



# RENDERING MILLIONS OF GRASS BLADES **USING UE AND NIAGARA**

Radosław Paszkowski

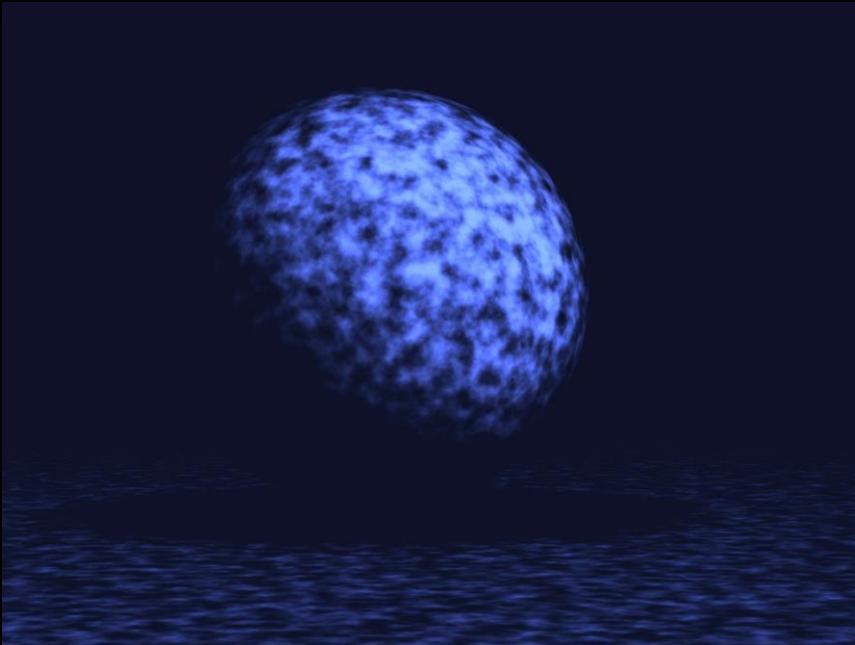
**PixelAnt**  
GAMES

---

A SUMO DIGITAL STUDIO

# RADEK PASZKOWSKI

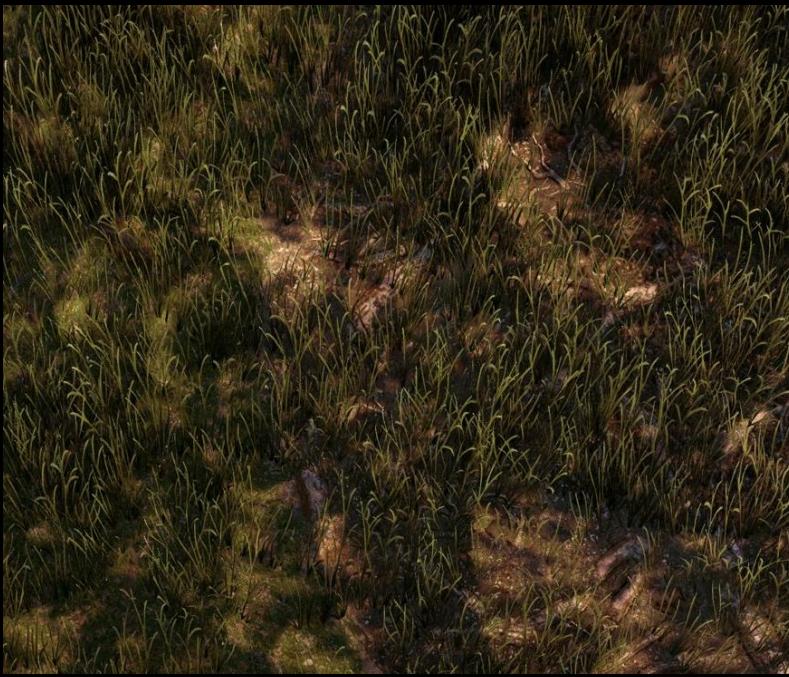
---



**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

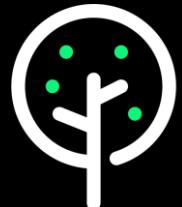
Credits: Genshin Impact by miHoYo



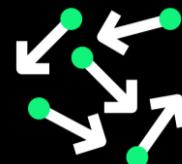
Credits: Grounded by Obsidian Entertainment

