RENDERING MILLIONS OF GRASS BLADES

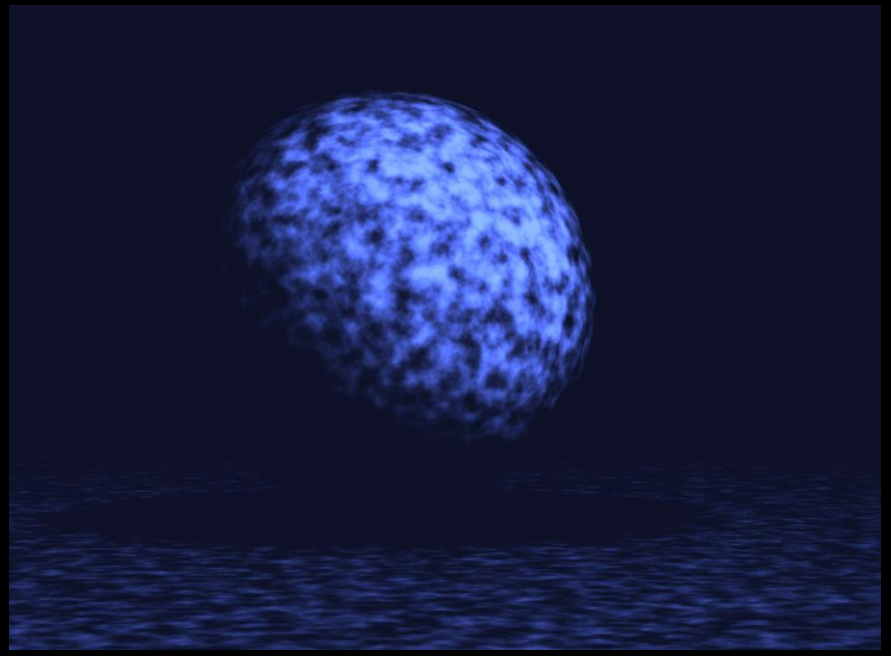
USING UE AND NIAGARA

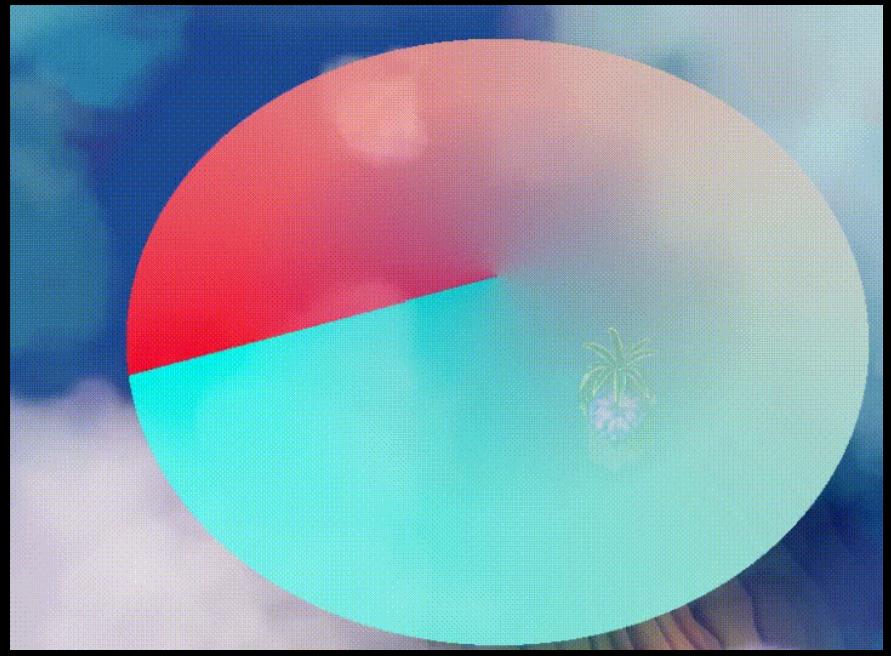
Radosław Paszkowski

PixelAnt GAMES

A SUMO DIGITAL STUDIO

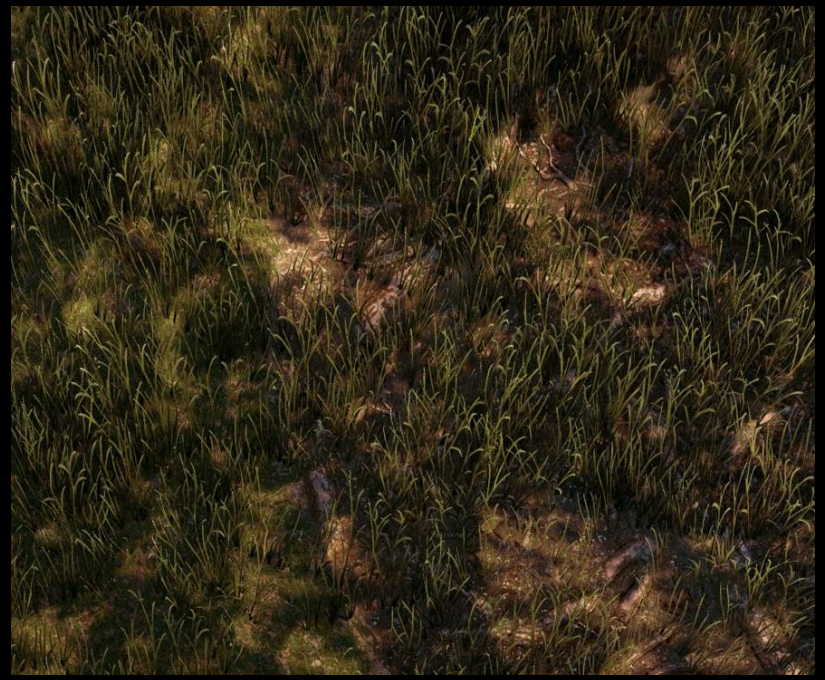
RADEK PASZKOWSKI





RENDERING MILLIONS OF GRASS BLADES USING UE AND NIAGARA

MOST COMMON
MESH CARD GRASS



Credits: Horizon Zero Dawn by Guerrilla Games

Credits: Elden Ring by FromSoftware

THE NEW TECHNIQUE
PER-BLADE MESH GRASS

Credits: Ghost of Tsushima by Sucker Punch Productions

Credits: Genshin Impact by miHoYo

Credits: Grounded by Obsidian Entertainment

ASSUMPTIONS

3D Opaque Models

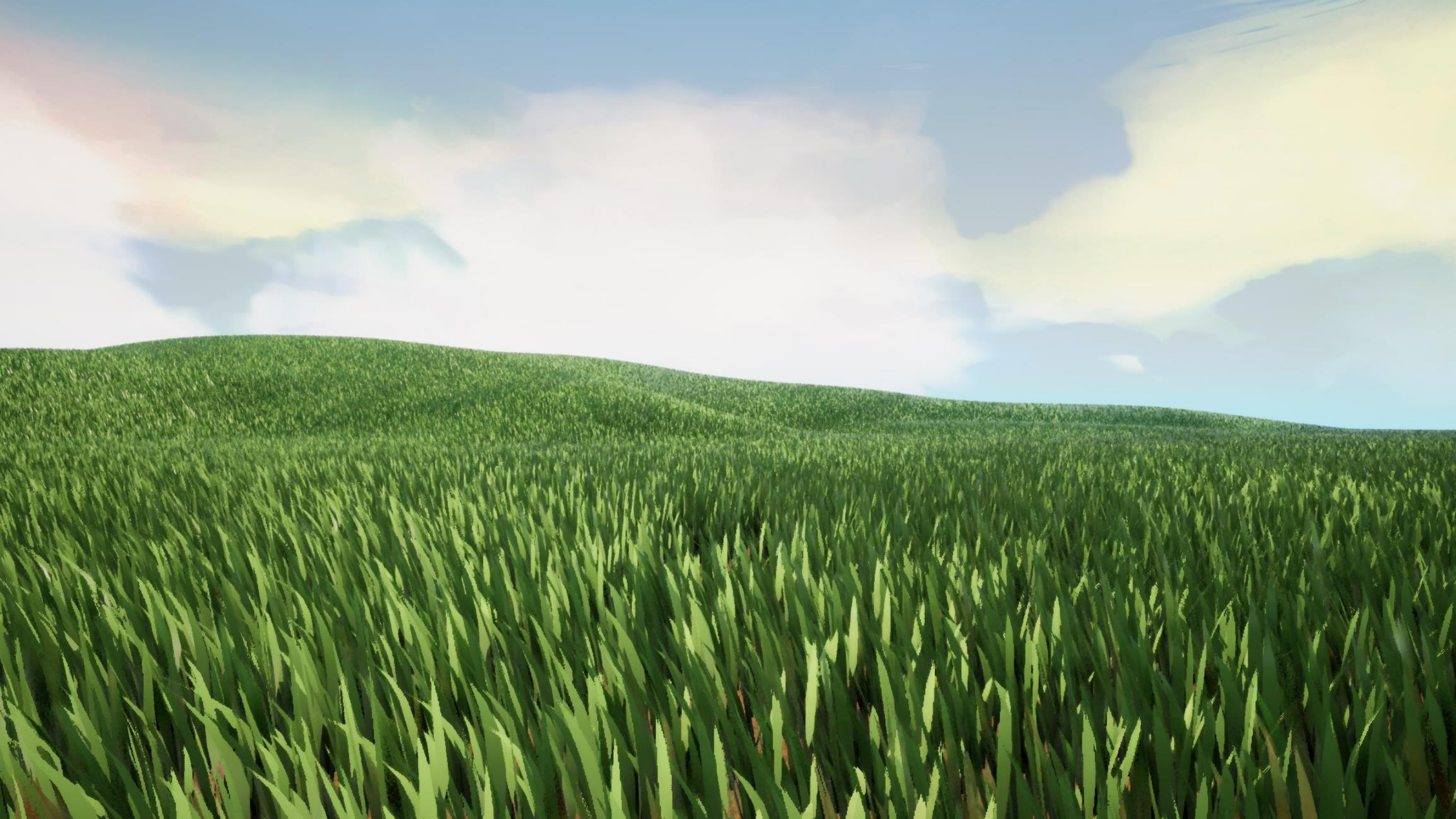
Deterministic

Millions of instances

Artist authored

Interactable

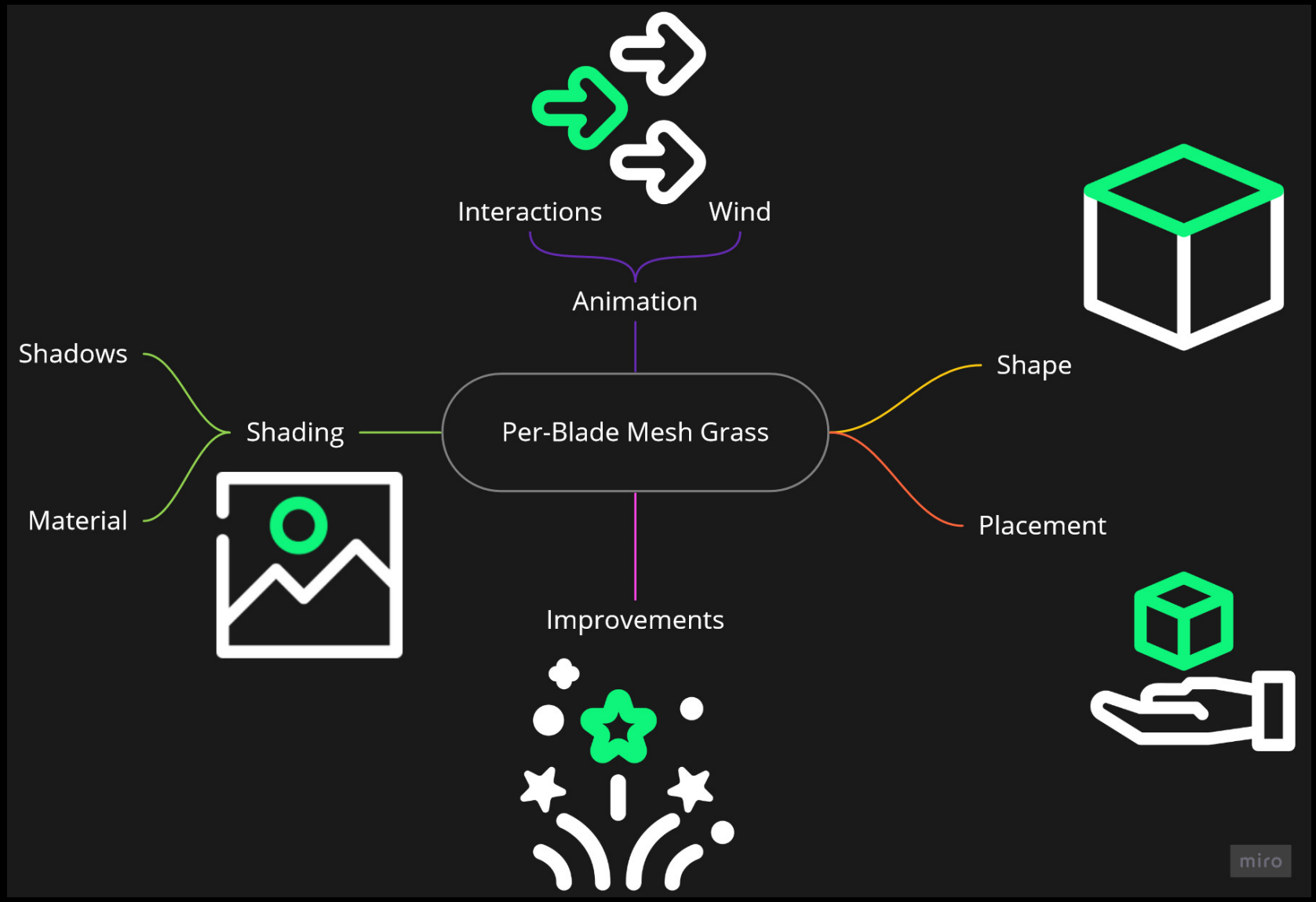
Real-Time



