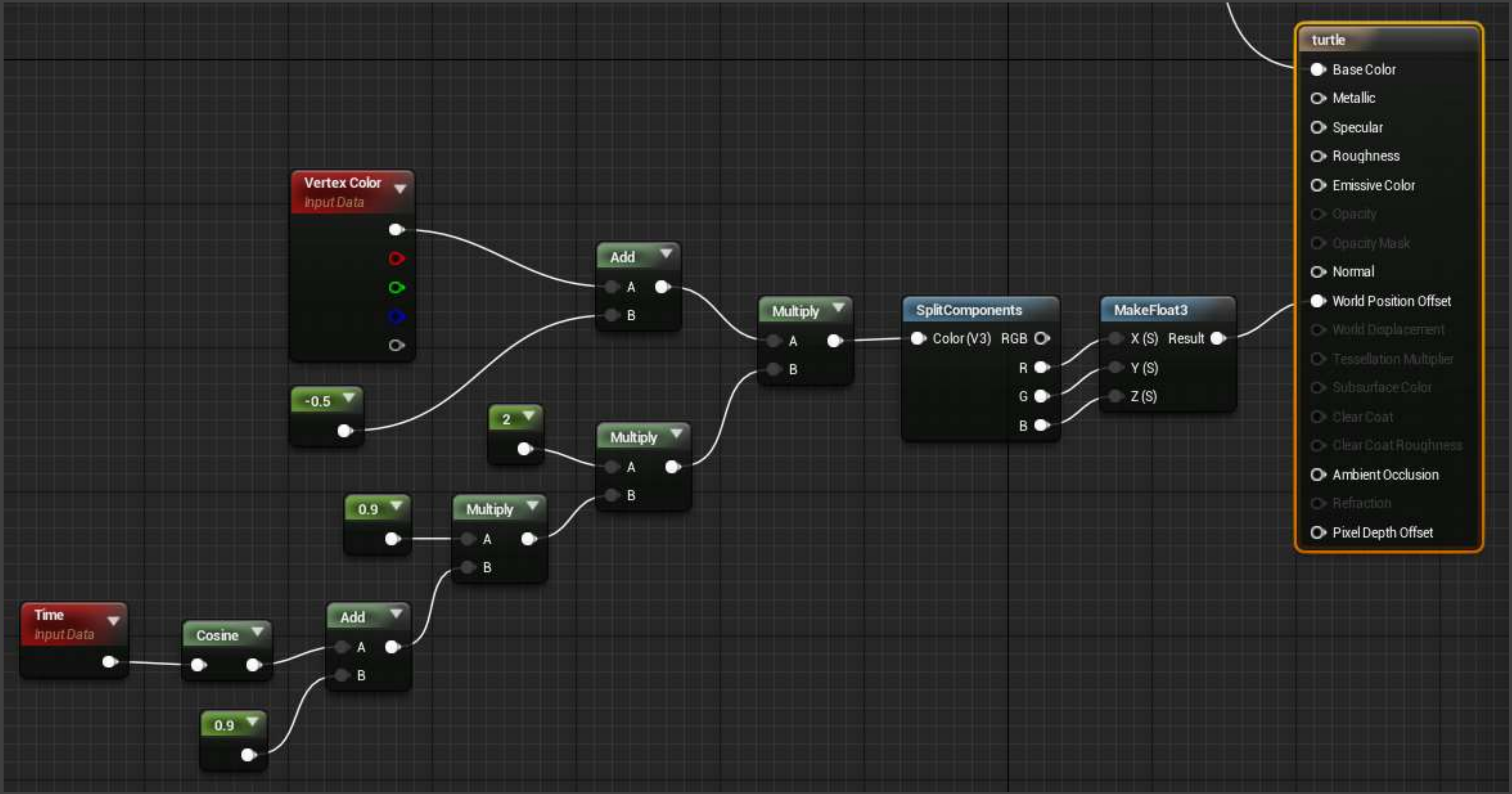
Local Space Displacement

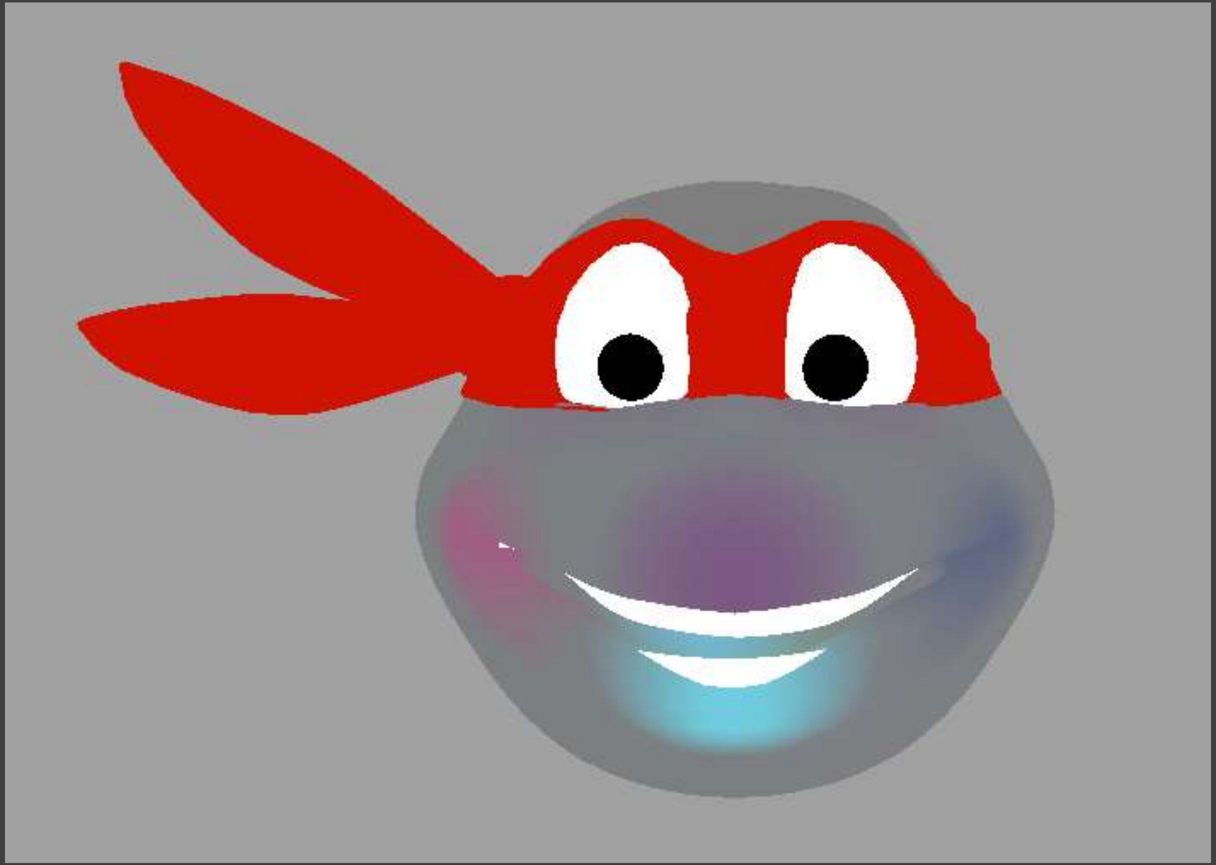
X

Y

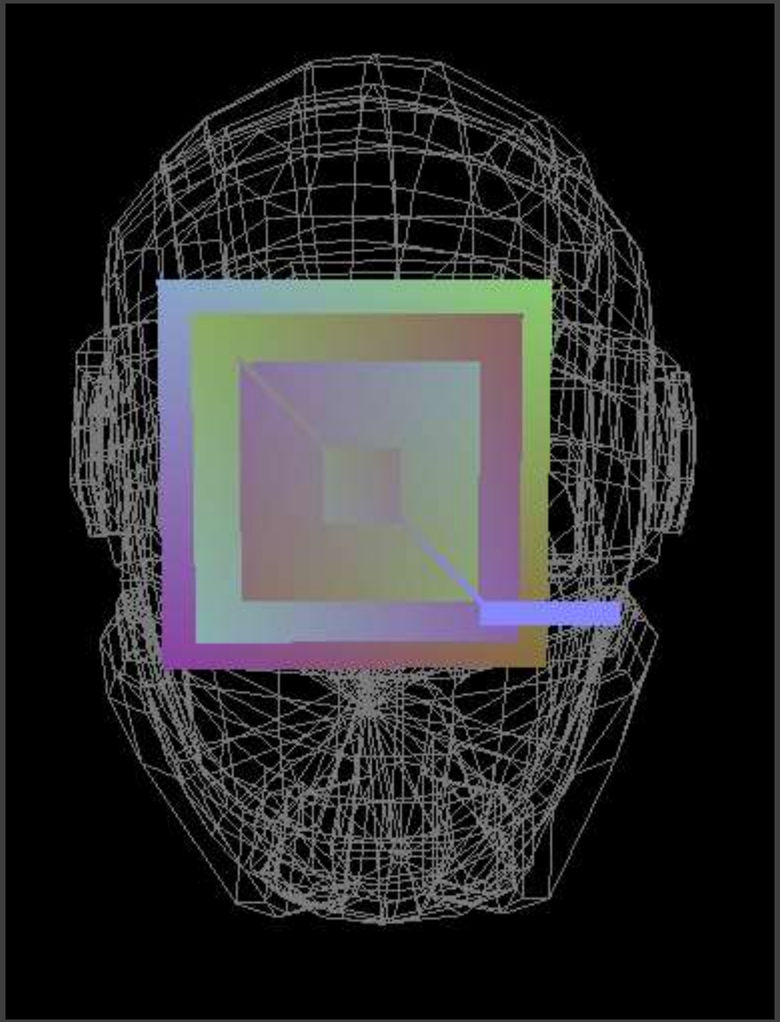
Z

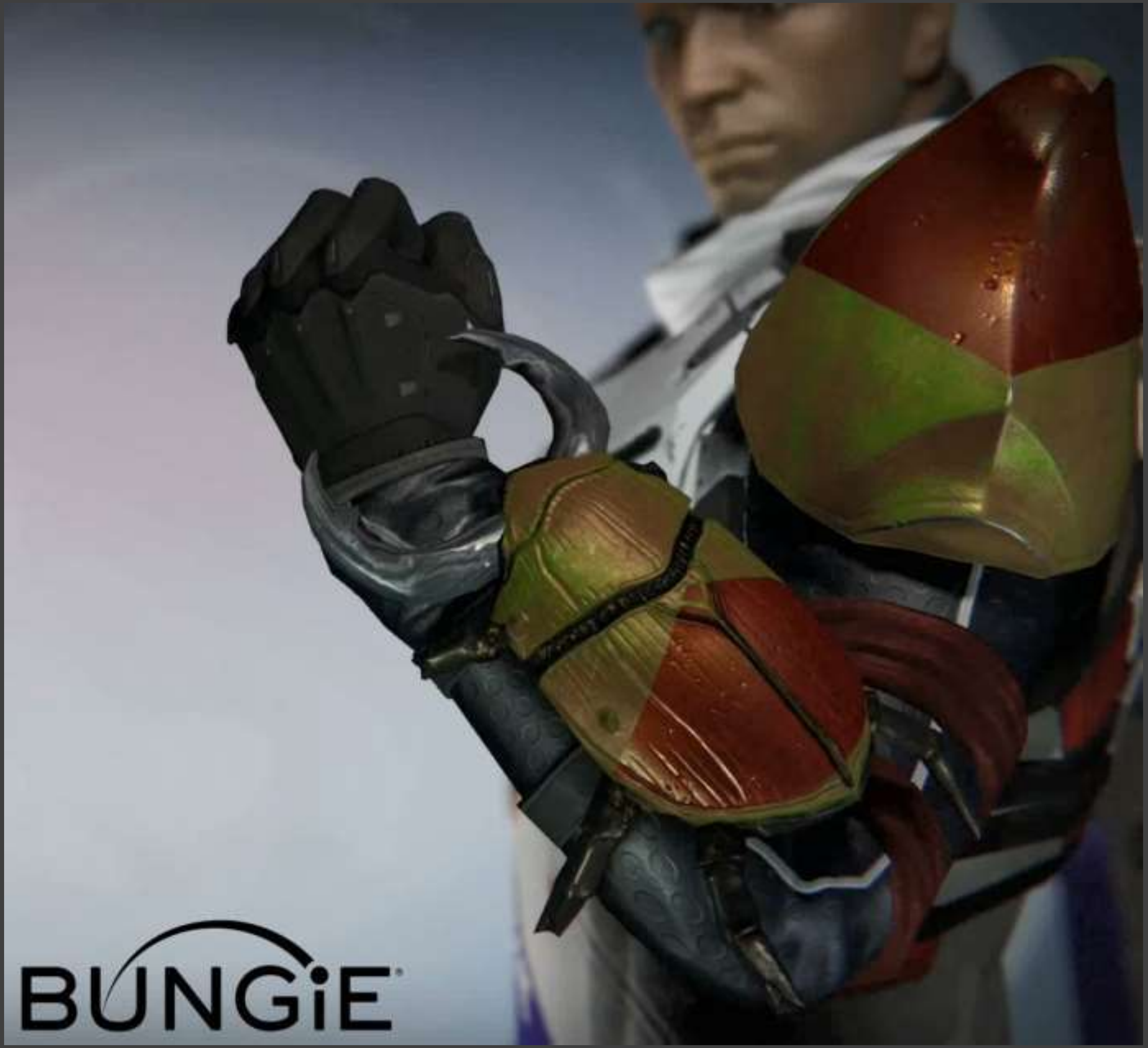
Unreal Engine Example



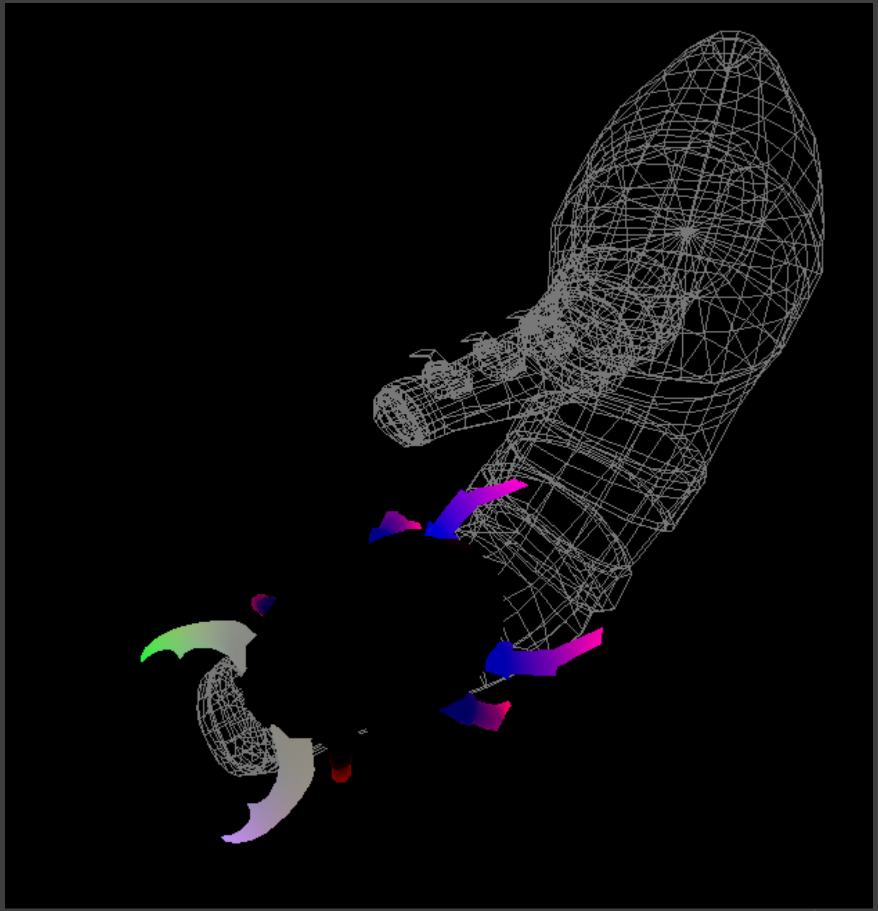


Knucklehead Radar

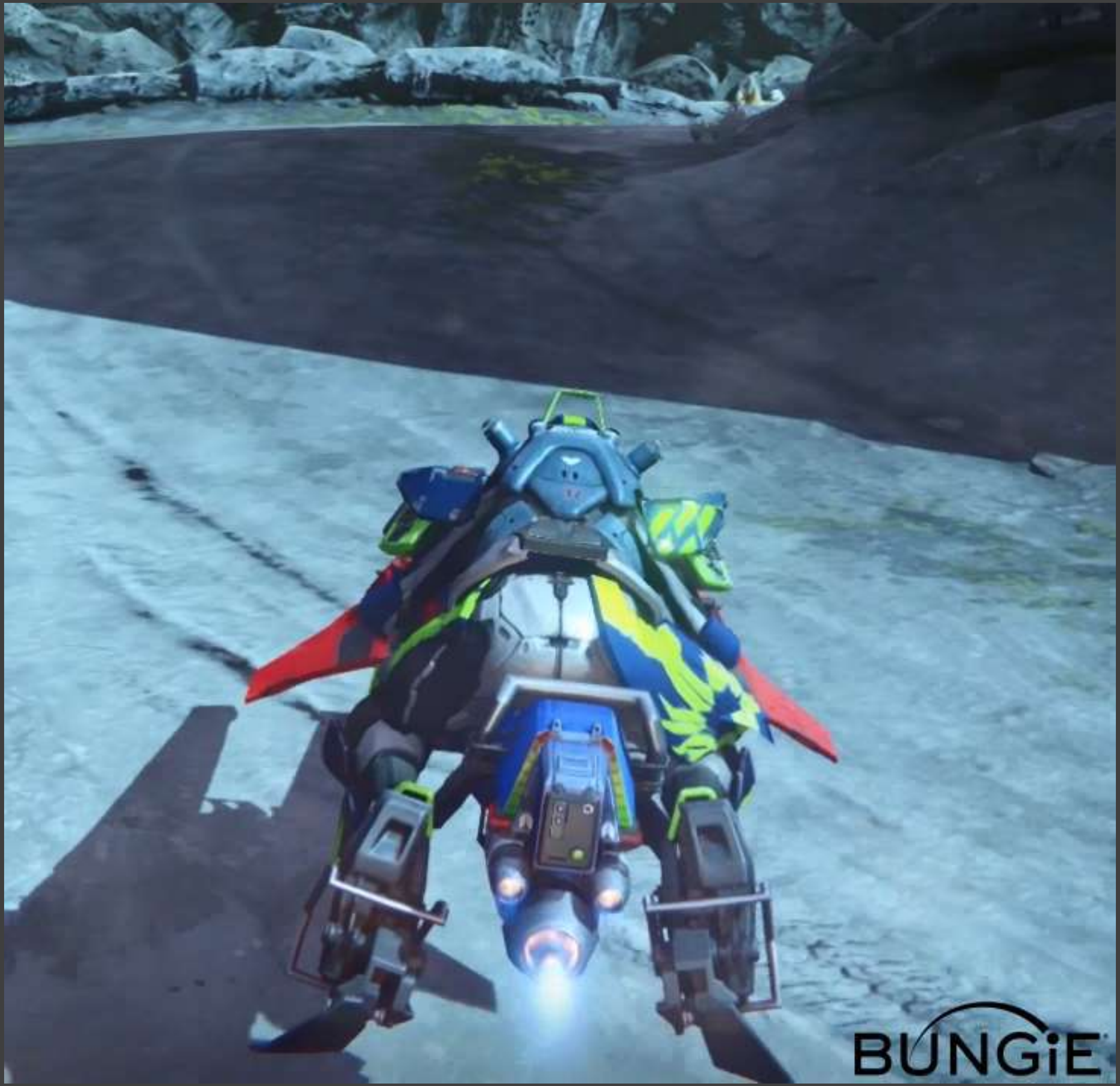


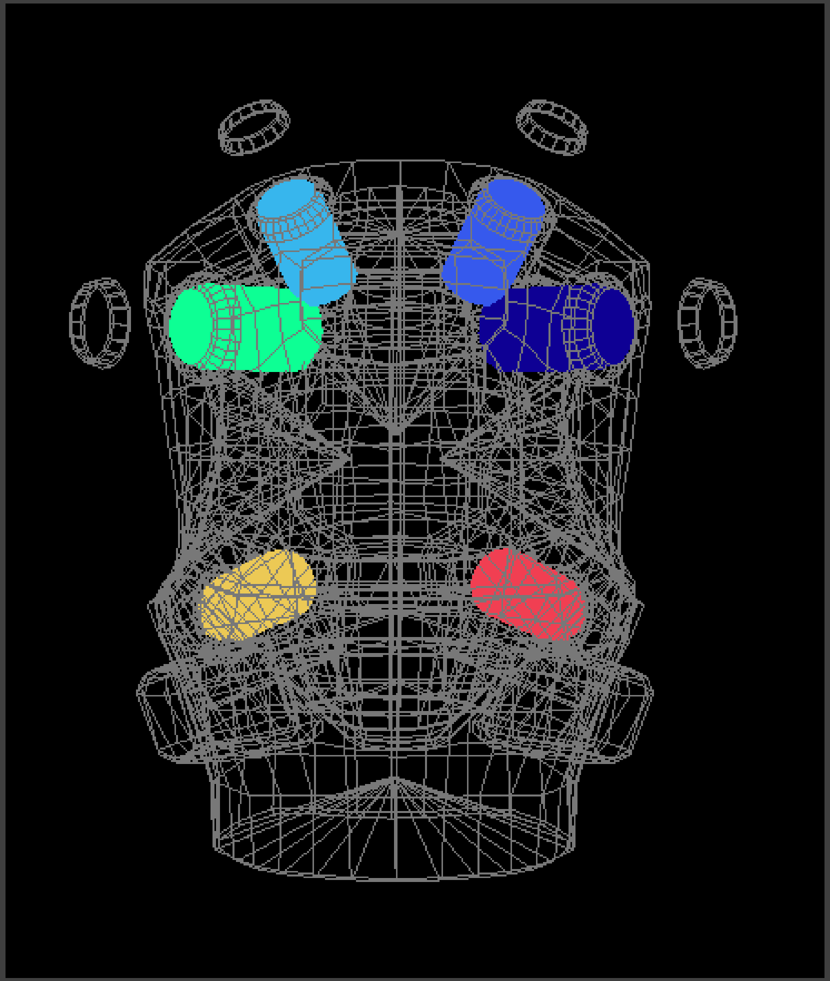