# ASSUMPTIONS

---



3D Opaque Models



Deterministic



Millions of instances



Artist authored

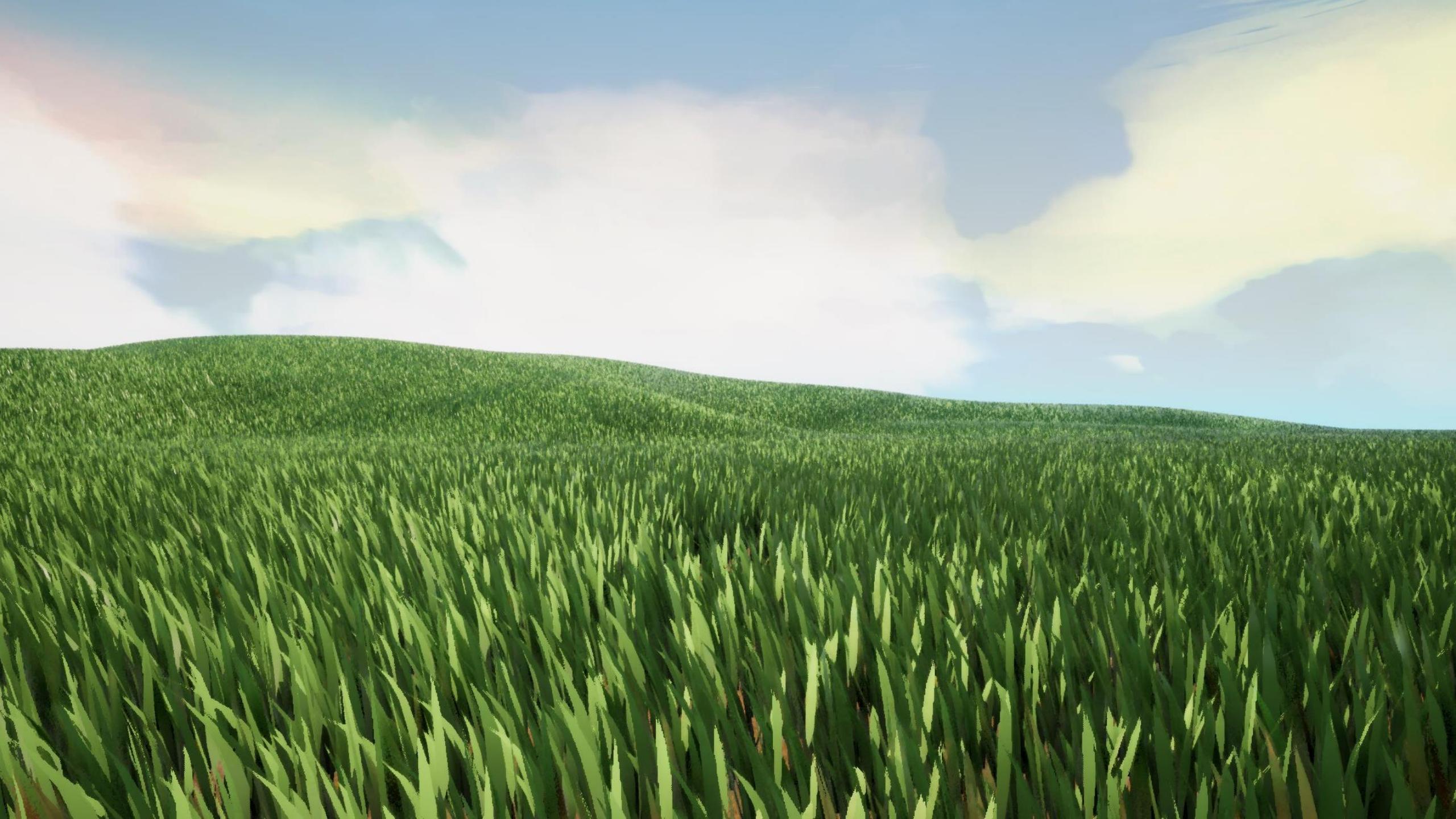


Interactable