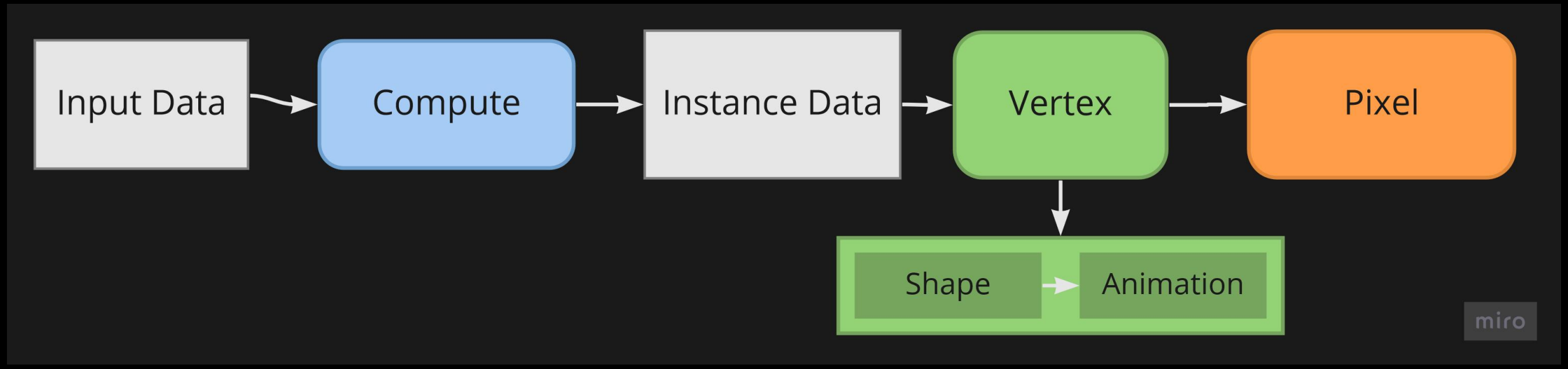
Concreteosis – Betonoza

Credits: Włocławek Old Town Square (Old) / JFZ ^^ / Creative Commons

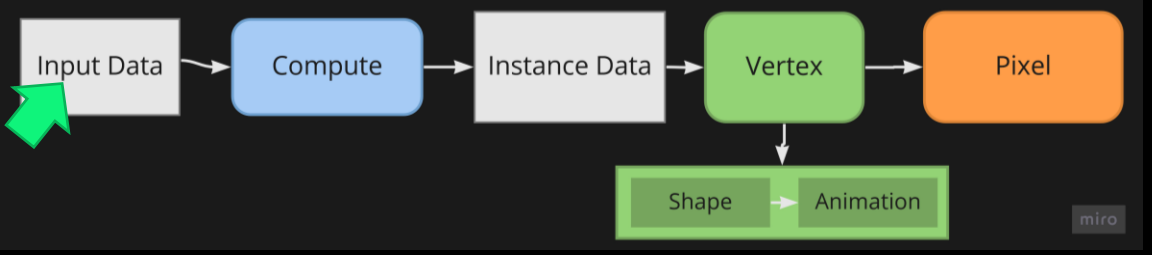
Credits: Włocławek Old Town Square (New) / Krzysztof Wojasiński / Creative Commons



GRASS SYSTEM OVERVIEW


Input Data
Compute
Instance Data
Vertex
Pixel
Shape
Animation
miro

PROCESS



INPUT DATA

- HEIGHTMAP
- WIND & INTERACTION TEXTURE
- AREA SIZE
- BLADES PER GRID / DENSITY
- GRASS BLADE MODEL
- GRASS MATERIALS

Wind Texture

GENERATING WIND TEXTURE

$$wind.x = 0.5\,(1 + \cos(noise_{xy}(uv + kt)))$$

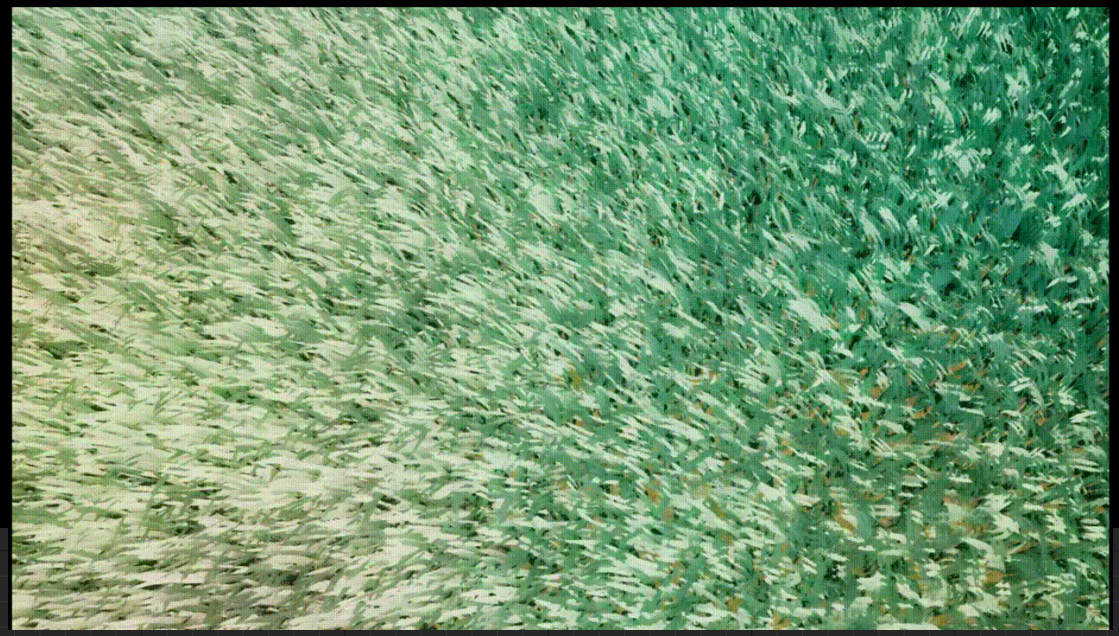
$$wind.y = 0.5\,(1 + \sin(noise_{xy}(uv + kt)))$$

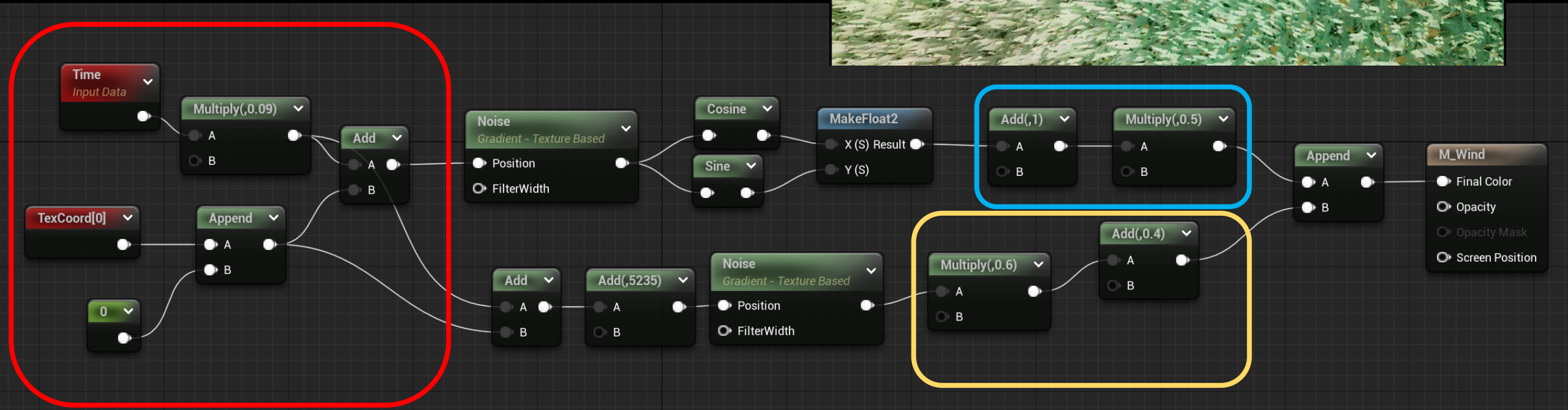
$$wind.z = 0.4 + 0.6\,noise_z((uv + kt) + 0.5235)$$

Generate random seed from position and time

Change range to [0–1]