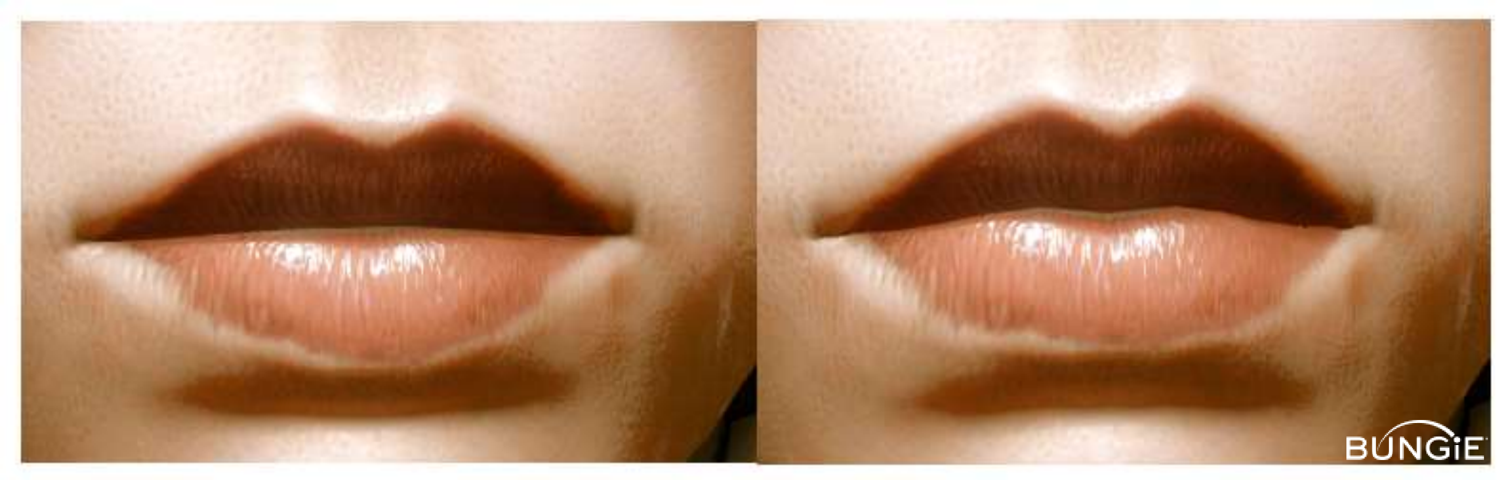
Khepri's Sting

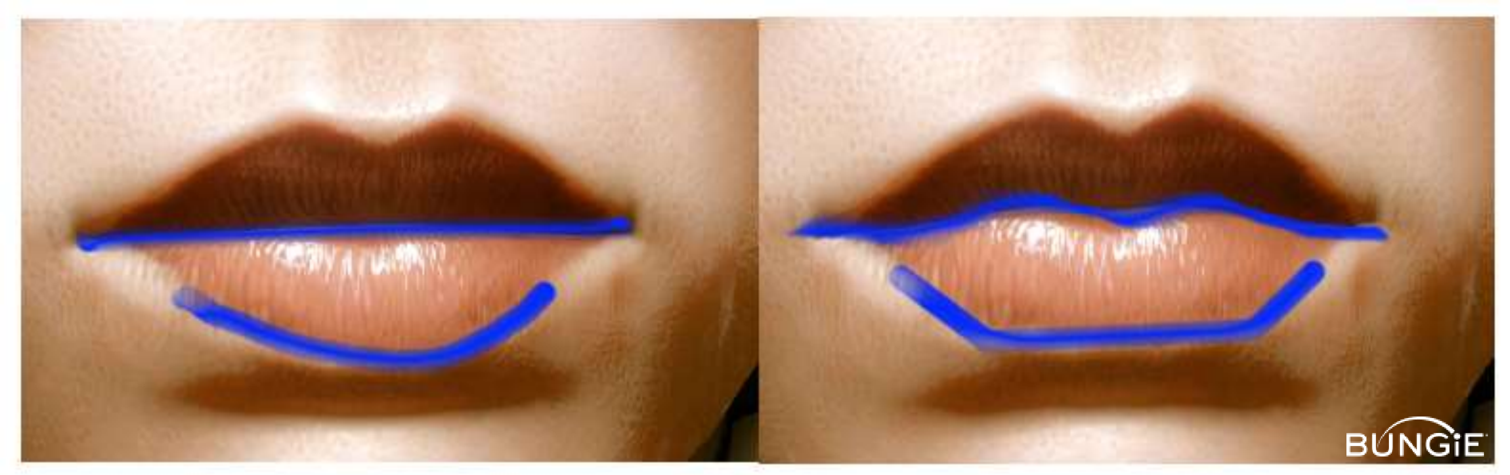


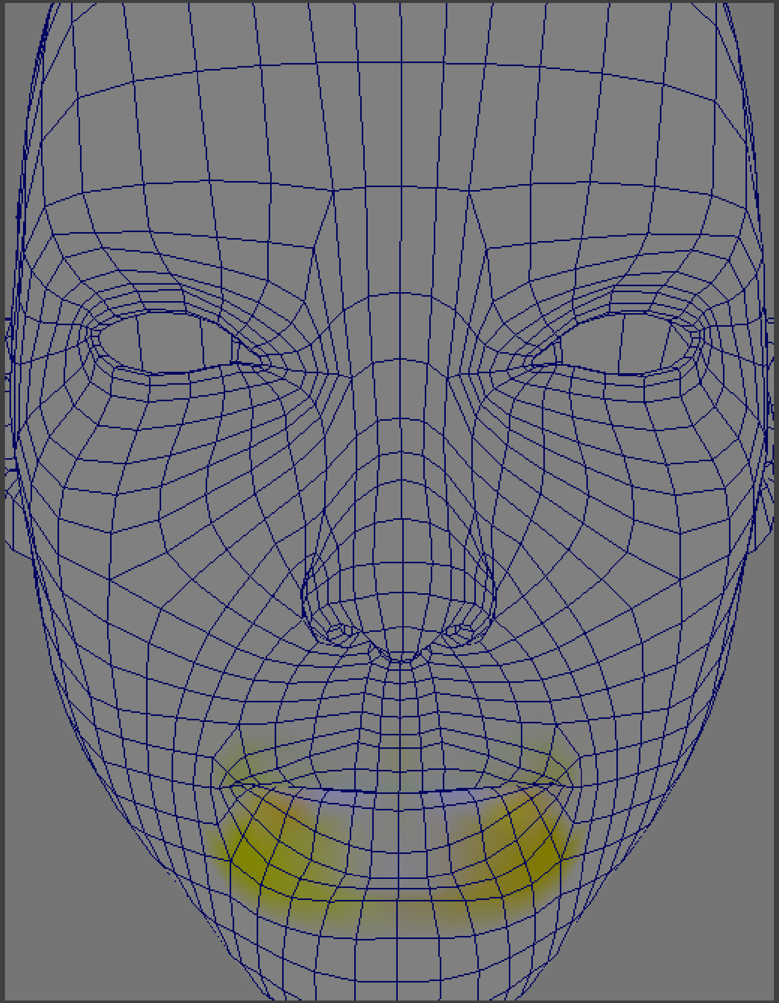
Twilight Garrison



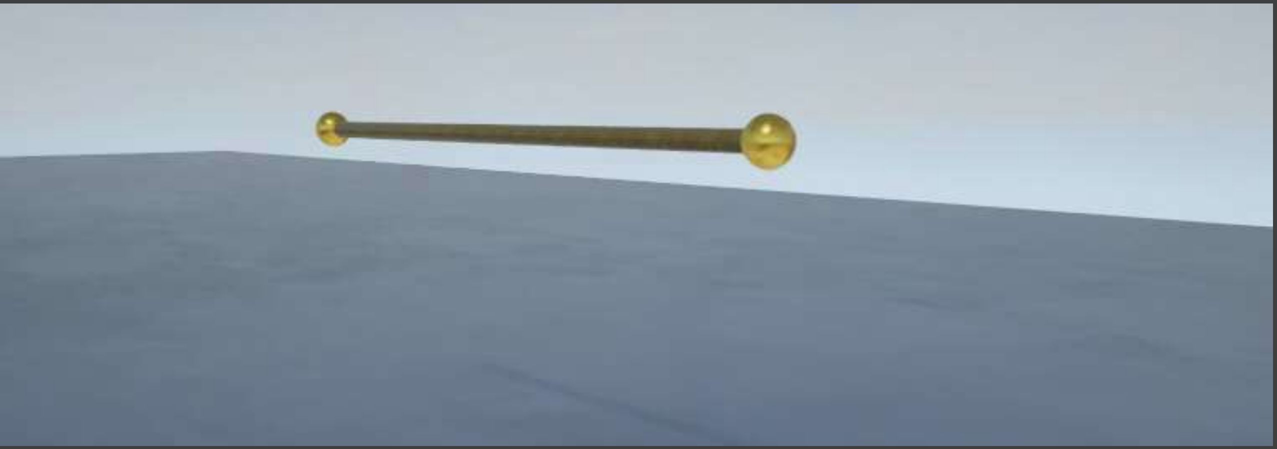


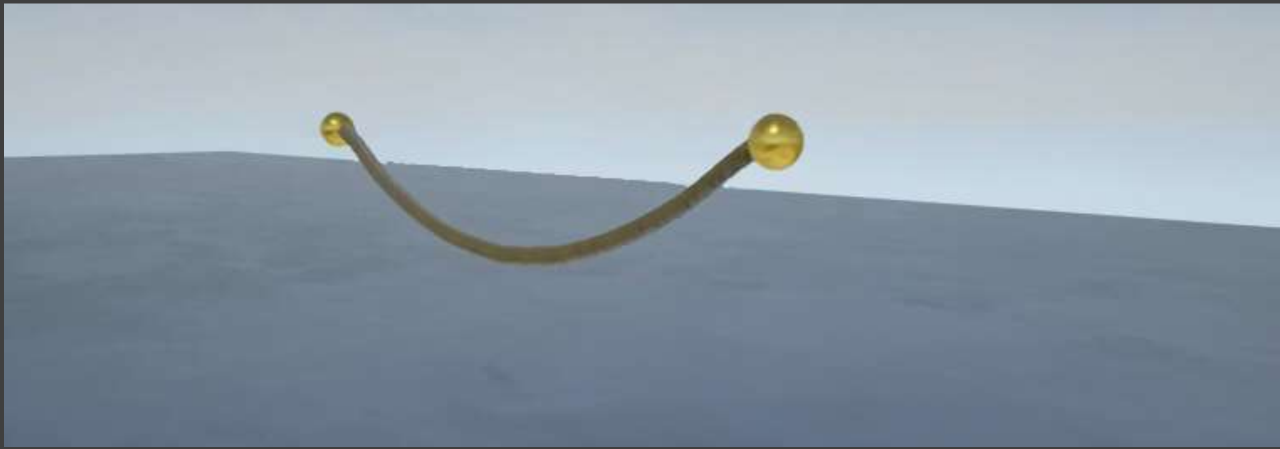






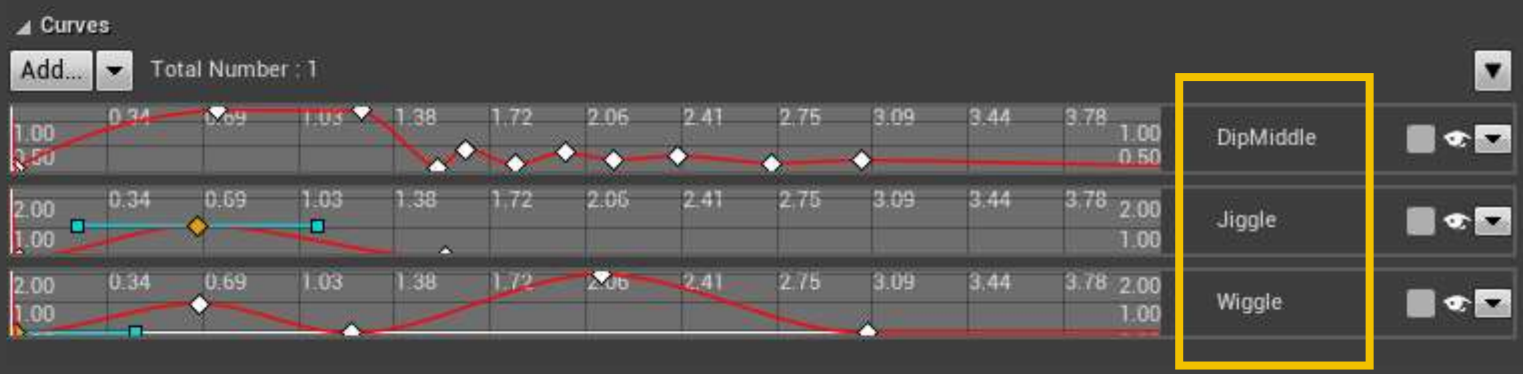
Unreal Engine Example