Real-Time

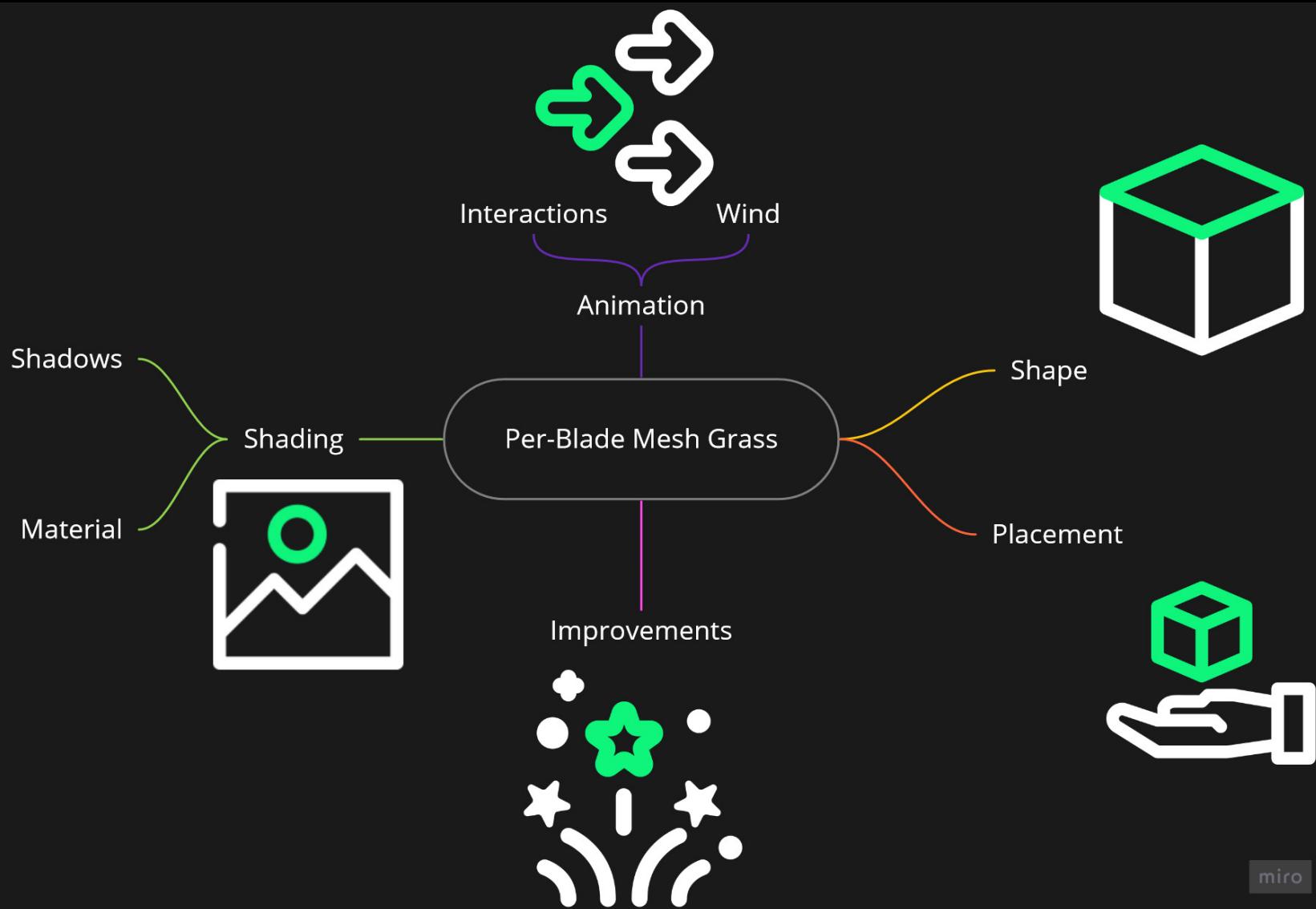




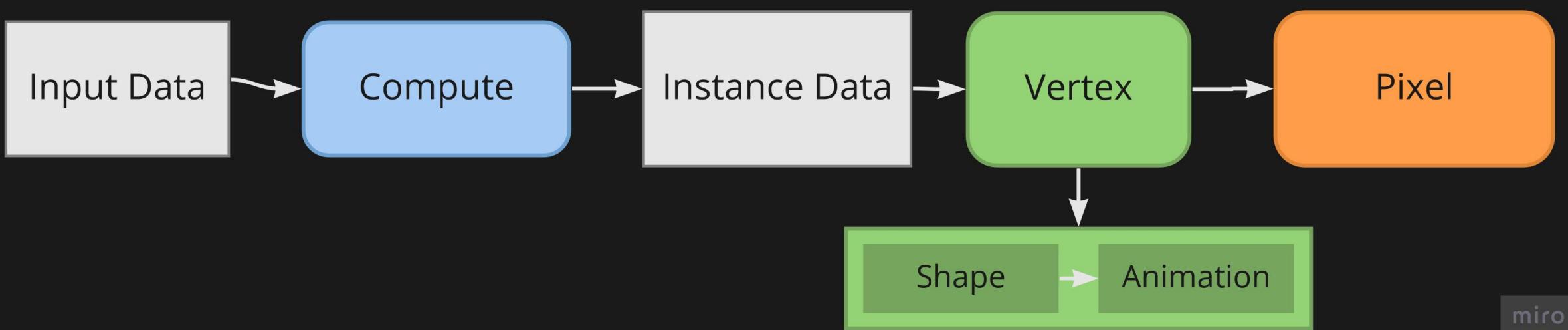


# Concreteosis - Betonoza





# GRASS SYSTEM OVERVIEW