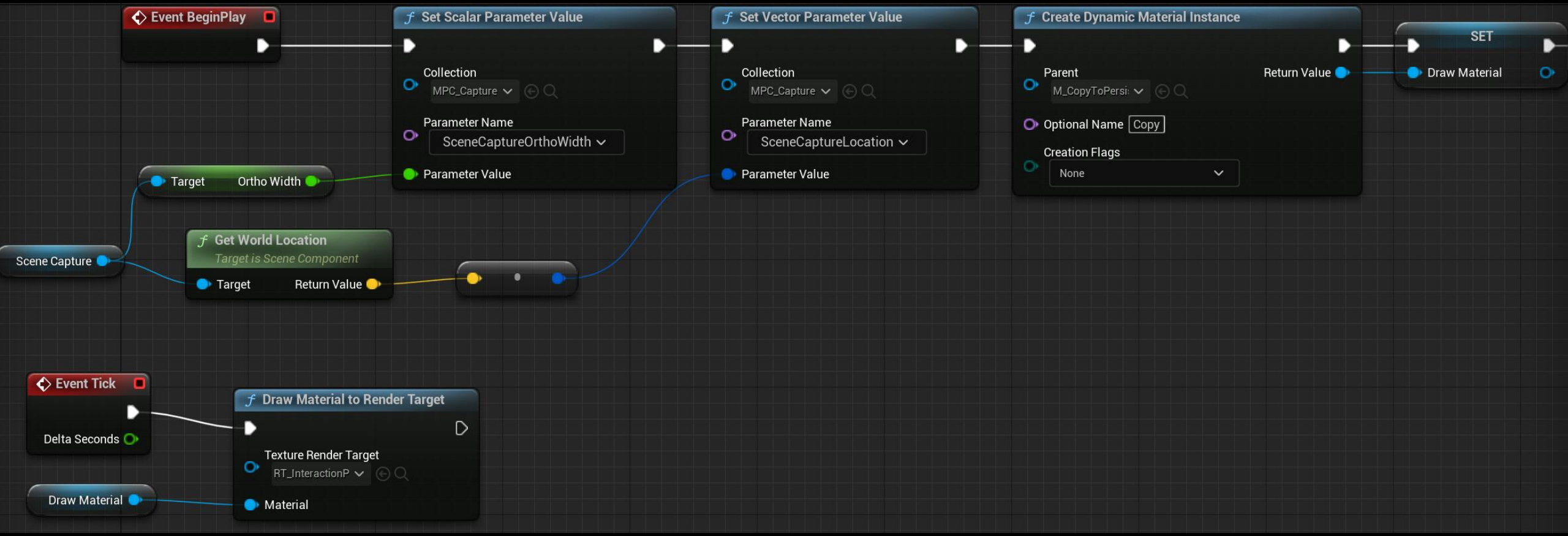
Change range to [0.4–1]