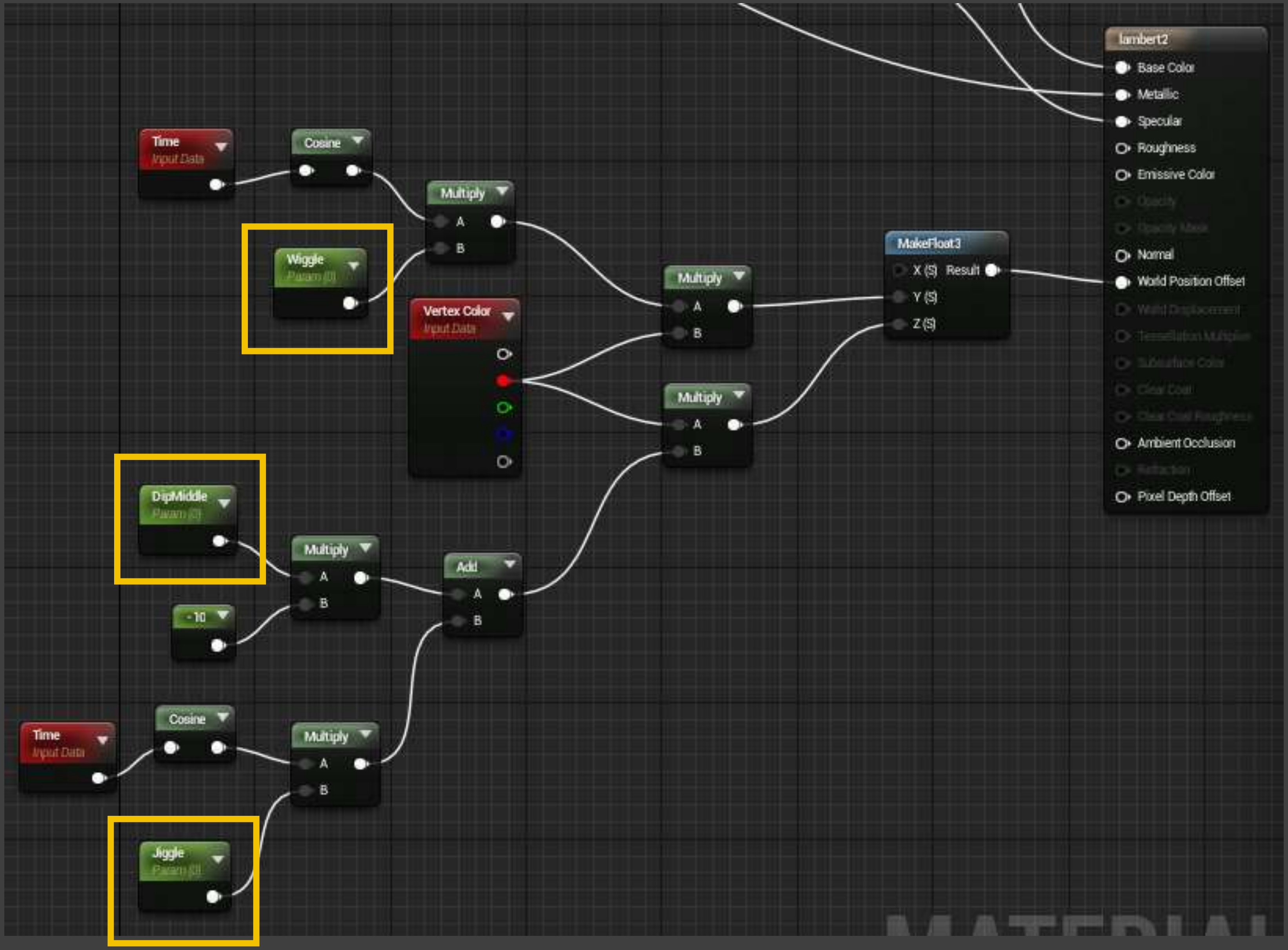


Without Vertex Animation

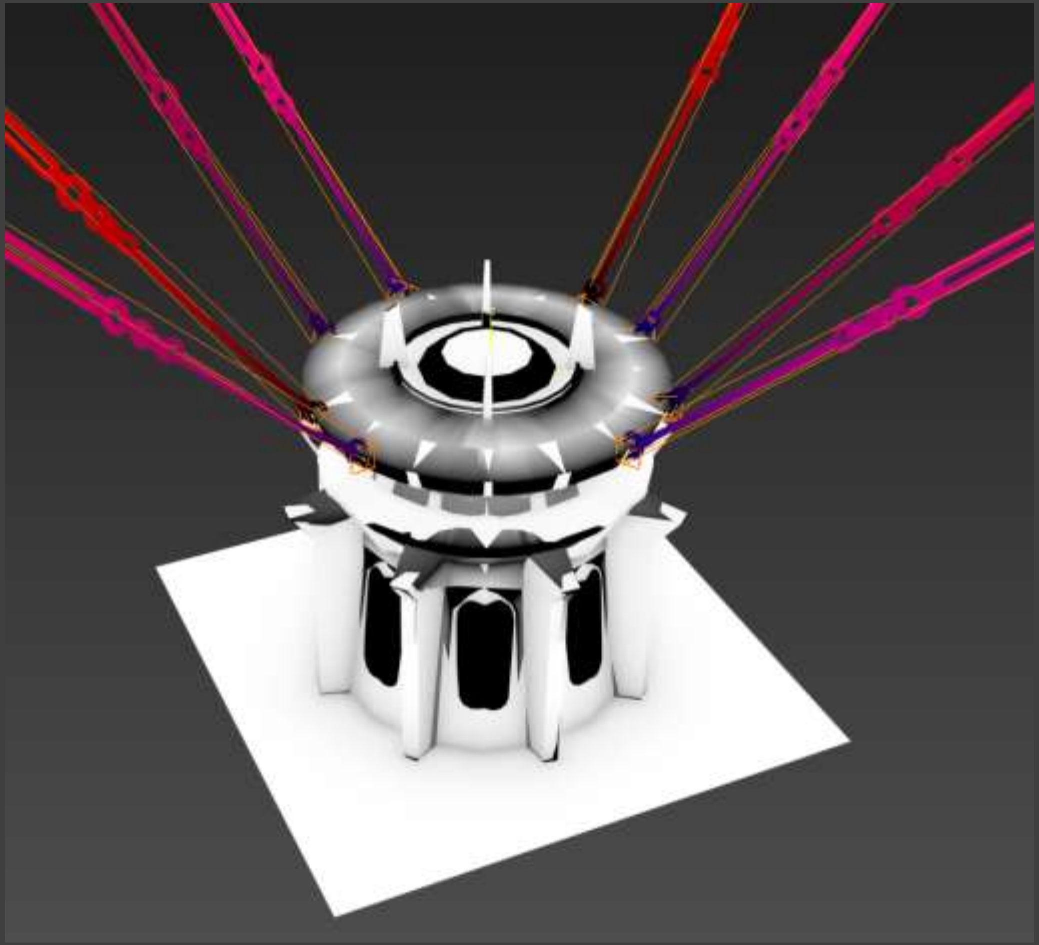
With Vertex Animation

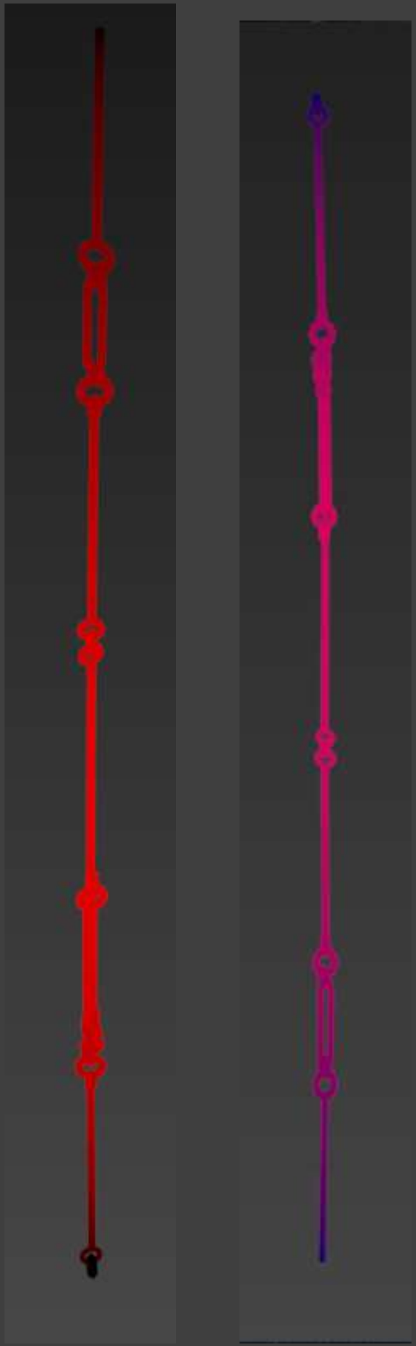


Sometimes Math Alone is Not Enough!!


Time
Input Data
Cosine
Multiply
A
B
Wiggle
Param (0)
Vertex Color
Input Data
Multiply
A
B
Multiply
A
B
MakeFloat3
X (S)
Result
Y (S)
Z (S)
lambert2
Base Color
Metallic
Specular
Roughness
Emissive Color
Opacity
Opacity Mask
Normal
World Position Offset
World Displacement
Tessellation Multiplier
Subsurface Color
Clear Coat
Clear Coat Roughness
Ambient Occlusion
Refraction
Pixel Depth Offset
DipMiddle
Param (0)
Multiply
A
B
-10
Add
A
B
Time
Input Data
Cosine
Multiply
A
B
Jiggle
Param (0)
MATERIAL