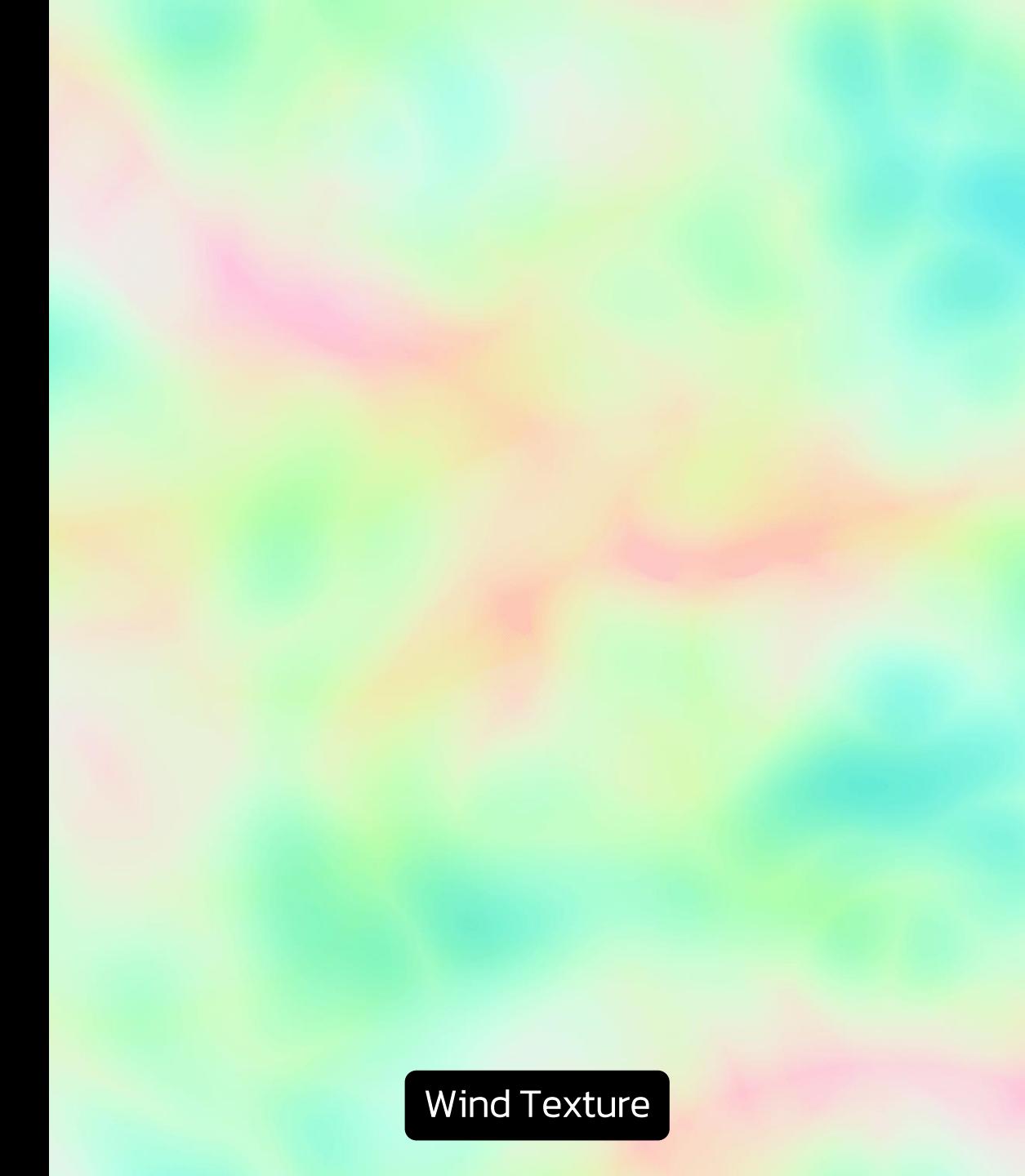
# PROCESS

miro



# INPUT DATA

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



# GENERATING WIND TEXTURE

$$\text{wind.x} = 