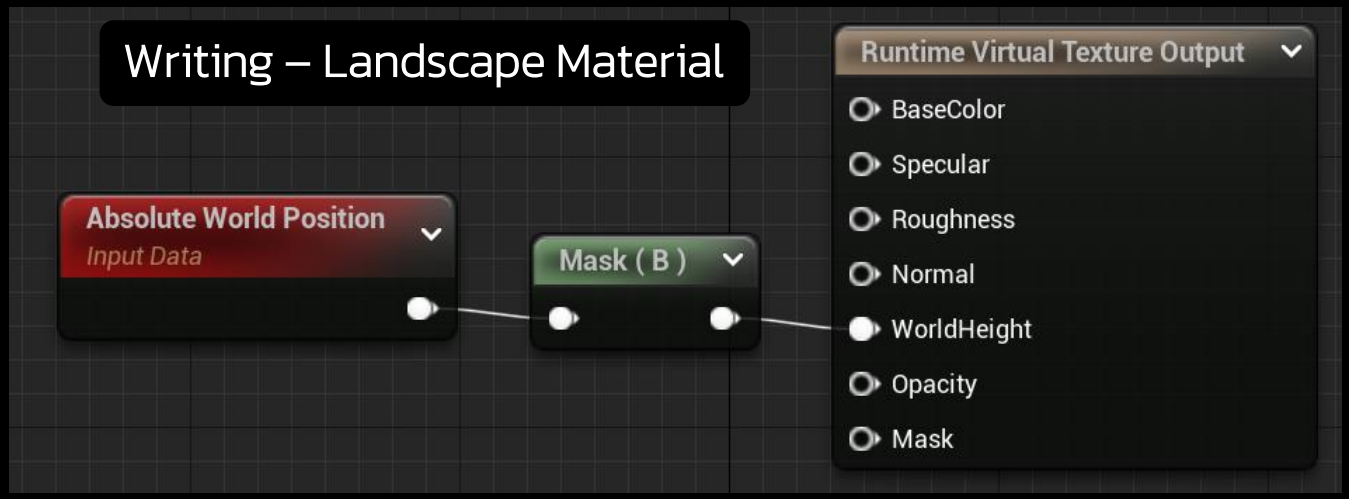


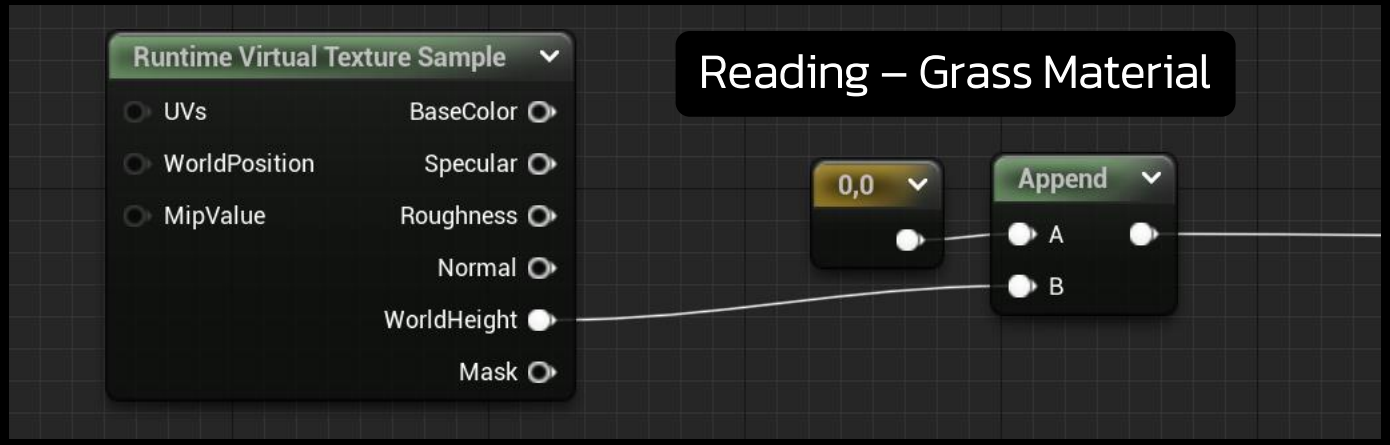
INTERACTION TEXTURE

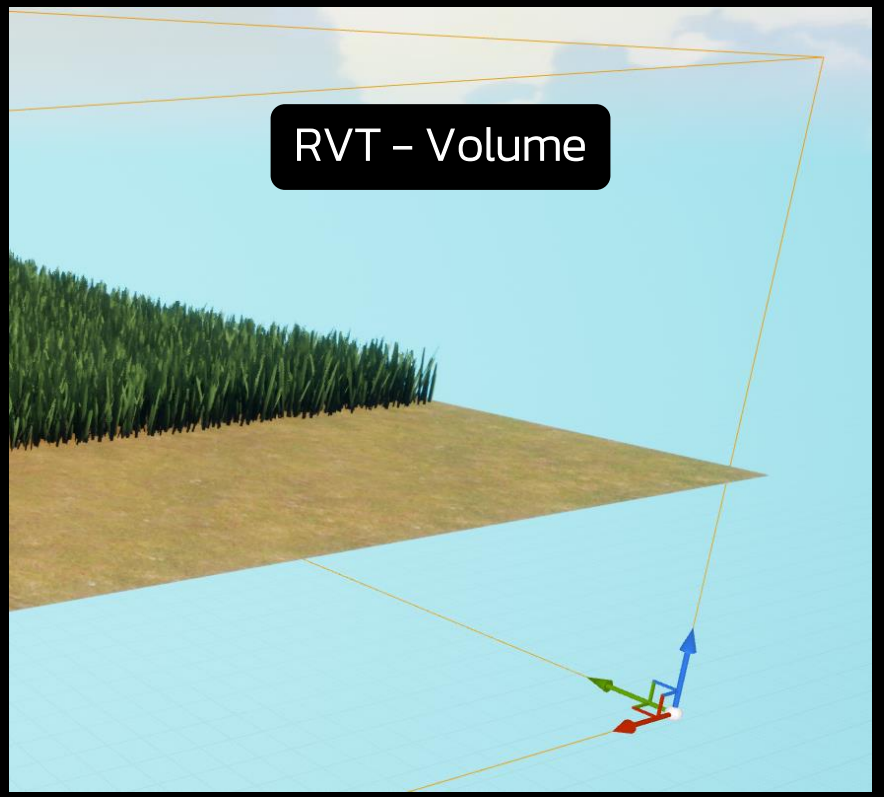
INTERACTION TEXTURE

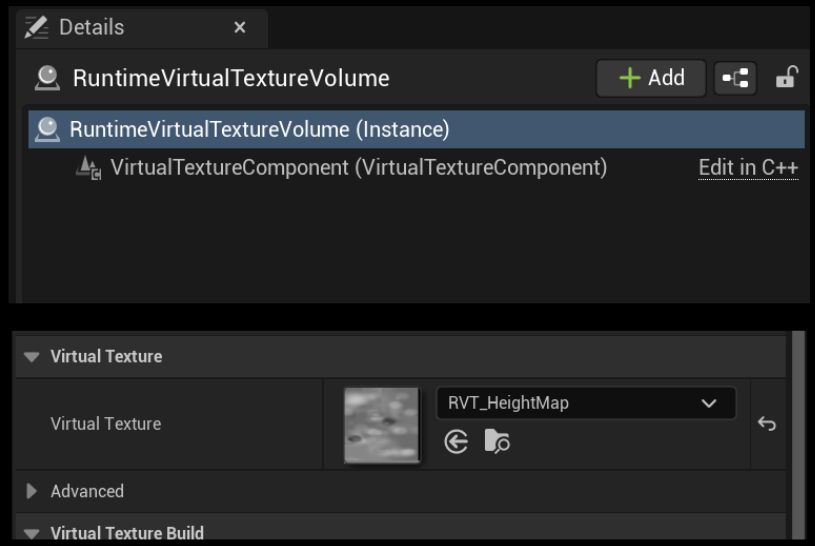


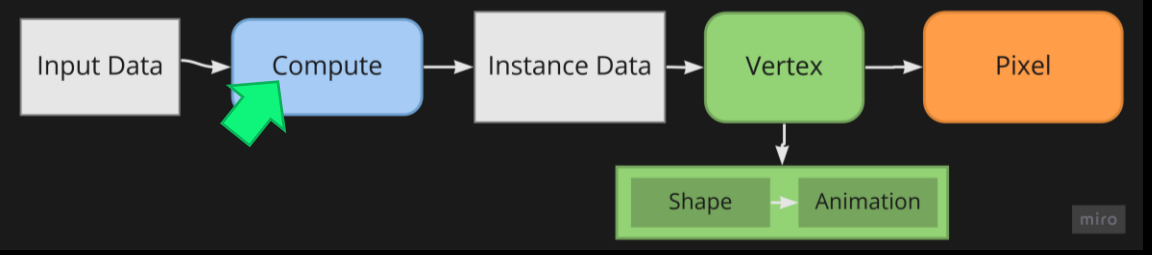
HEIGHTMAP USING RVT





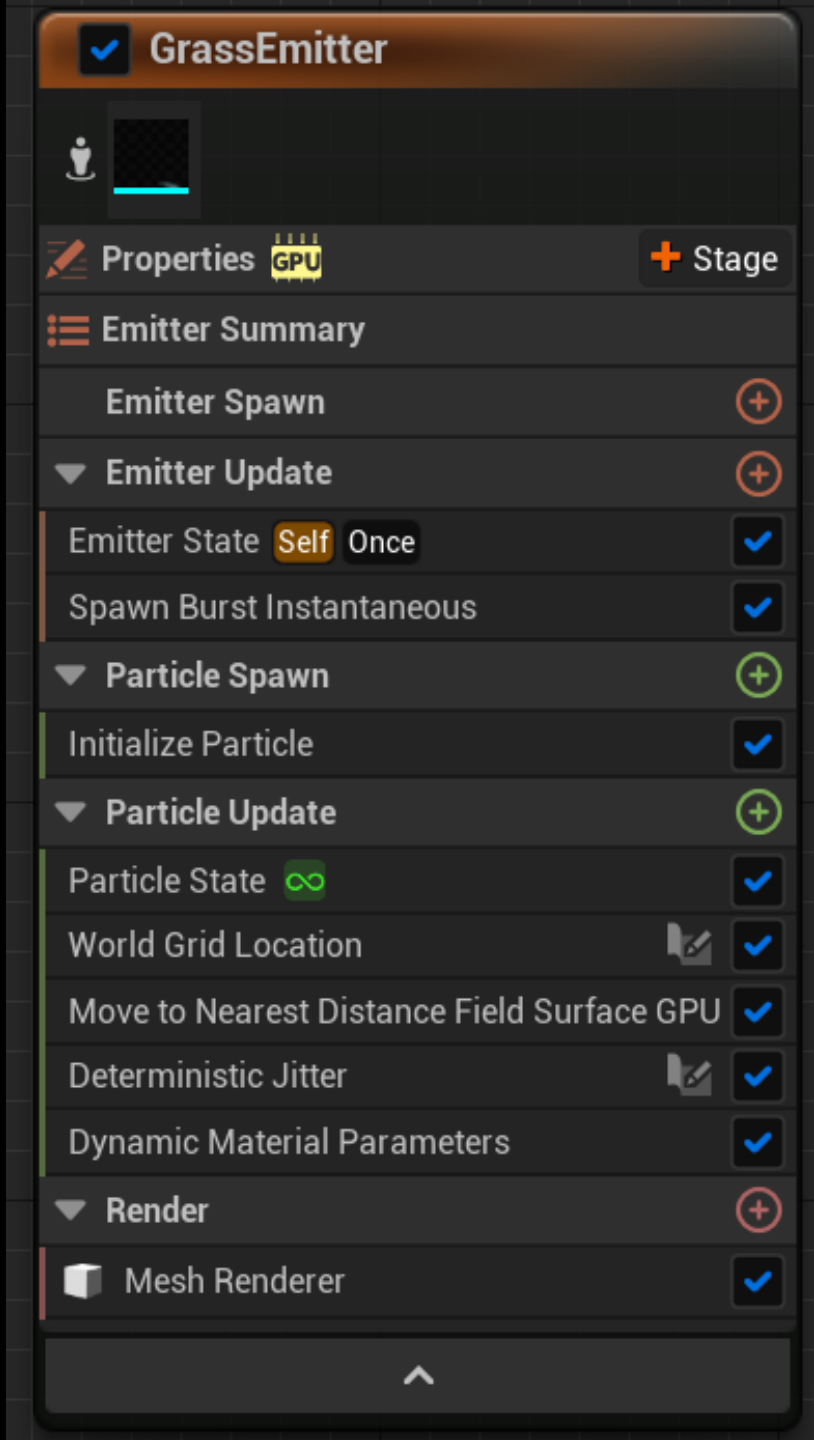


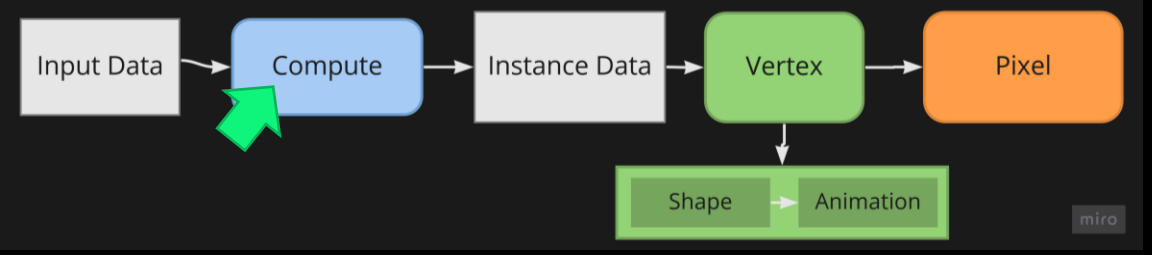




COMPUTE – NIAGARA

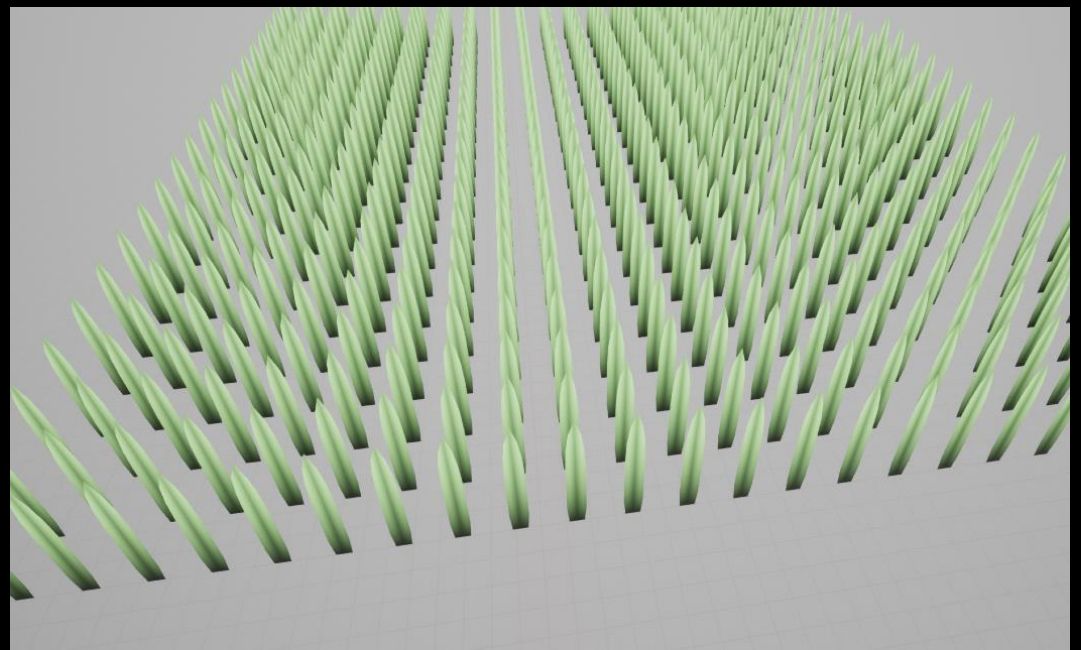
- GRASS GRID POSITION
- FACING DIRECTION
- BÉZIER CURVE PARAMETERS

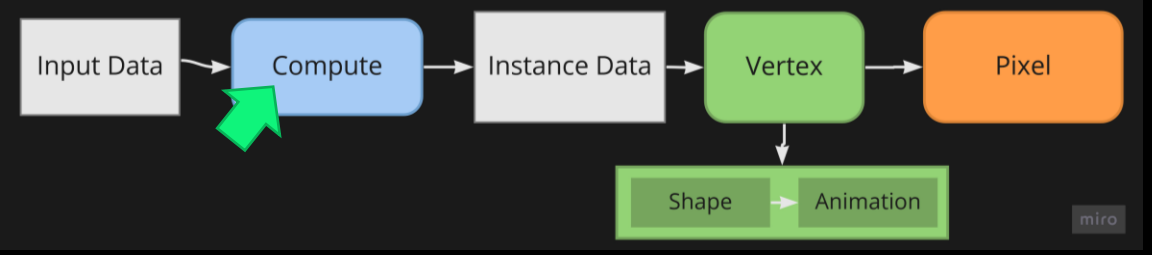




NIAGARA PLACEMENT CALCULATION

- GRID SIZE
- SPAWN COUNT
- UNIQUE PARTICLE ID



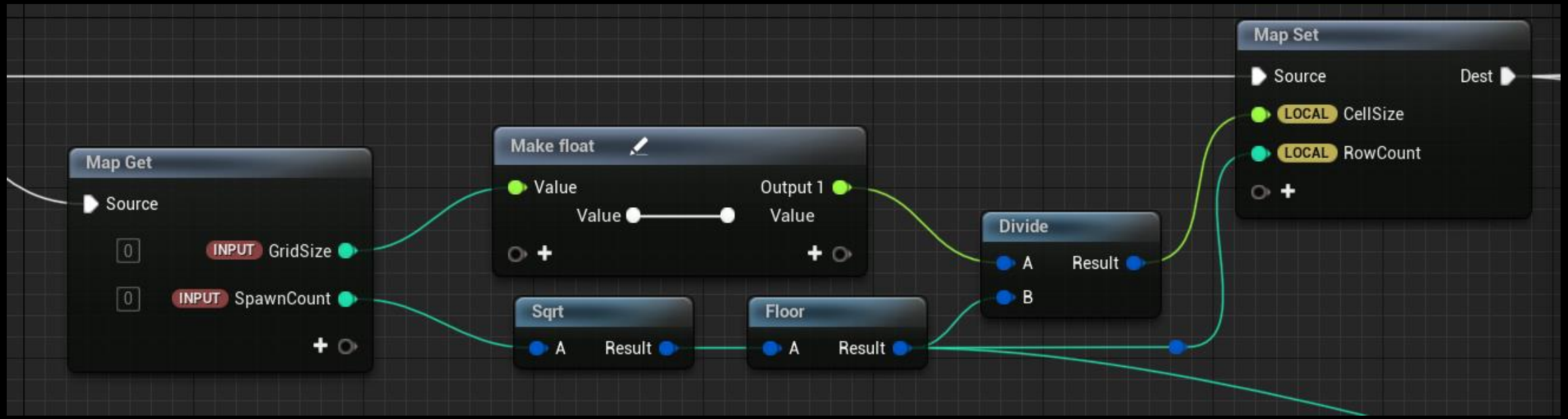


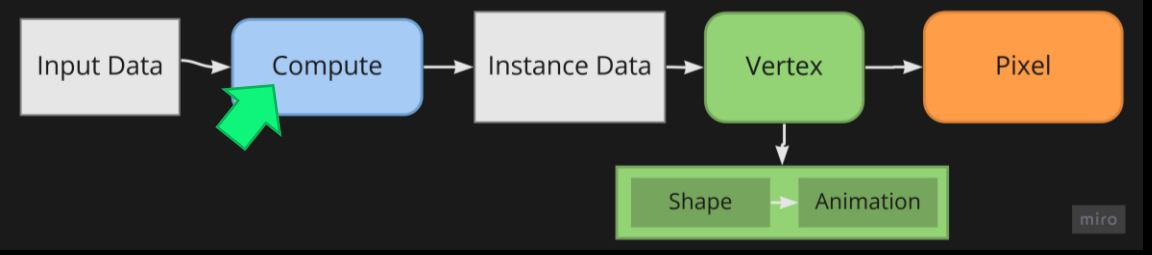
NIAGARA PLACEMENT CALCULATION

- GRID SIZE
- SPAWN COUNT
- UNIQUE PARTICLE ID

$$LineCount\ (Row\&Column) = \left\lfloor \sqrt{SpawnCount} \right\rfloor$$