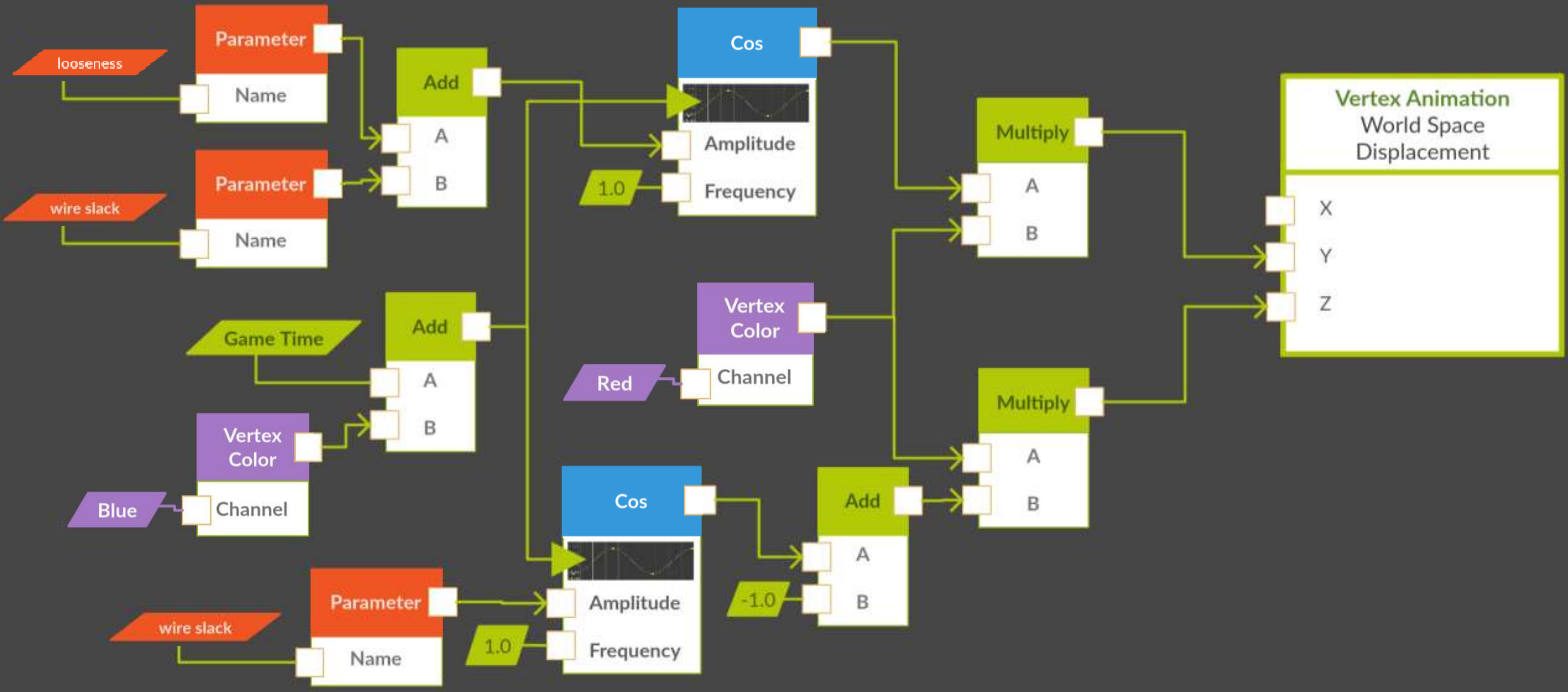
BUNGiE

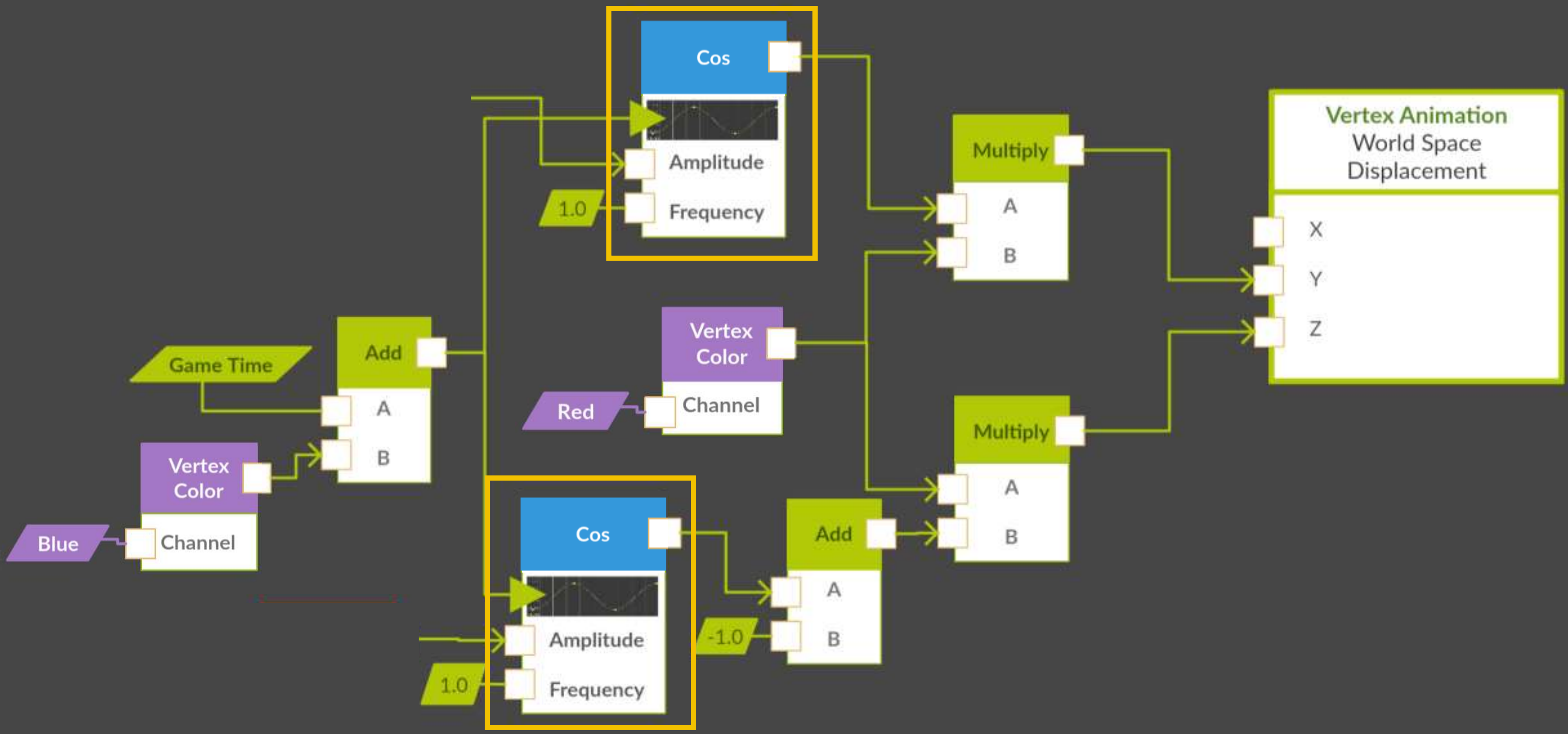


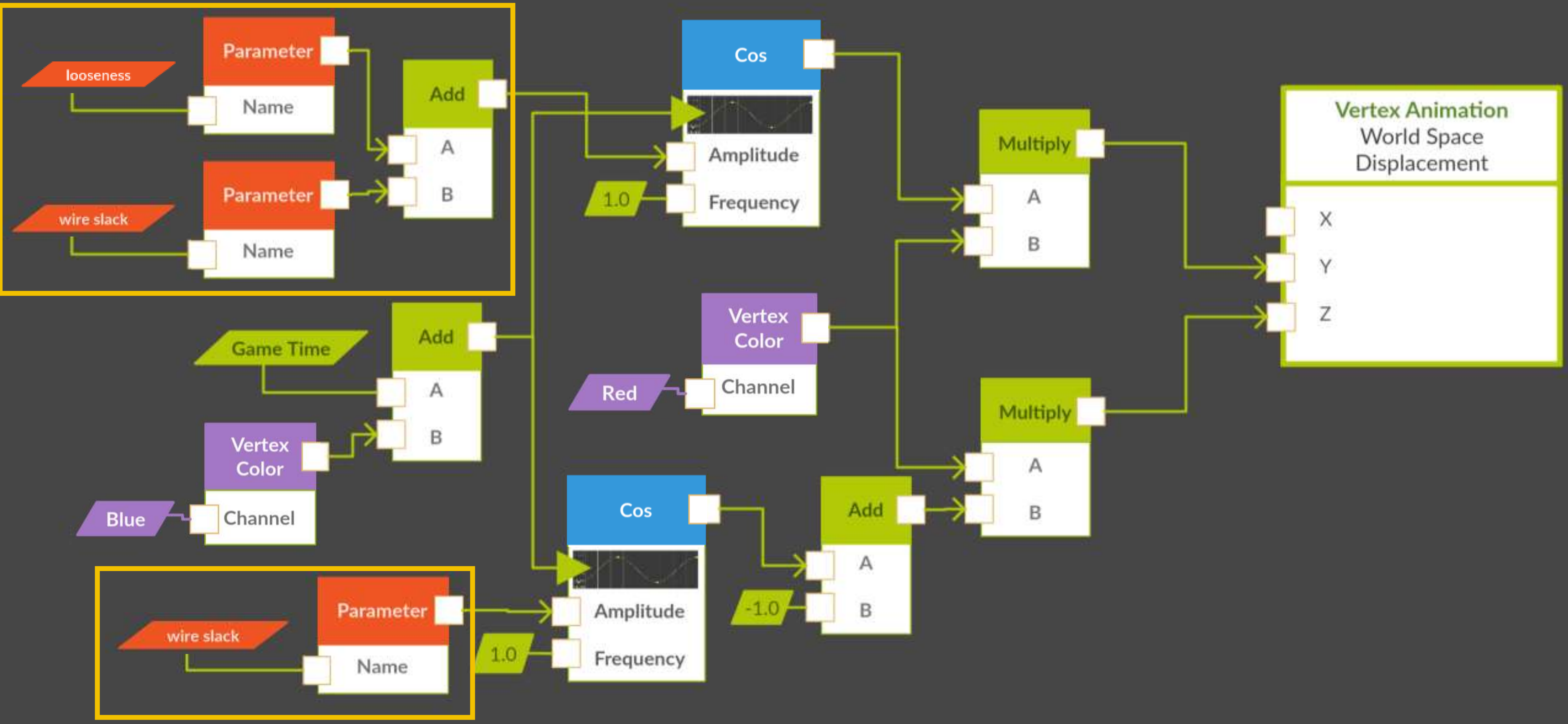


Red = Amount of movement

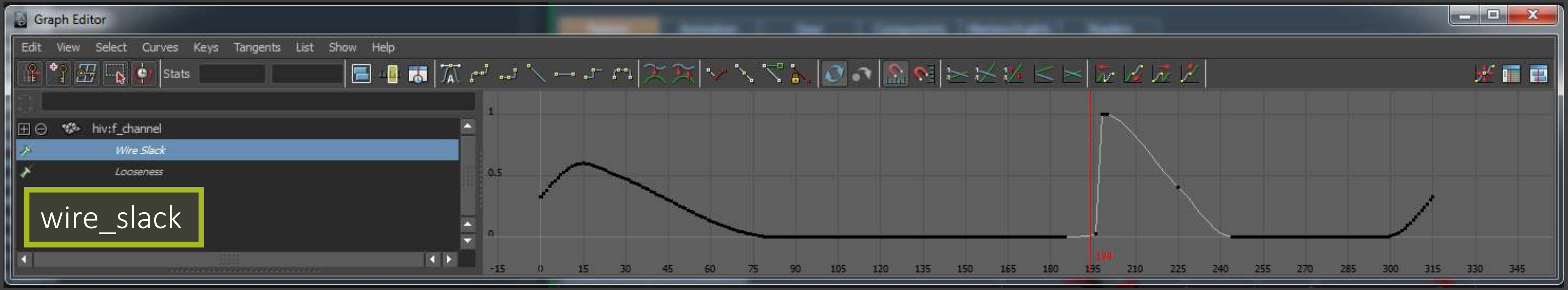
Blue = Timing of movement

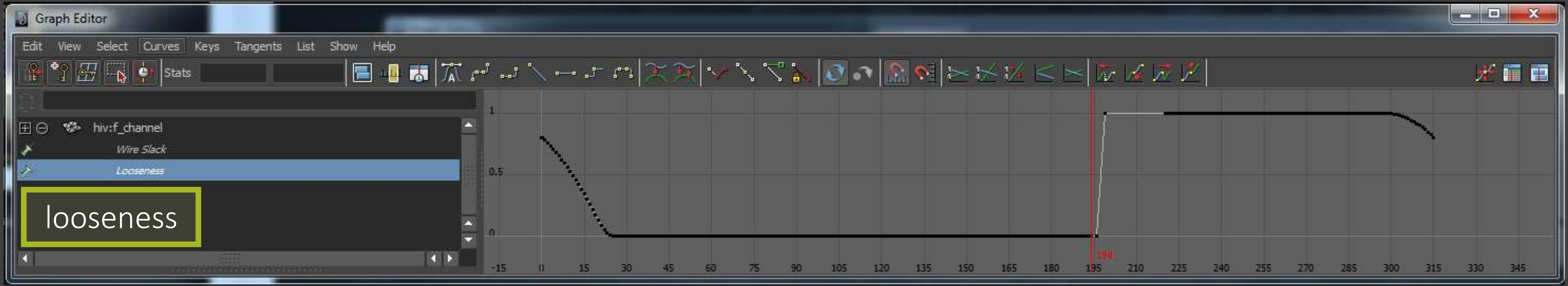

looseness
Parameter
Name
wire slack
Parameter
Name
Add
A
B
Cos
Amplitude
Frequency
1.0
Multiply
A
B
Vertex Animation
World Space
Displacement
X
Y
Z
Game Time
Add
A
B
Vertex Color
Channel
Blue
Vertex Color
Channel
Red
Multiply
A
B
Cos
Amplitude
Frequency
1.0
Add
A
B
-1.0
wire slack
Parameter
Name