0.5 (1 + \cos(\text{noise}_{xy}(\text{uv} + kt)))$$

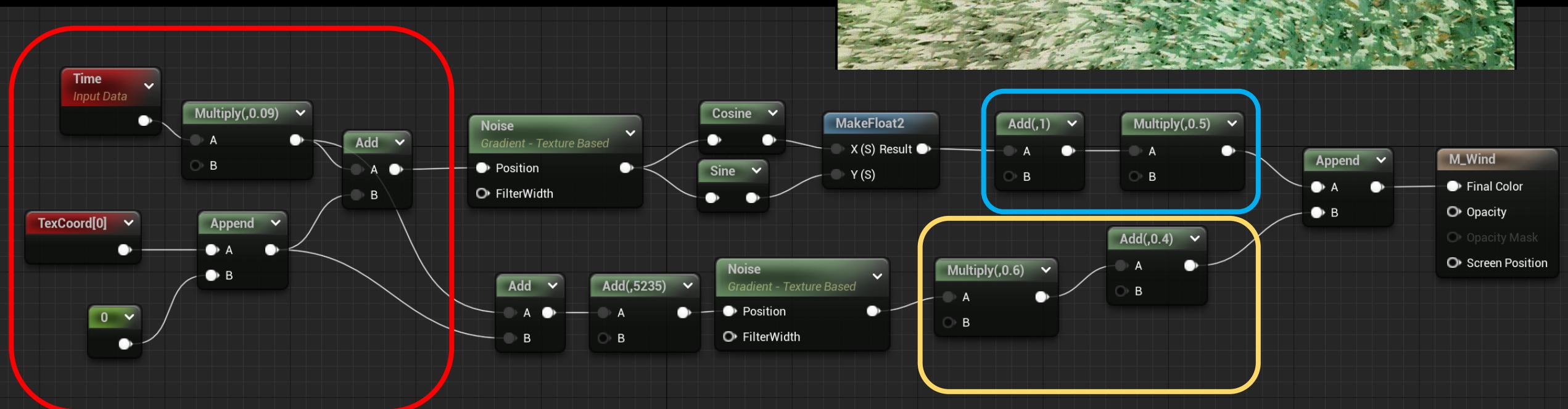
$$\text{wind.y} = 0.5 (1 + \sin(\text{noise}_{xy}(\text{uv} + kt)))$$