$$Cell\ Size = \frac{GridSize}{LineCount}$$





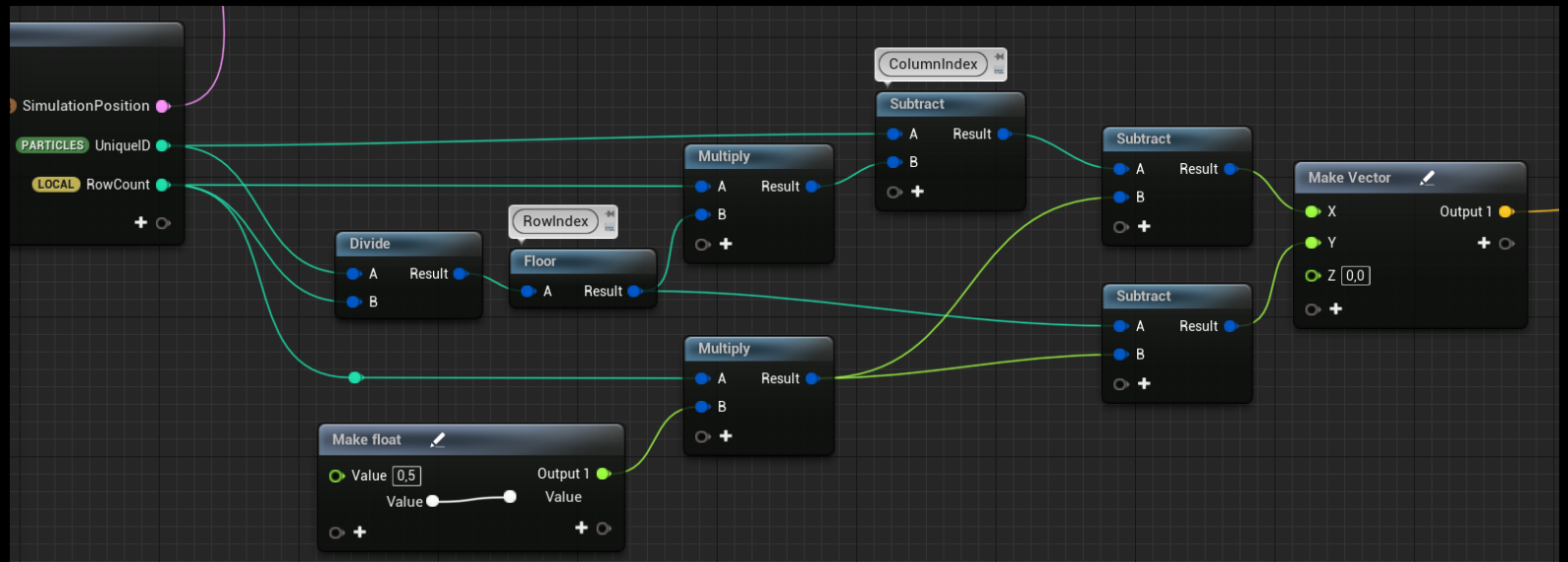
NIAGARA PLACEMENT CALCULATION

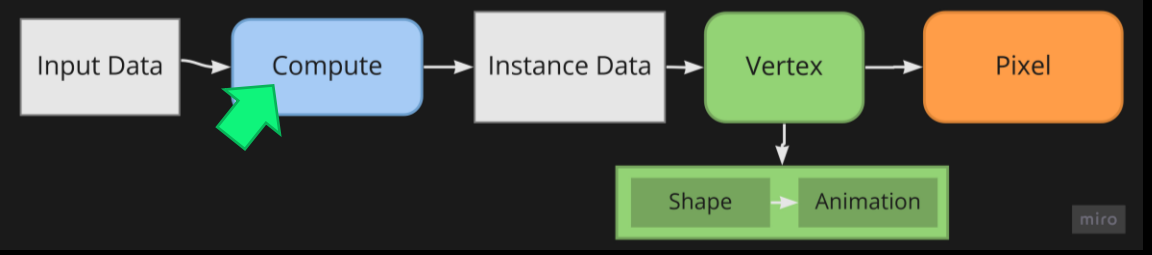
$$Row\ Index = \left\lfloor \frac{ParticleID}{LineCount} \right\rfloor$$

$$Column\ Index = ParticleID - RowIndex * LineCount$$

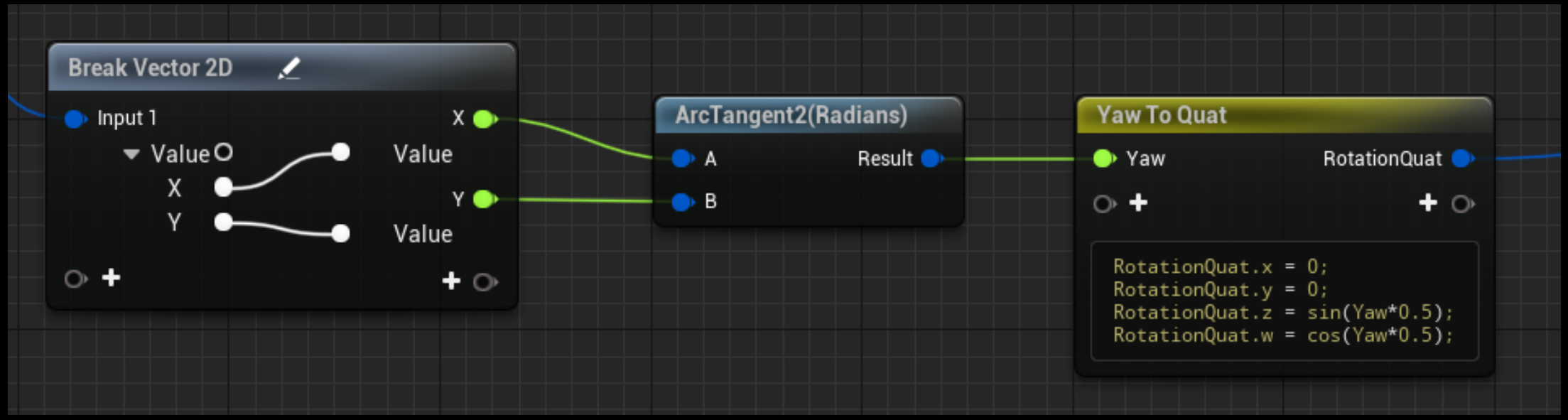
$$Position = CellSize * (\mathrm{ColumnIndex}, \mathrm{RowIndex}) - 0.5\mathrm{LineCount}$$

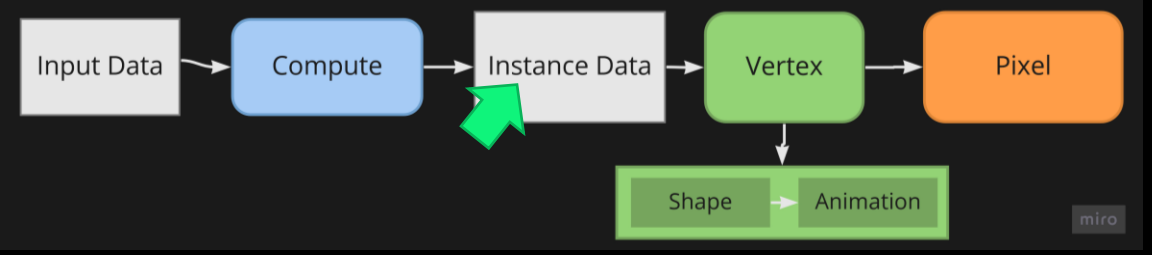
$$BladeHash = Hash(Position)$$





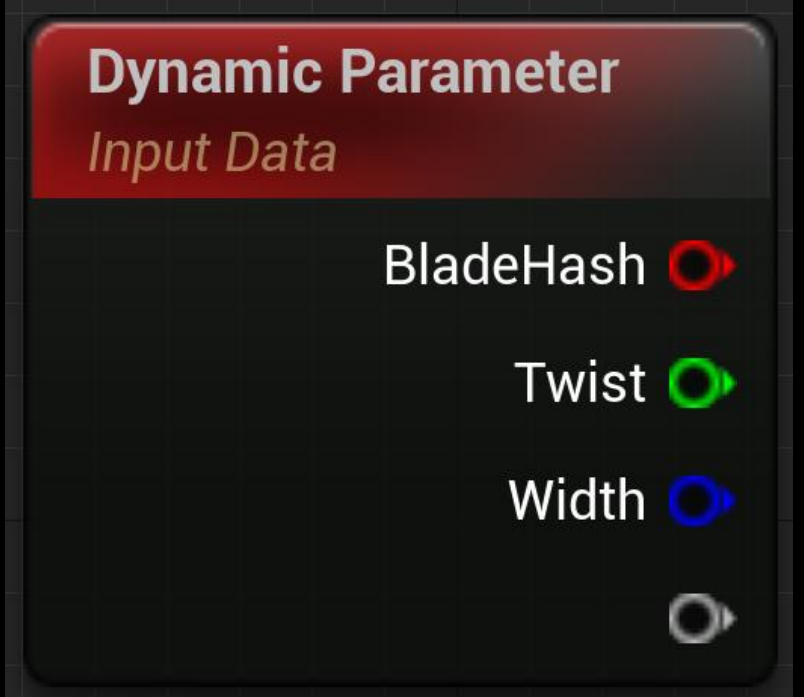
RANDOM ROTATION

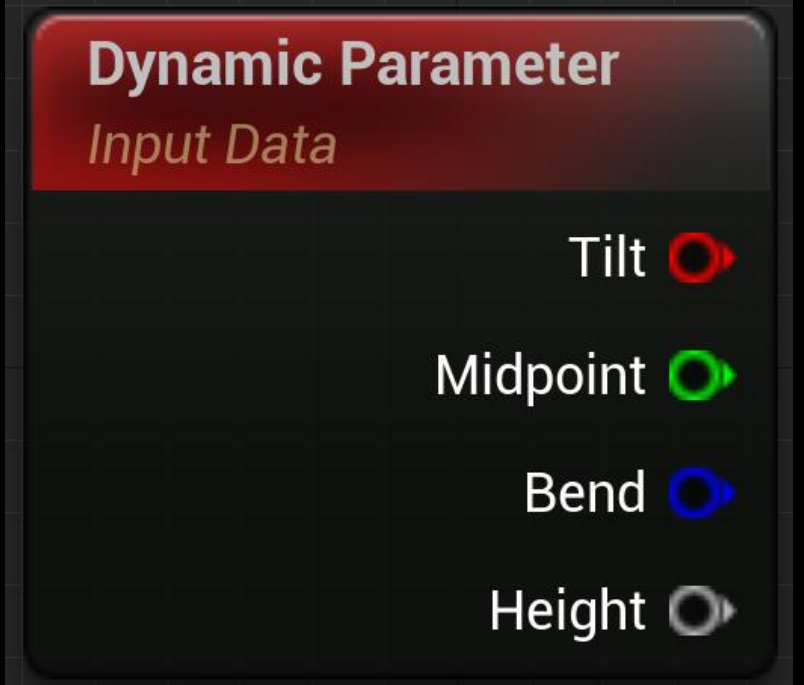


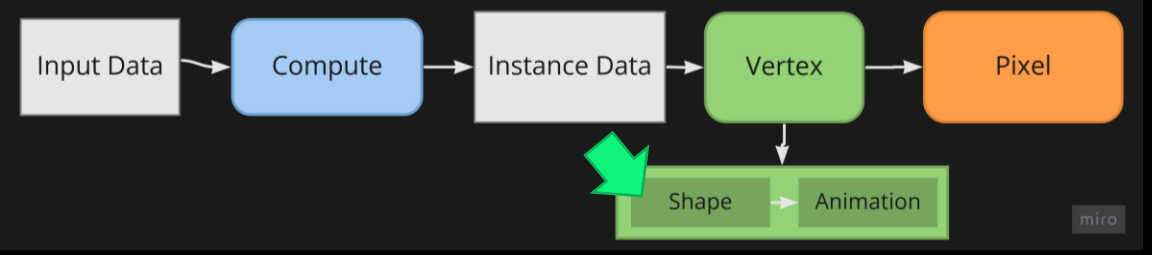


INSTANCE DATA

- 16-FLOAT PER INSTANCE DATA:
  - POSITION – 3 FLOATS
  - FACING – 2 FLOATS
  - PER-BLADE HASH
  - CLUMP FACING – 2 FLOATS
  - HEIGHT
  - WIDTH
  - BÉZIER CURVE – 3 FLOATS
    - TILT
    - BEND
    - MIDPOINT
  - TWIST