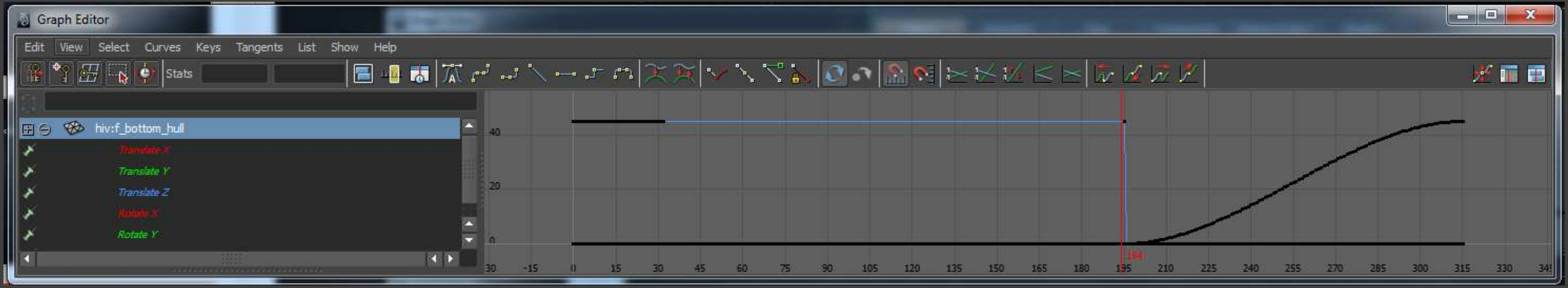

Cos
Amplitude
1.0
Frequency
Multiply
A
B
Vertex Animation
World Space
Displacement
X
Y
Z
Vertex
Color
Red
Channel
Game Time
Add
A
B
Vertex
Color
Blue
Channel
Cos
Amplitude
1.0
Frequency
Add
A
-1.0
B
Multiply
A
B



looseness
Parameter
Name
wire slack
Parameter
Name
Add
A
B
Cos
Amplitude
1.0
Frequency
Multiply
A
B
Vertex Animation
World Space
Displacement
X
Y
Z
Game Time
Add
A
B
Vertex Color
Blue
Channel
Vertex Color
Red
Channel
Multiply
A
B
Cos
Amplitude
1.0
Frequency
Add
A
-1.0
B
wire slack
Parameter
Name