$$\text{wind.z} = 0.4 + 0.6 \text{noise}_z((\text{uv} + kt) + 0.5235)$$

Generate random seed from position and time

Change range to [0-1]

Change range to [0.4-1]

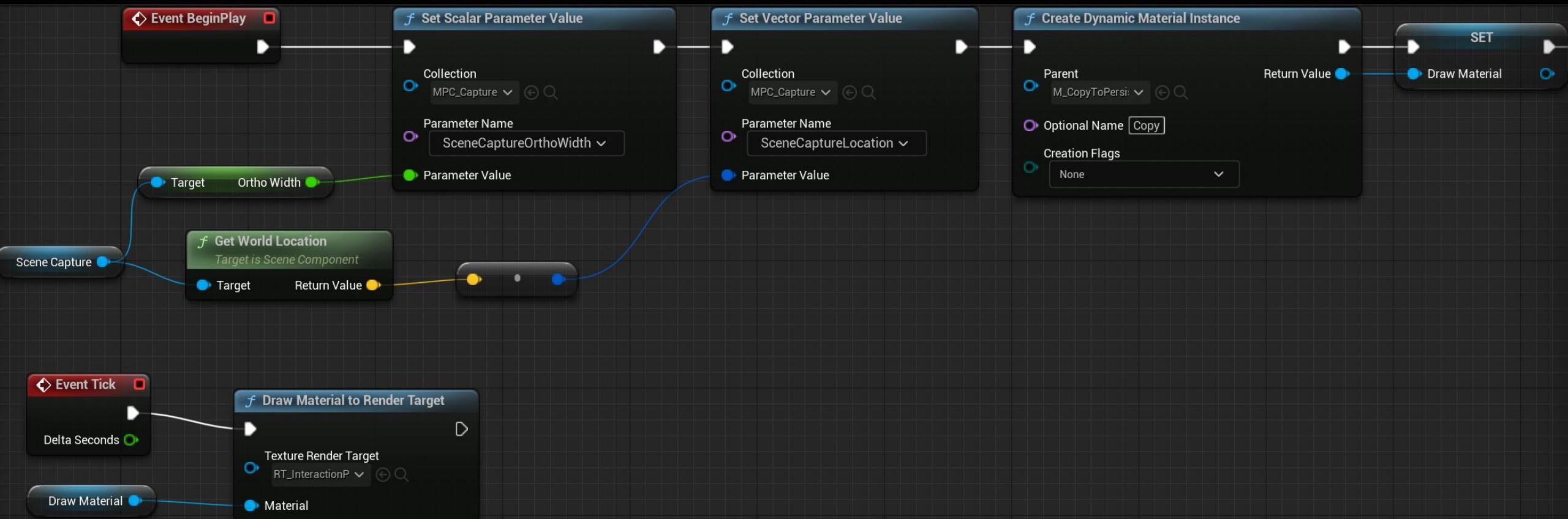


# INTERACTION TEXTURE

---



# INTERACTION TEXTURE



# HEIGHTMAP USING RVT

Writing – Landscape Material

Absolute World Position  
Input Data

Runtime Virtual Texture Output

- BaseColor
- Specular
- Roughness
- Normal