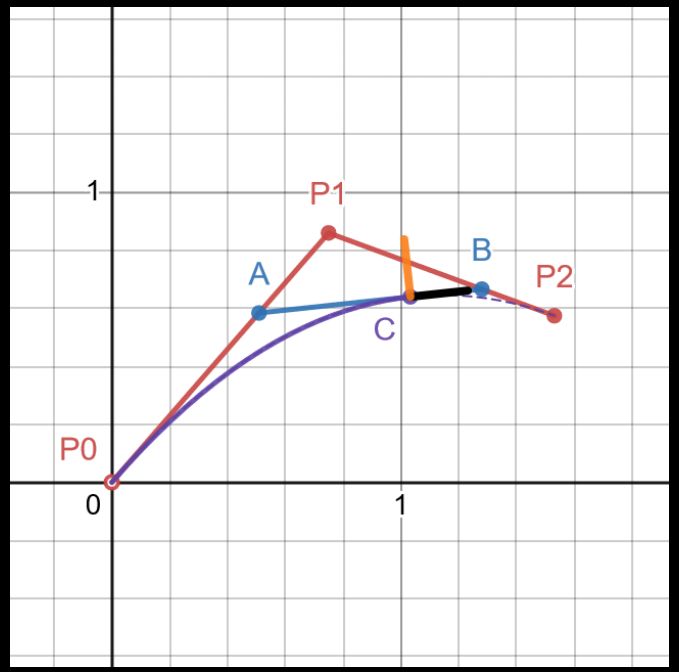




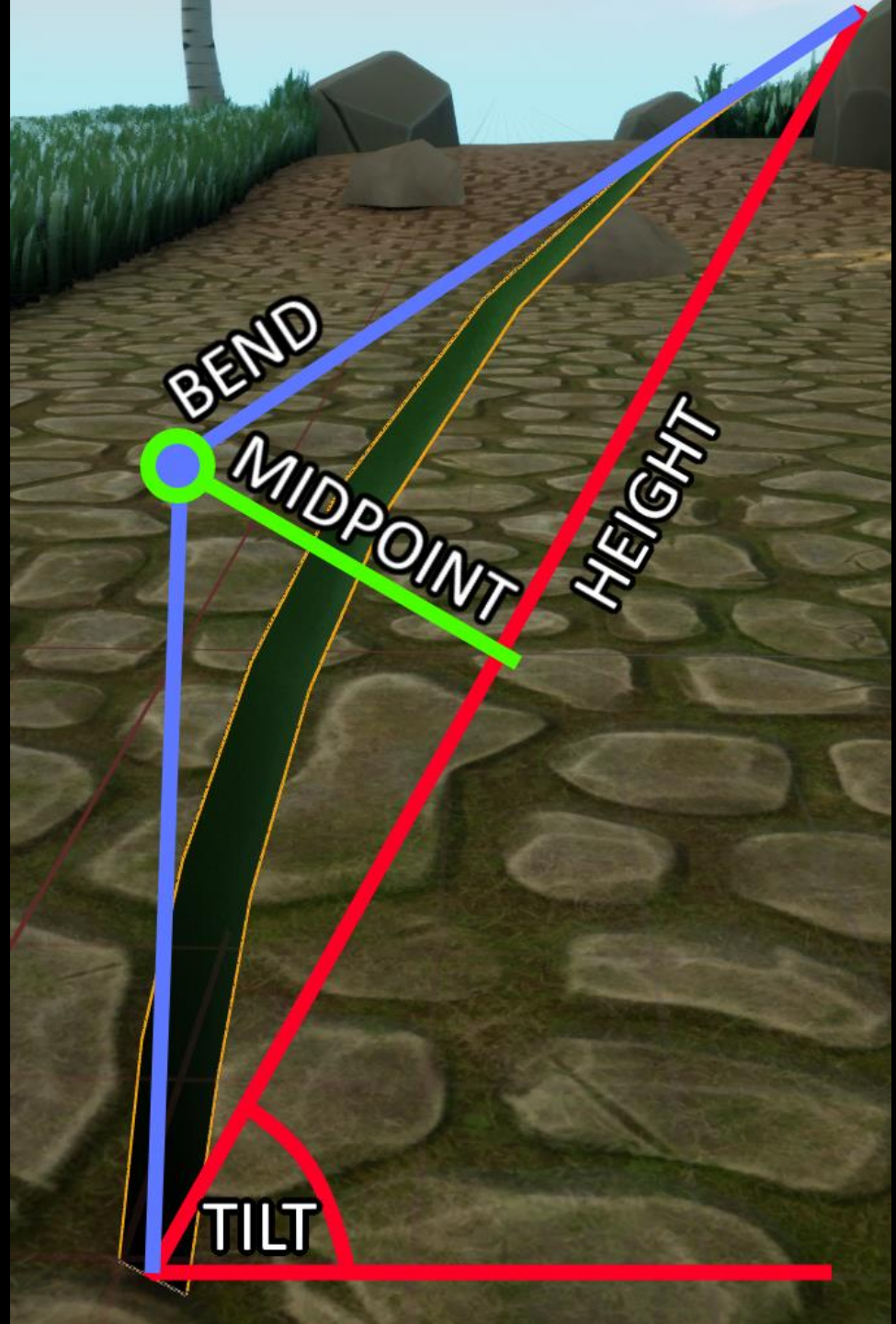


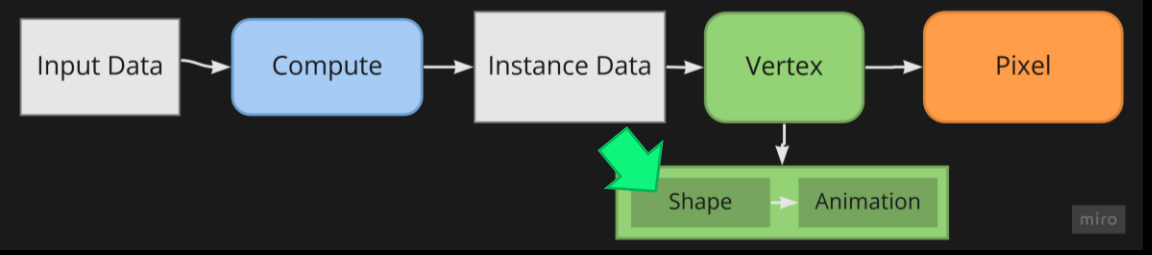
SHAPE – BEZIER CURVE

- Fast and cheap
- Easy to calculate derivative
- Simple to animate



Desmos | Bézier Curve visualisation





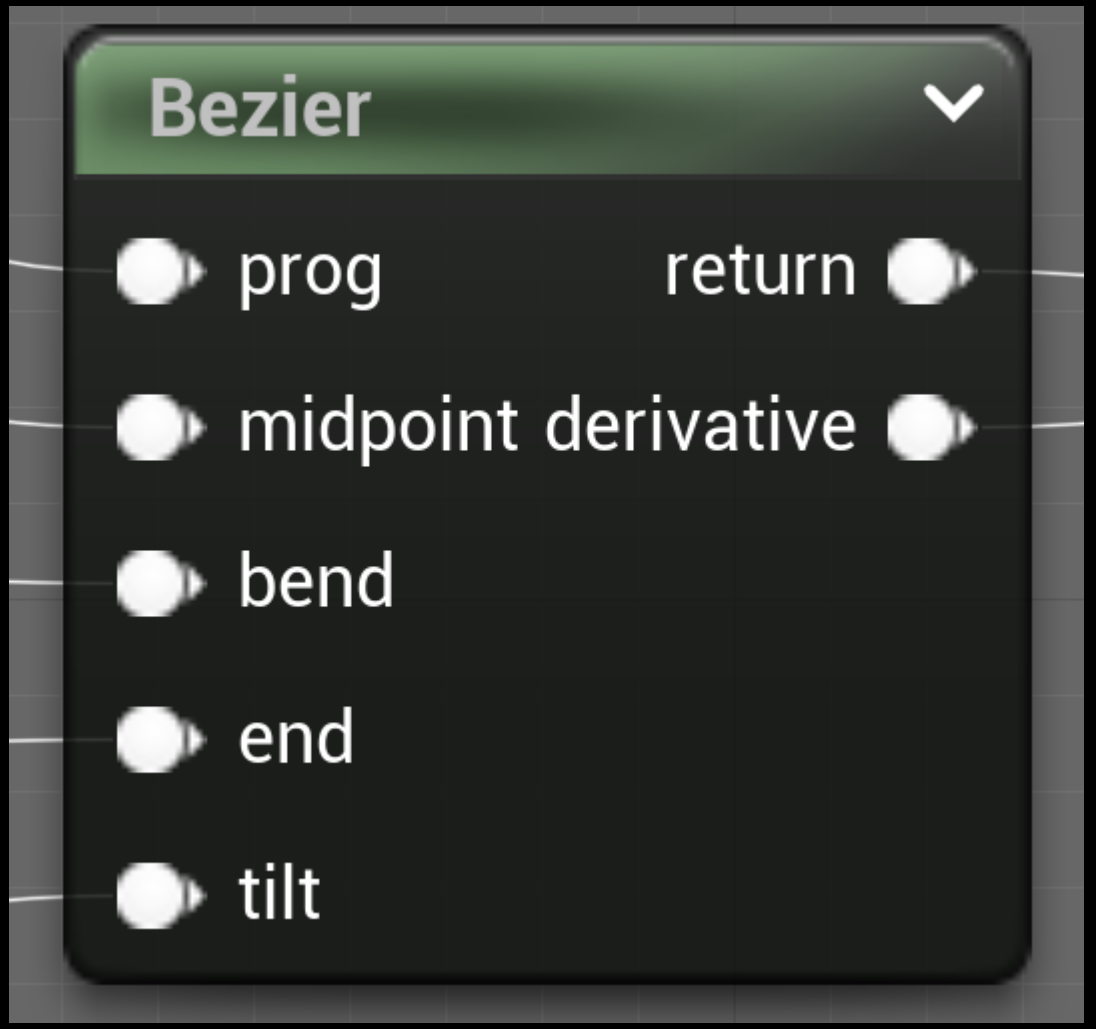
SHAPE – BEZIER CURVE

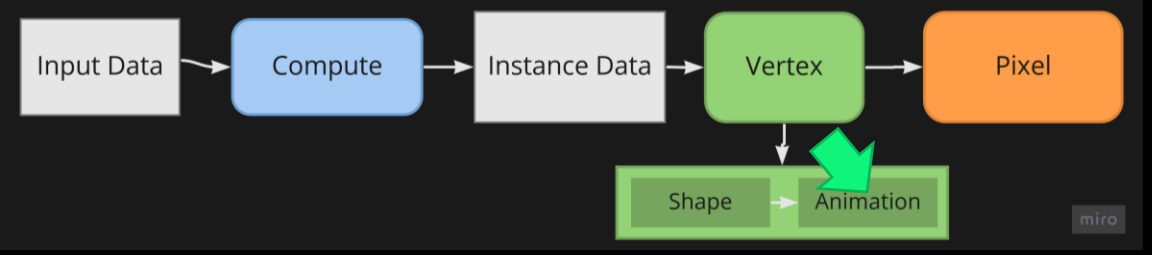
```
mid.x = midpoint.x – sin(tilt) * bend / 2.0;
mid.y = midpoint.y + cos(tilt) * bend / 2.0;

bezier = (2.0 * (1 – prog) * prog) * mid + prog * prog * end;
derivative = 2.0 * (1.0 – 2.0 * prog) * mid + 2.0 * prog * end;
```