wire_slack

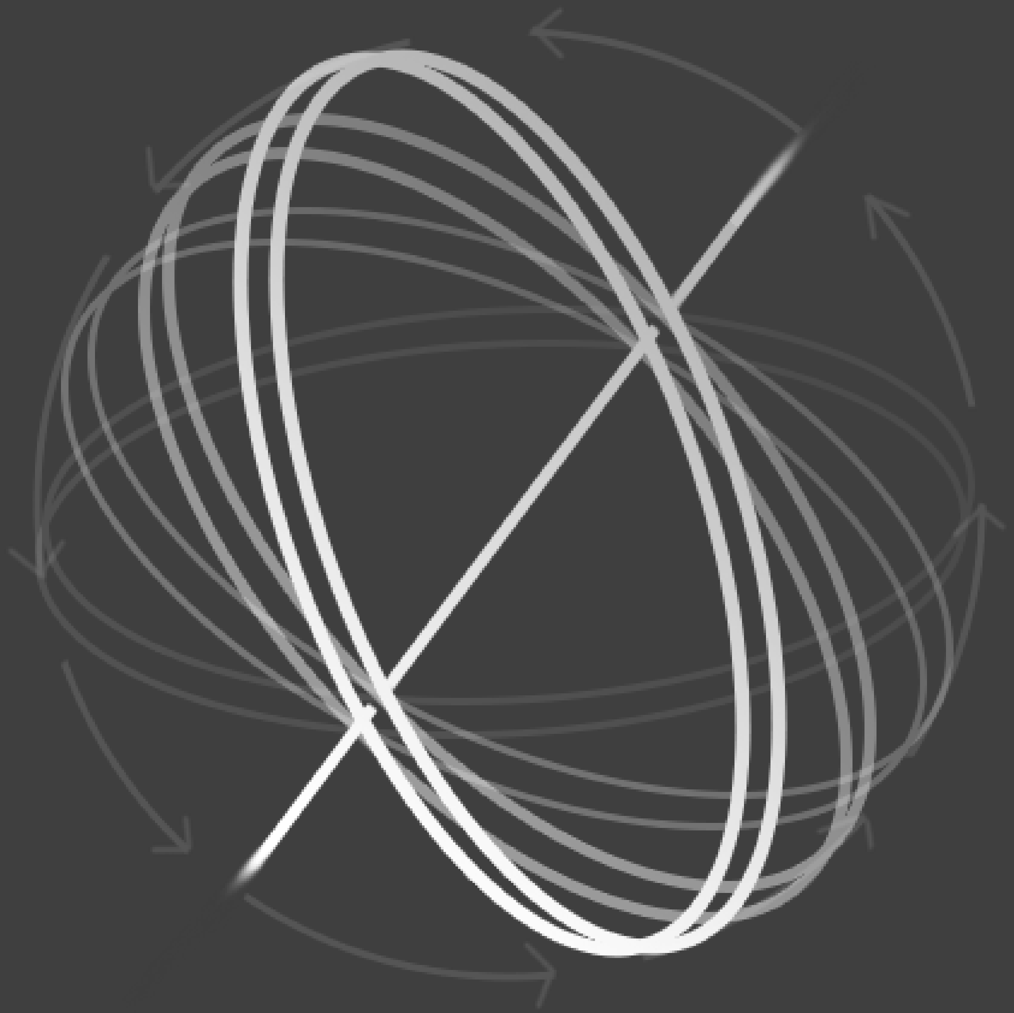

looseness



BUNGiE

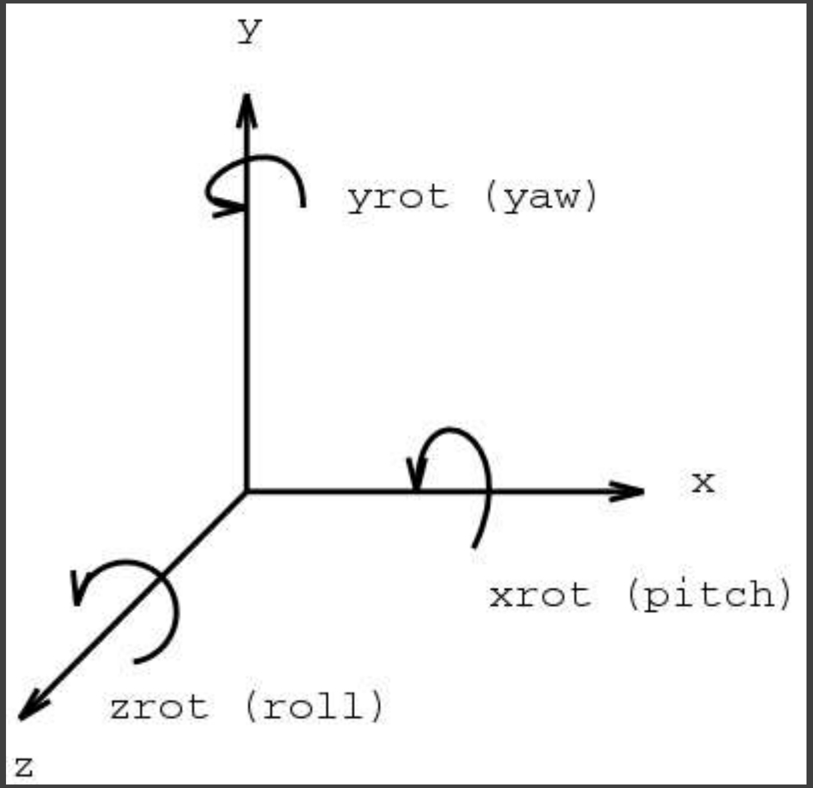
Motion Beyond Oscillation

The Motion Concept

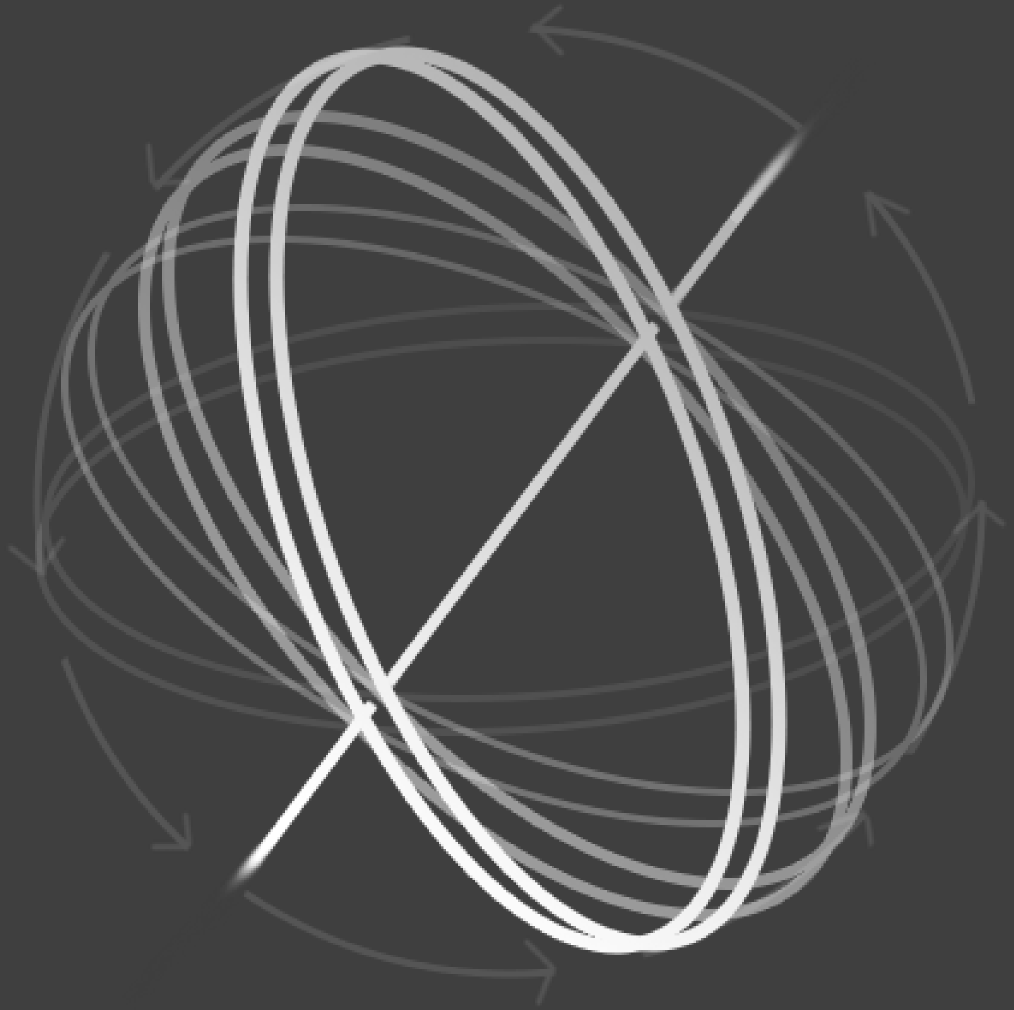


The Math

Rotation about an axis

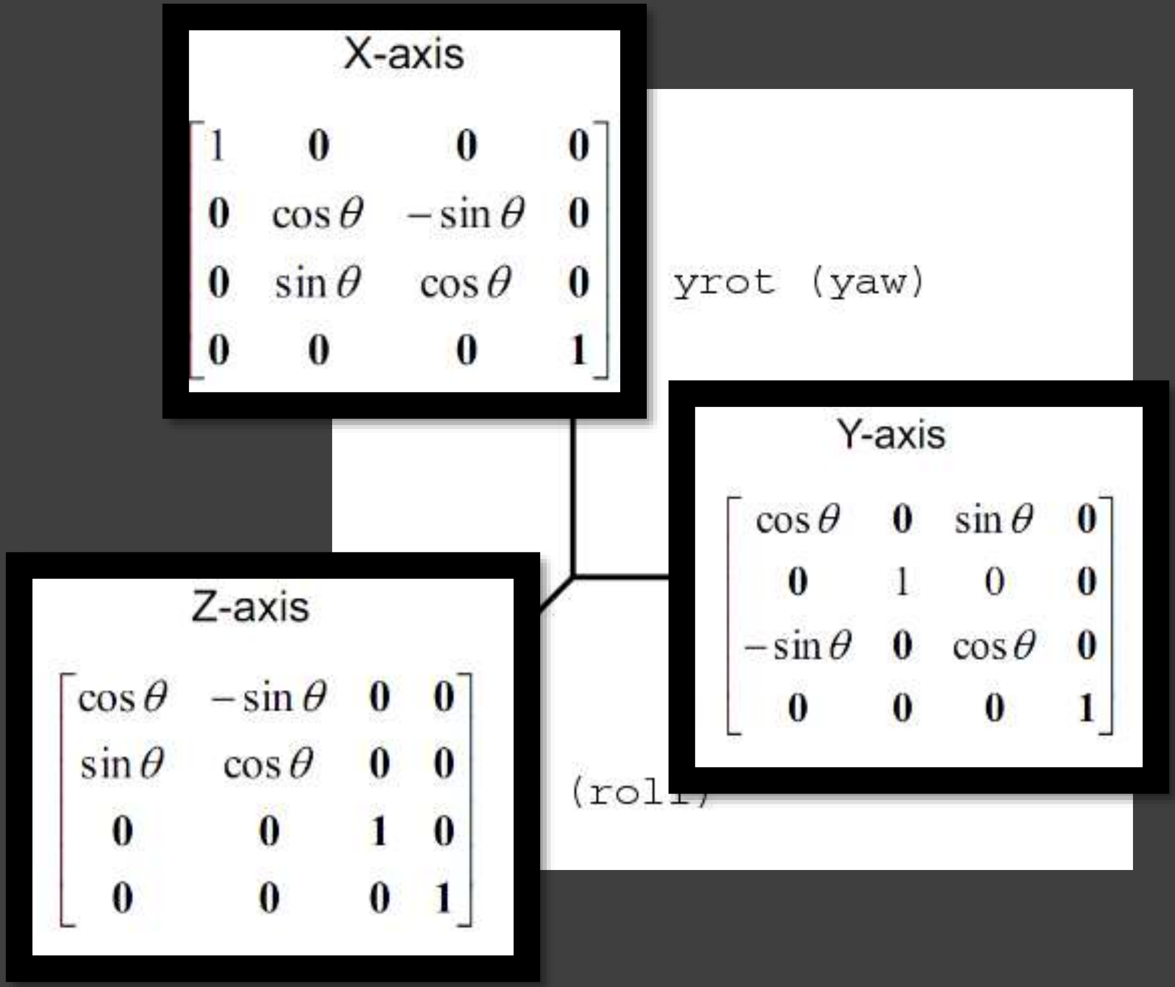


The Motion Concept



The Math

Rotation about an axis



X-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y-axis

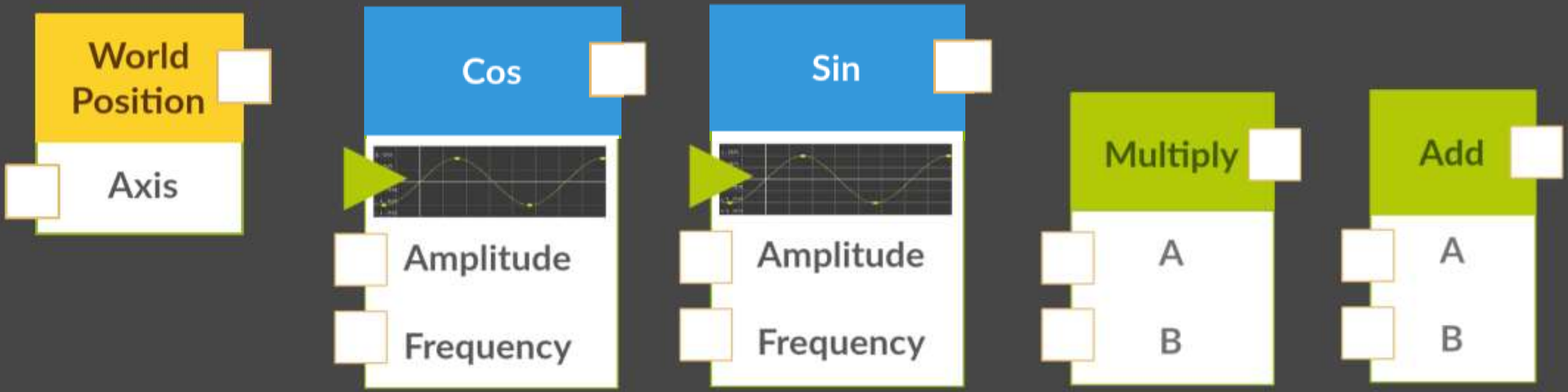
$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-axis

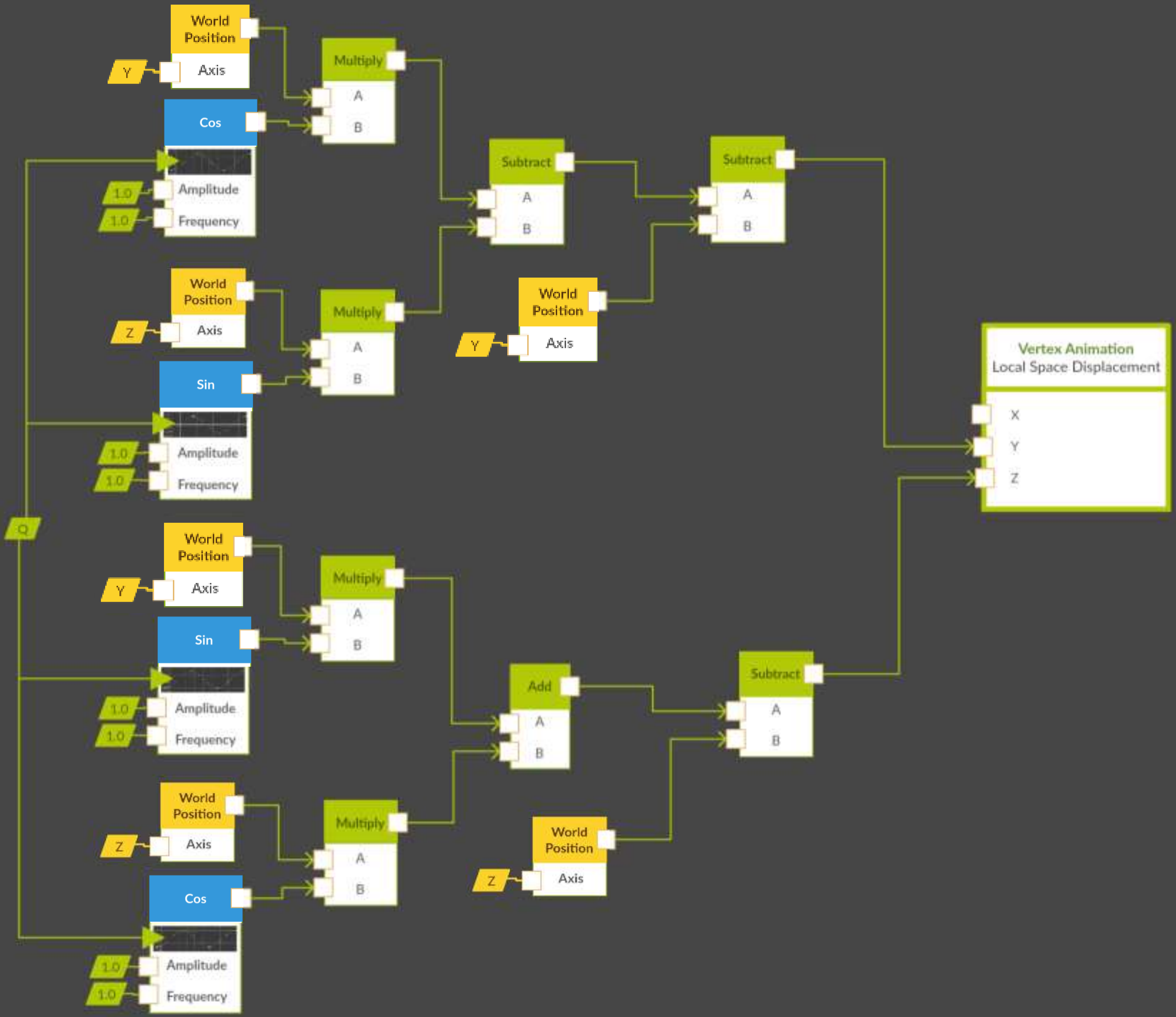
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about the X axis:

y' = y*cos q - z*sin q
z' = y*sin q + z*cos q
x' = x

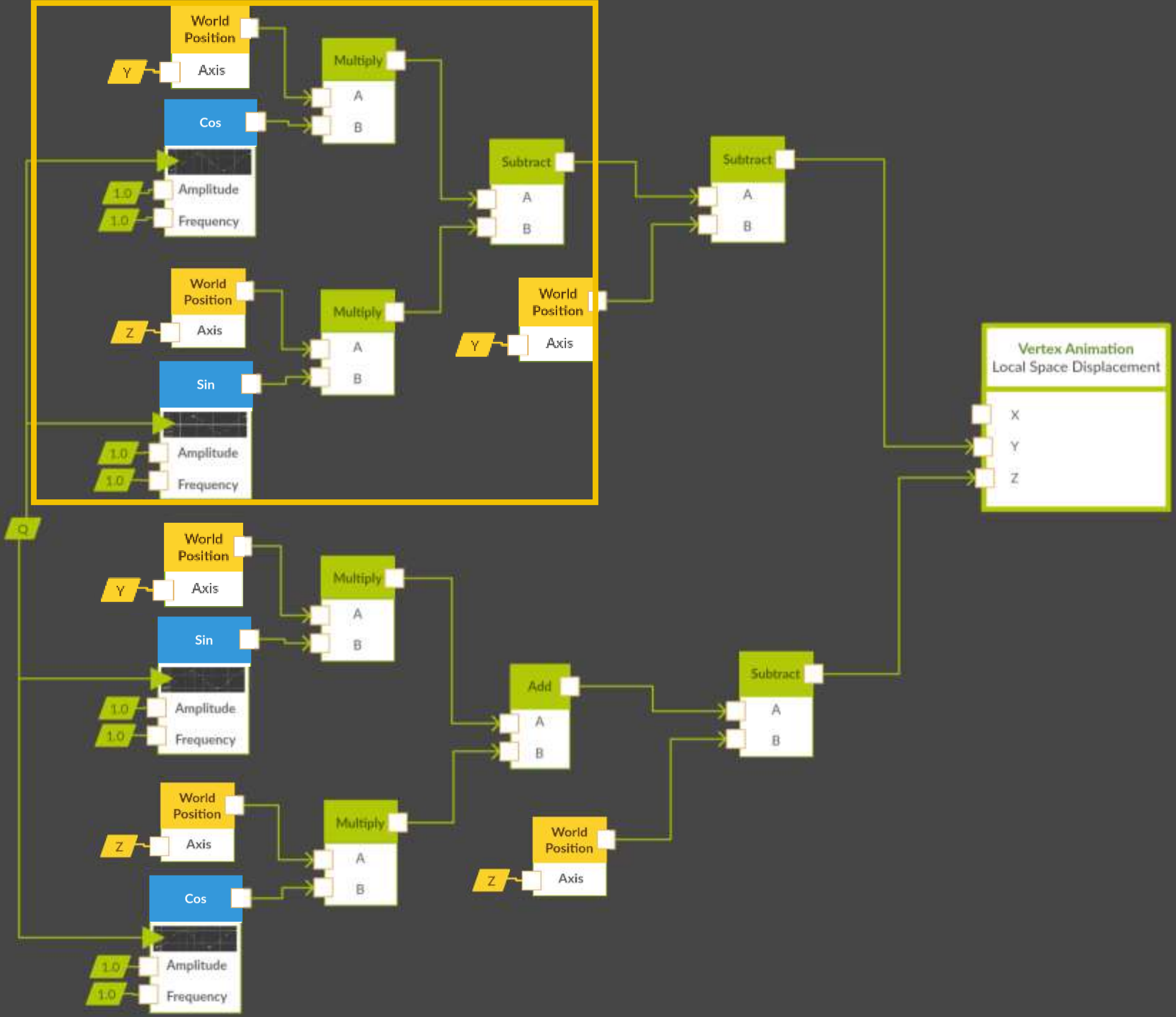


$$y' = y*\cos q - z*\sin q$$
$$z' = y*\sin q + z*\cos q$$

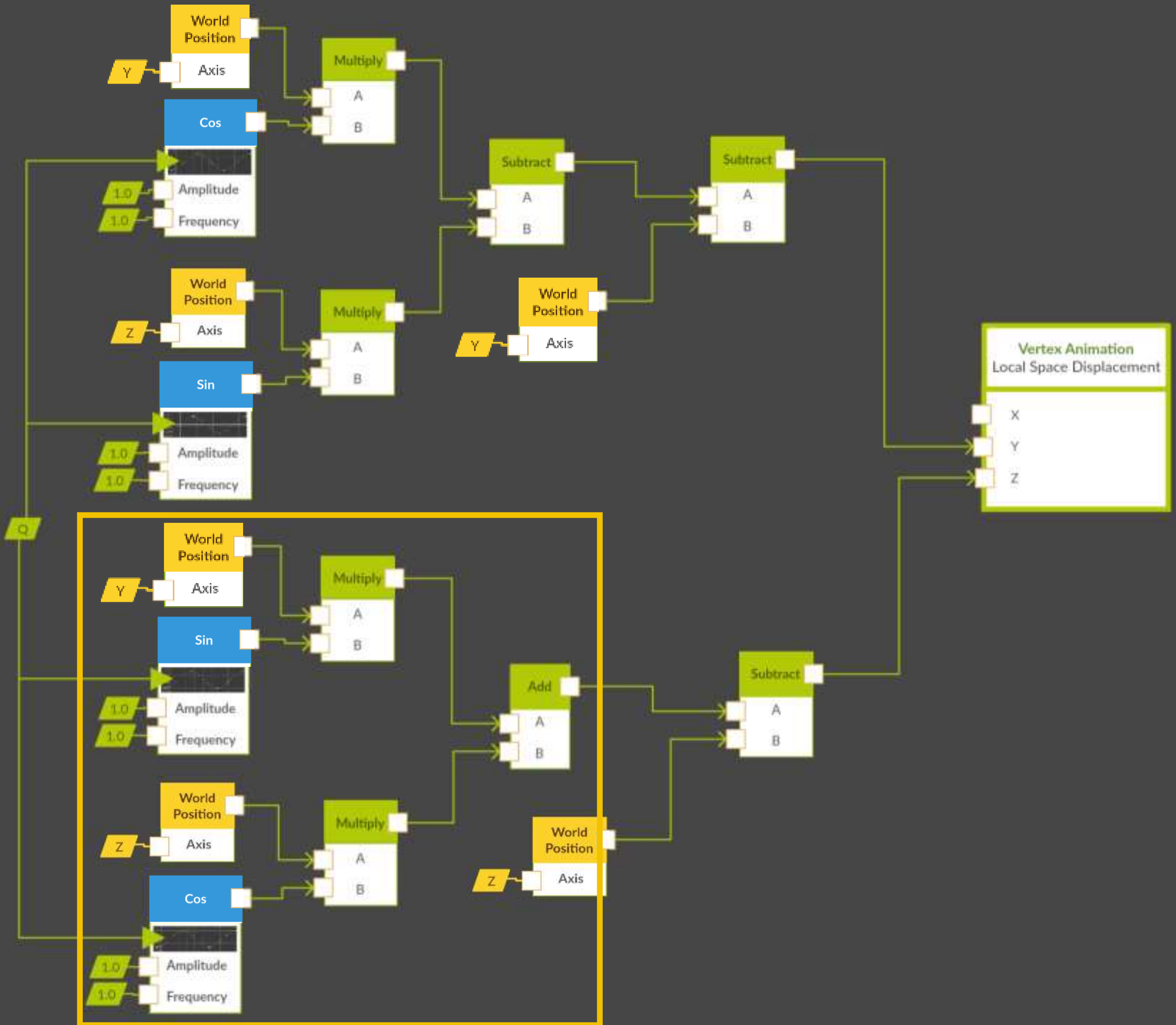


$y' = \boxed{y*\cos q - z*\sin q}$

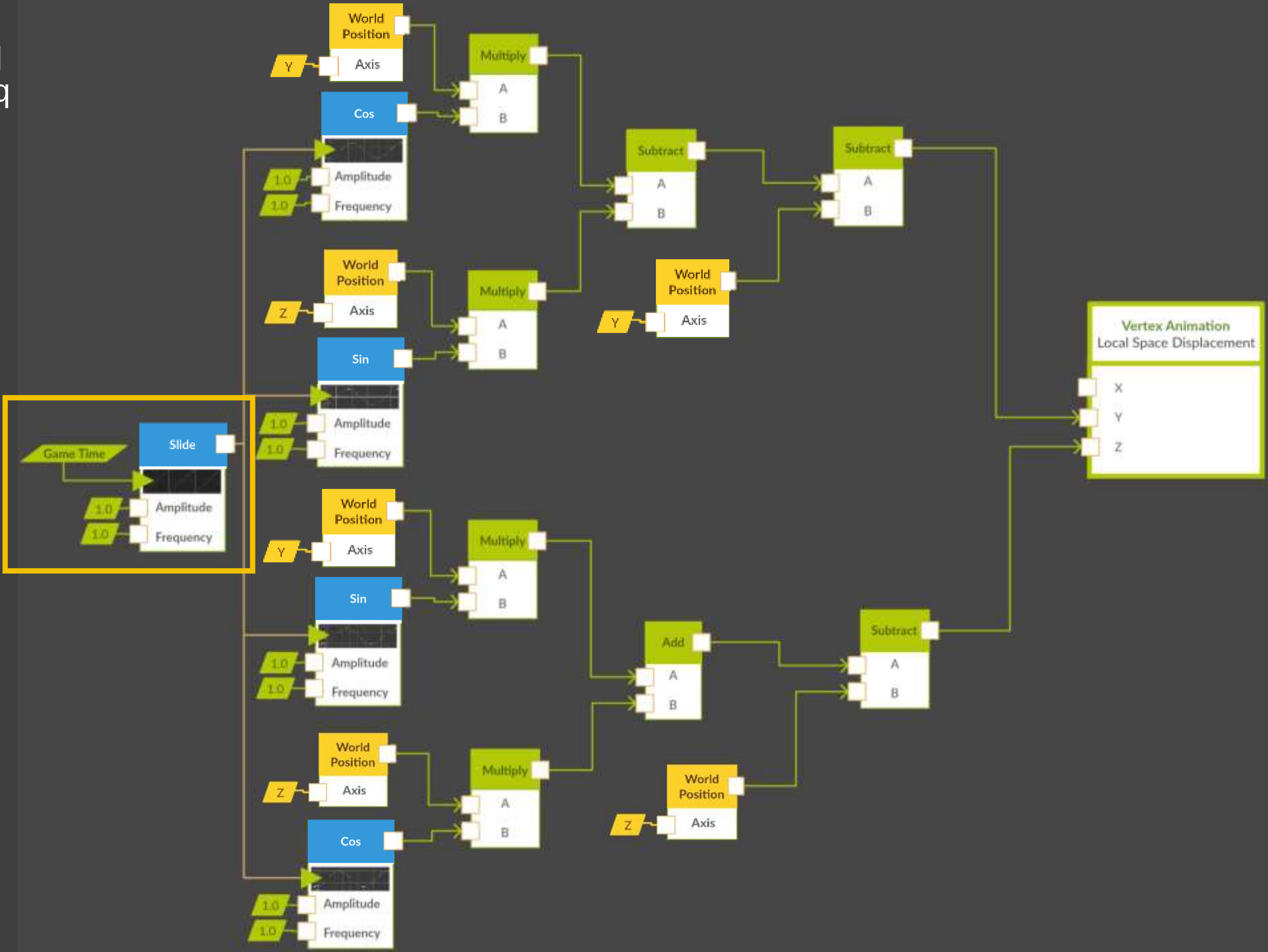
$z' = y*\sin q + z*\cos q$



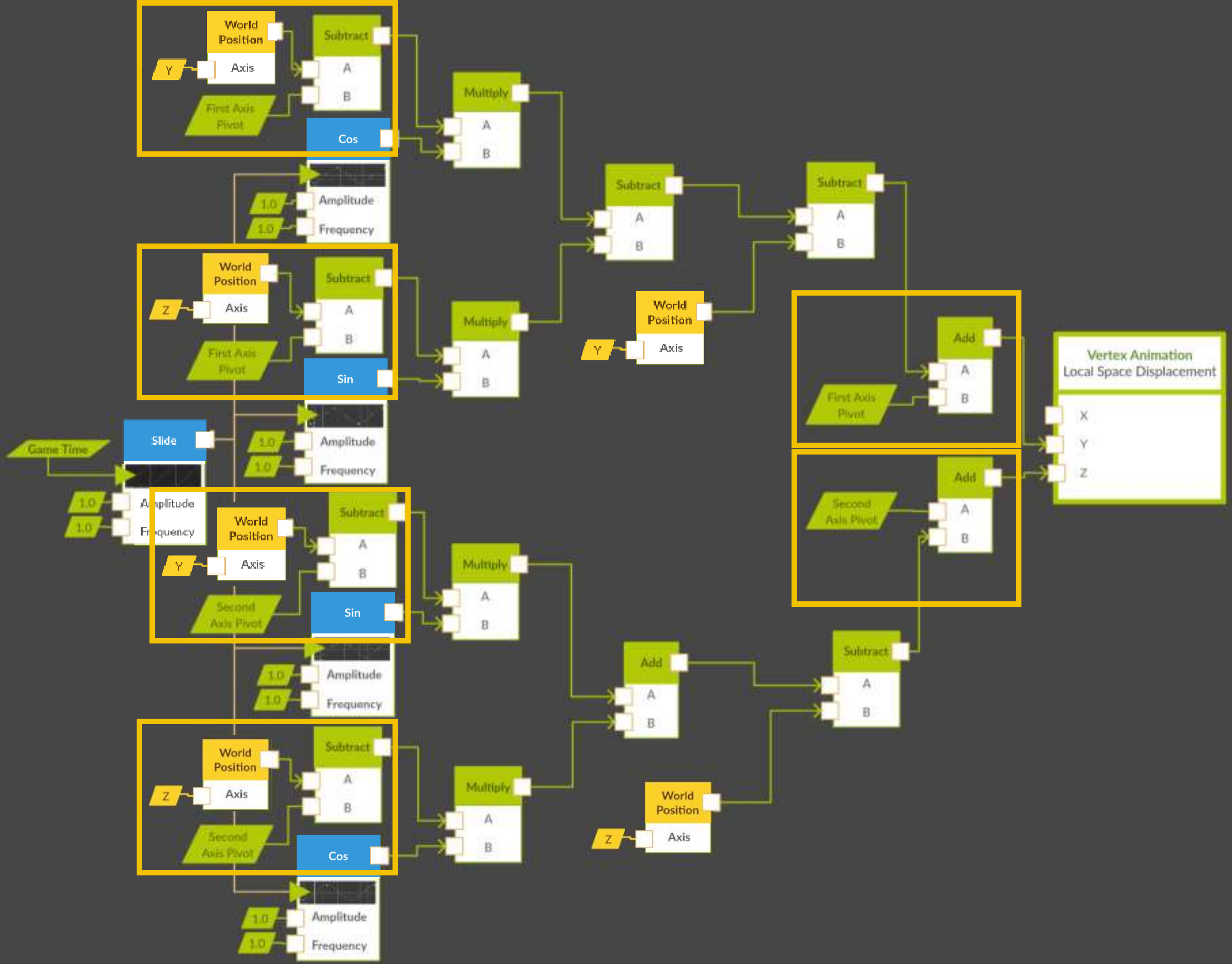
y' = y*cos q - z*sin q
z' = y*sin q + z*cos q



y' = y*cos q - z*sin q
z' = y*sin q + z*cos q



BUNGiE


World Position
Axis
Y
First Axis Pivot
Subtract
A
B
Cos
Amplitude
Frequency
1.0
1.0
Multiply
A
B
World Position
Axis
Z
First Axis Pivot
Subtract
A
B
Sin
Amplitude
Frequency
1.0
1.0
Multiply
A
B
Subtract
A
B
World Position
Axis
Y
Subtract
A
B
Game Time
Slide
Amplitude
Frequency
1.0
1.0
World Position
Axis
Y
Second Axis Pivot
Subtract
A
B
Sin
Amplitude
Frequency
1.0
1.0
Multiply
A
B
World Position
Axis
Z
Second Axis Pivot
Subtract
A
B
Cos
Amplitude
Frequency
1.0
1.0
Multiply
A
B
Add
A
B
World Position
Axis
Z
Subtract
A
B
Add
A
B
First Axis Pivot
Add
A
B
Second Axis Pivot
Vertex Animation
Local Space Displacement
X
Y
Z

BUNGIE
DESTINY

BUNGIE
DESTINY

Supporting Tools

Shader node editor

Templating

Custom channel tools

Vertex color tools

Questions?