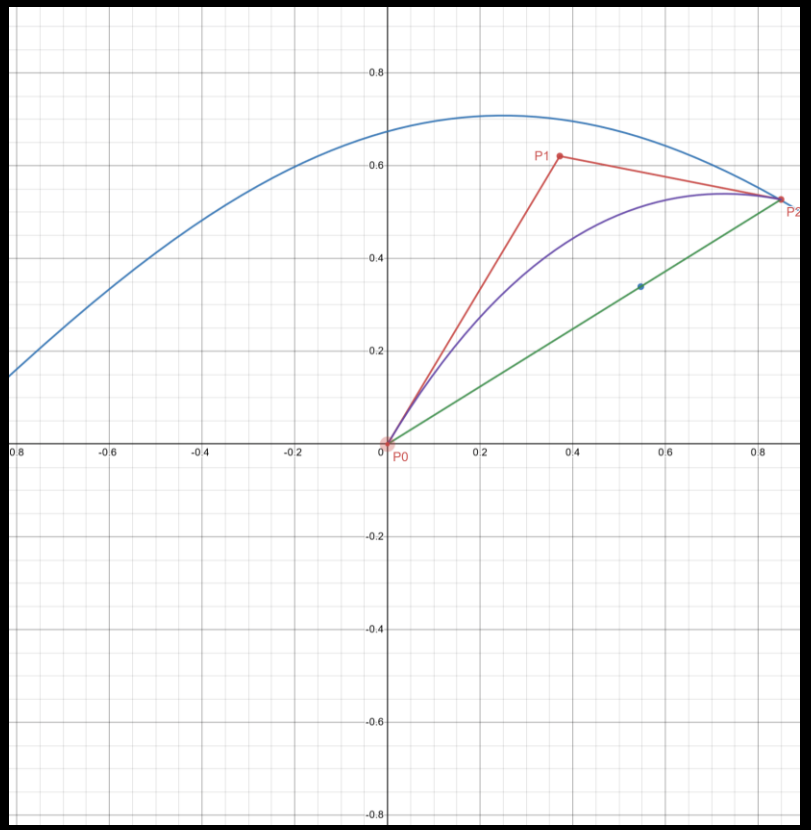
We can omit this as $P_0 = (0, 0)$

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2$$
$$B'(t) = 2(t-1)P_0 + 2(1-2t)P_1 + 2tP_2$$

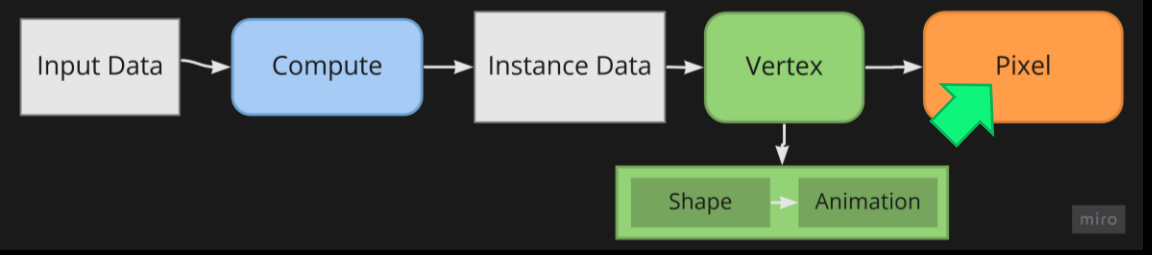




ANIMATION

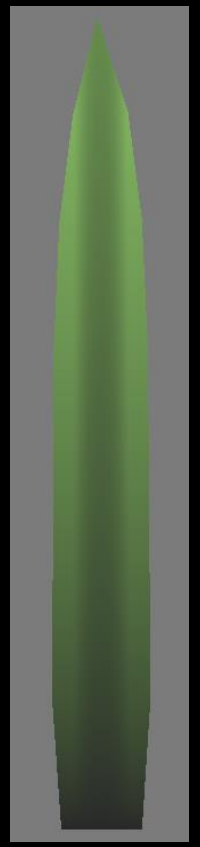


Desmos | Bézier Curve Animation

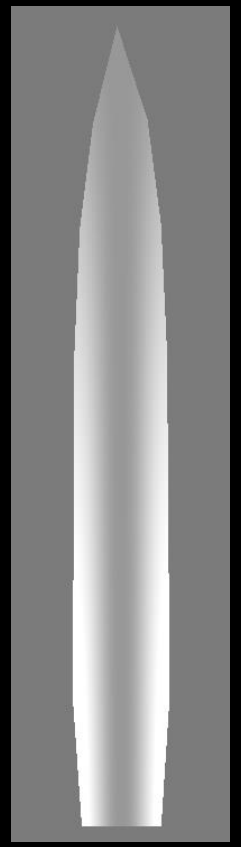


PIXEL

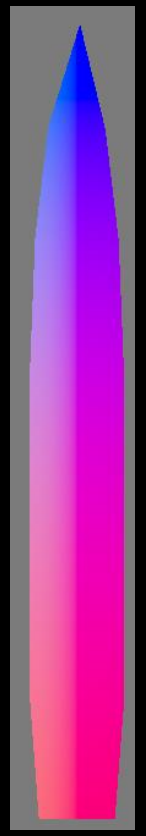
COLOR

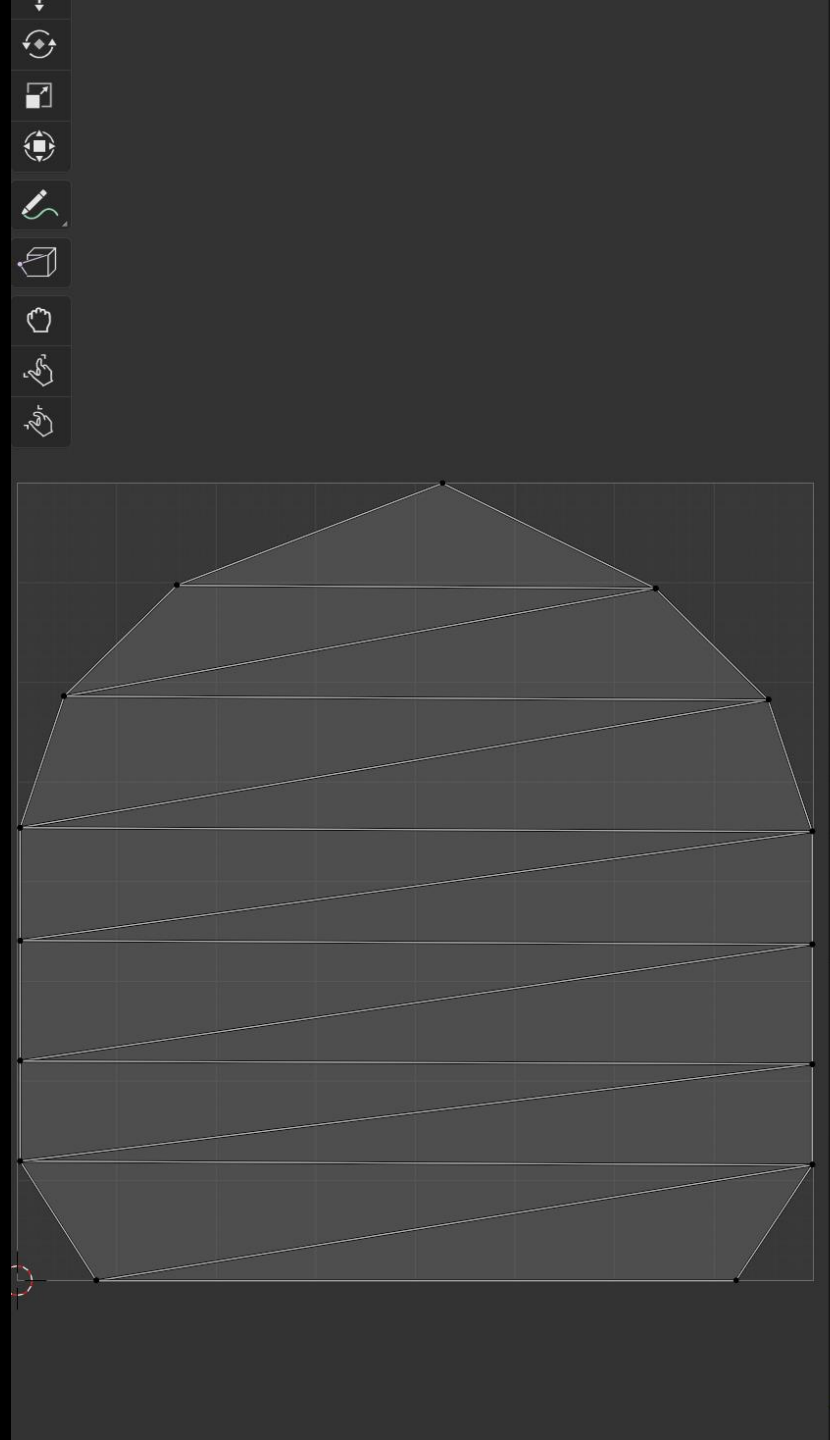


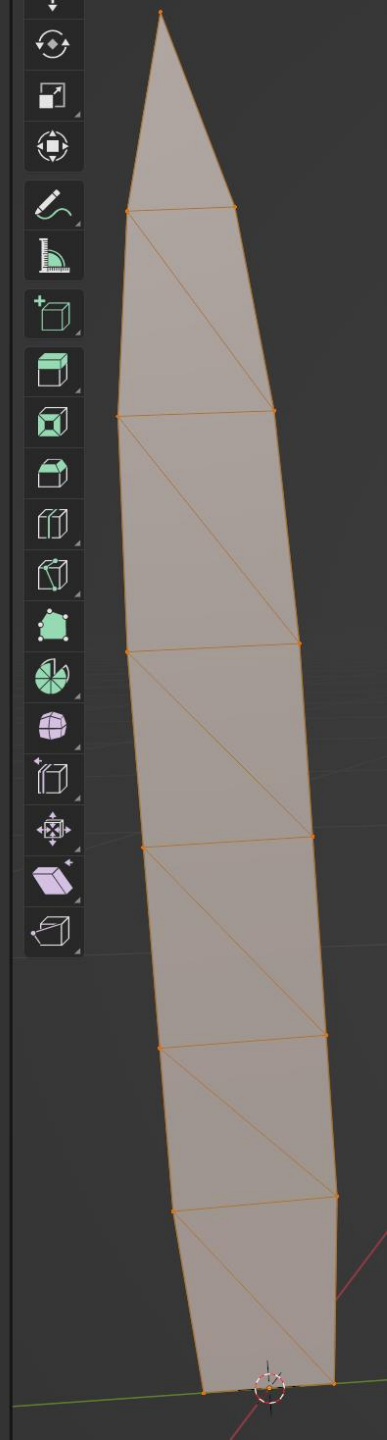
ROUGHNESS (INVERSE GLOSS)

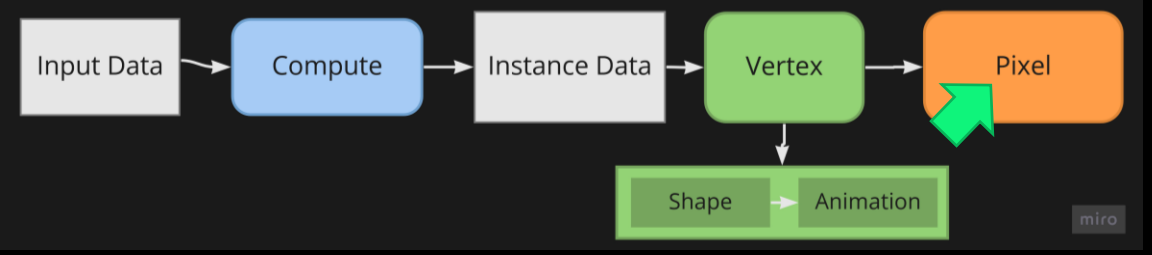


NORMAL

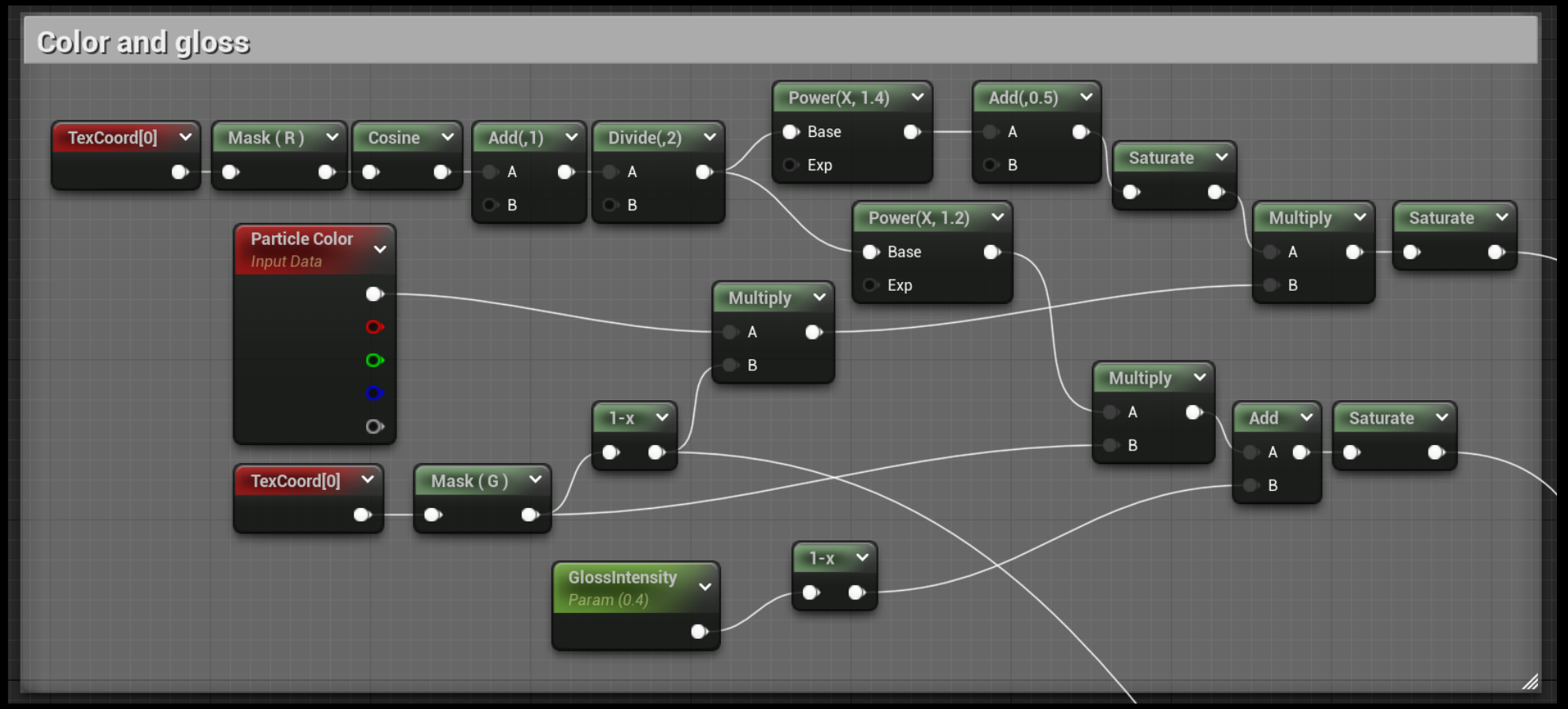


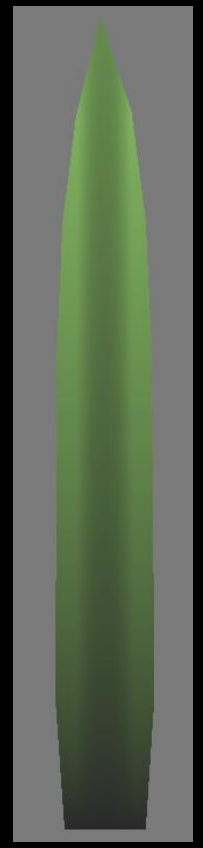


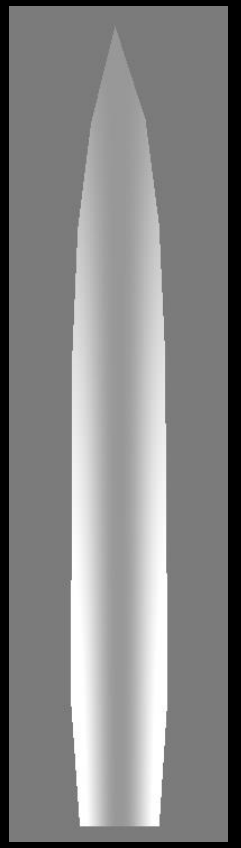



Input Data
Compute
Instance Data
Vertex
Pixel
Shape
Animation
miro

PIXEL


Color and gloss
TexCoord[0]
Mask (R)
Cosine
Add(,1)
A
B
Divide(,2)
A
B
Power(X, 1.4)
Base
Exp
Add(,0.5)
A
B
Saturate
Power(X, 1.2)
Base
Exp
Multiply
A
B
Saturate
Particle Color
Input Data
Multiply
A
B
1-x
Multiply
A
B
Add
A
B
Saturate
TexCoord[0]
Mask (G)
GlossIntensity
Param (0.4)
1-x






LOOK INTO THE PAST

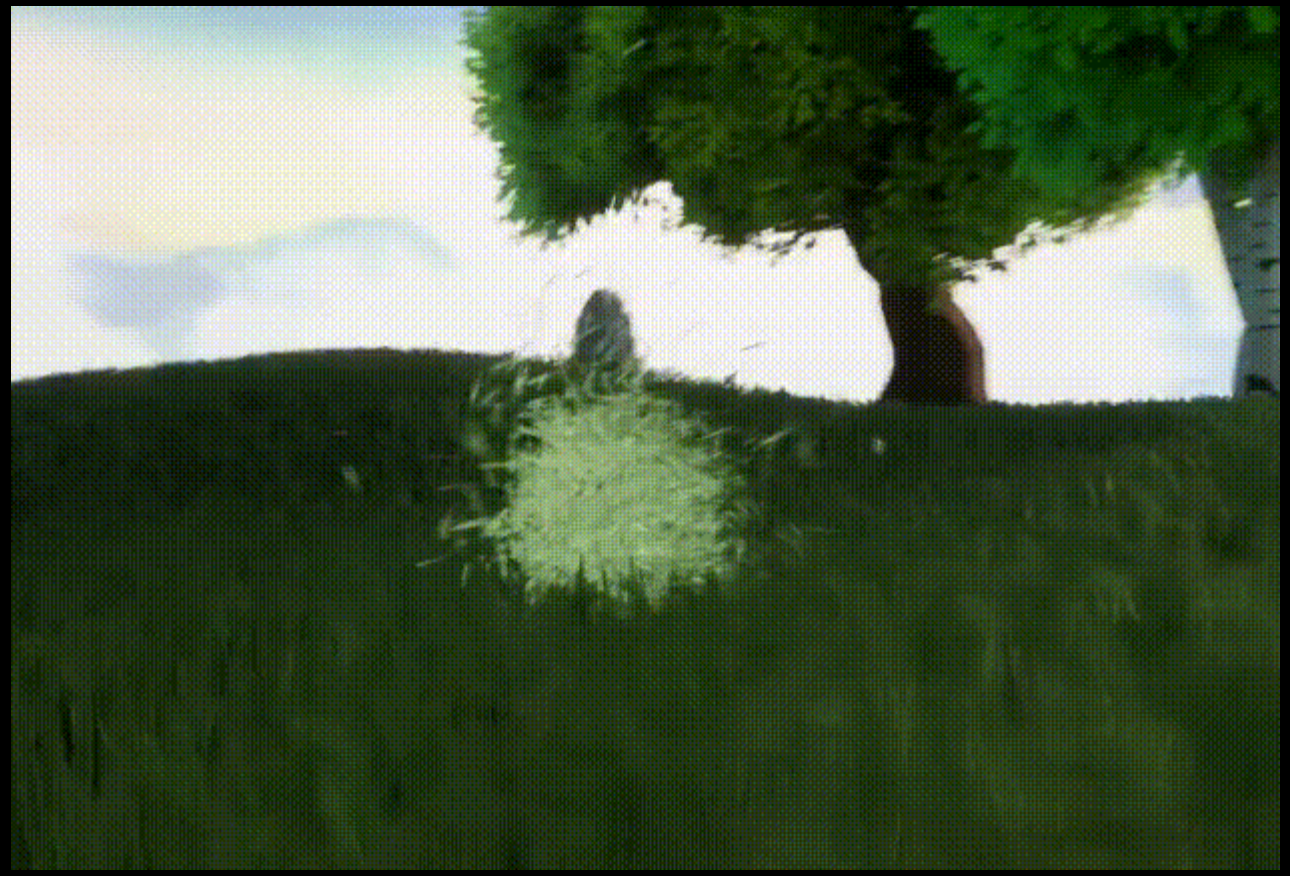
TAKEAWAYS

- Niagara produces results fast. Great for small scale effects or for prototyping.

- Runtime virtual textures are awesome for getting real-time data about surroundings

- Interactions using Render Target Textures

- Bézier curves are easy, fast, cheap and look good so are great for visuals.



THANK YOU

RESOURCES

- **GDCVAULT: PROCEDURAL GRASS IN 'GHOST OF TSUSHIMA'**

https://gdcvault.com/play/1027214/Advanced-Graphics-Summit-Procedural-Grass

- **80LV: SETTING UP A REALISTIC GRASS & WIND ANIMATION IN UNREAL**

https://cdn.80.lv/articles/tutorial-setting-up-a-realistic-grass-wind-animation-in-unreal

- **OUTERRA: PROCEDURAL GRASS RENDERING**

https://outerra.blogspot.com/2012/05/procedural-grass-rendering.html

- **CREATING A SPIDER SWARM IN UNREAL ENGINE NIAGARA**

https://www.youtube.com/watch?v=NFs4D_G7t_Q

- **INTRODUCTION TO NIAGARA**

https://www.youtube.com/watch?v=GIzwxwrM9Mk

- **CREATING SNOW TRAILS IN UNREAL ENGINE 4**

https://www.raywenderlich.com/5760-creating-snow-trails-in-unreal-engine-4

- **HOW TO BLEND OBJECTS WITH YOUR LANDSCAPE - UE4 RUNTIME VIRTUAL TEXTURING (RVT) TUTORIAL**

https://www.youtube.com/watch?v=xYuIDFzKaF4

- **ASSETS USED IN PROJECT**

"Dreamscape Nature: Meadows" by Polyart Studio - https://www.unrealengine.com/marketplace/en-US/product/dreamscape-nature-meadows

Q&A – LINK TO SLIDES

tinyurl.com/Grass2022