- WorldHeight
- Opacity
- Mask

Reading – Grass Material

Runtime Virtual Texture Sample

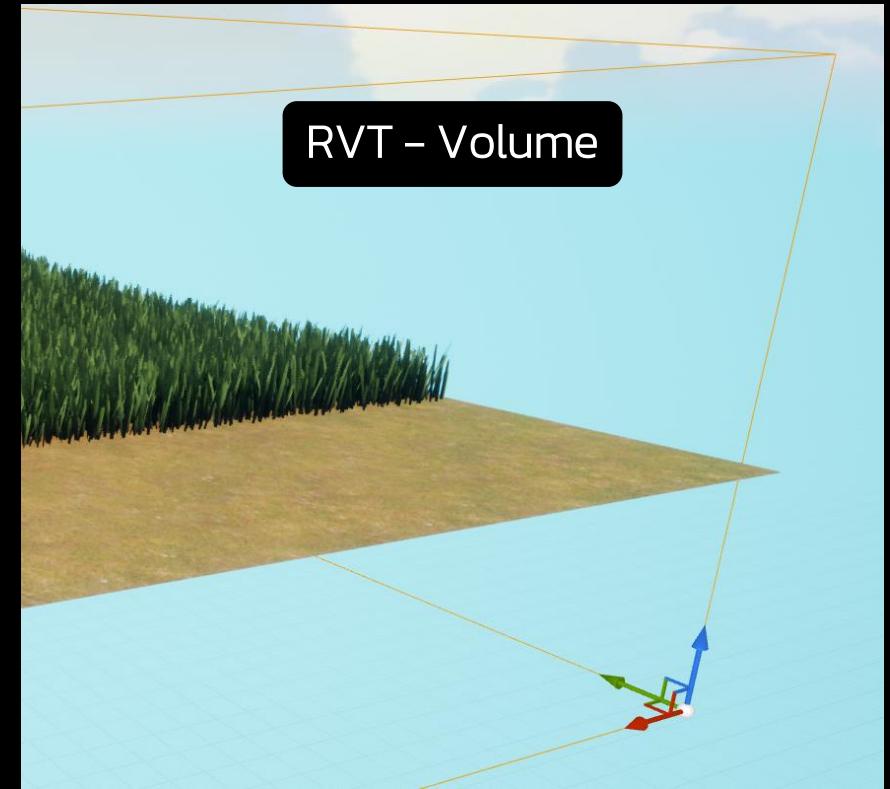
- UVs
- WorldPosition
- MipValue
- BaseColor
- Specular
- Roughness
- Normal
- WorldHeight
- Mask

Append

0,0

A

B



RuntimeVirtualTextureVolume

RuntimeVirtualTextureVolume (Instance)

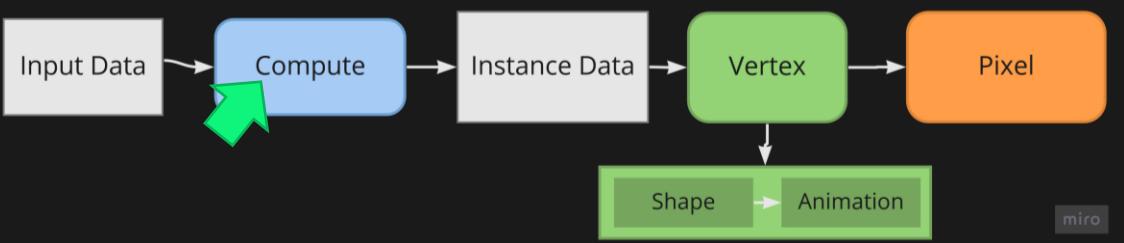
VirtualTextureComponent (VirtualTextureComponent)

Virtual Texture

RVT\_HeightMap

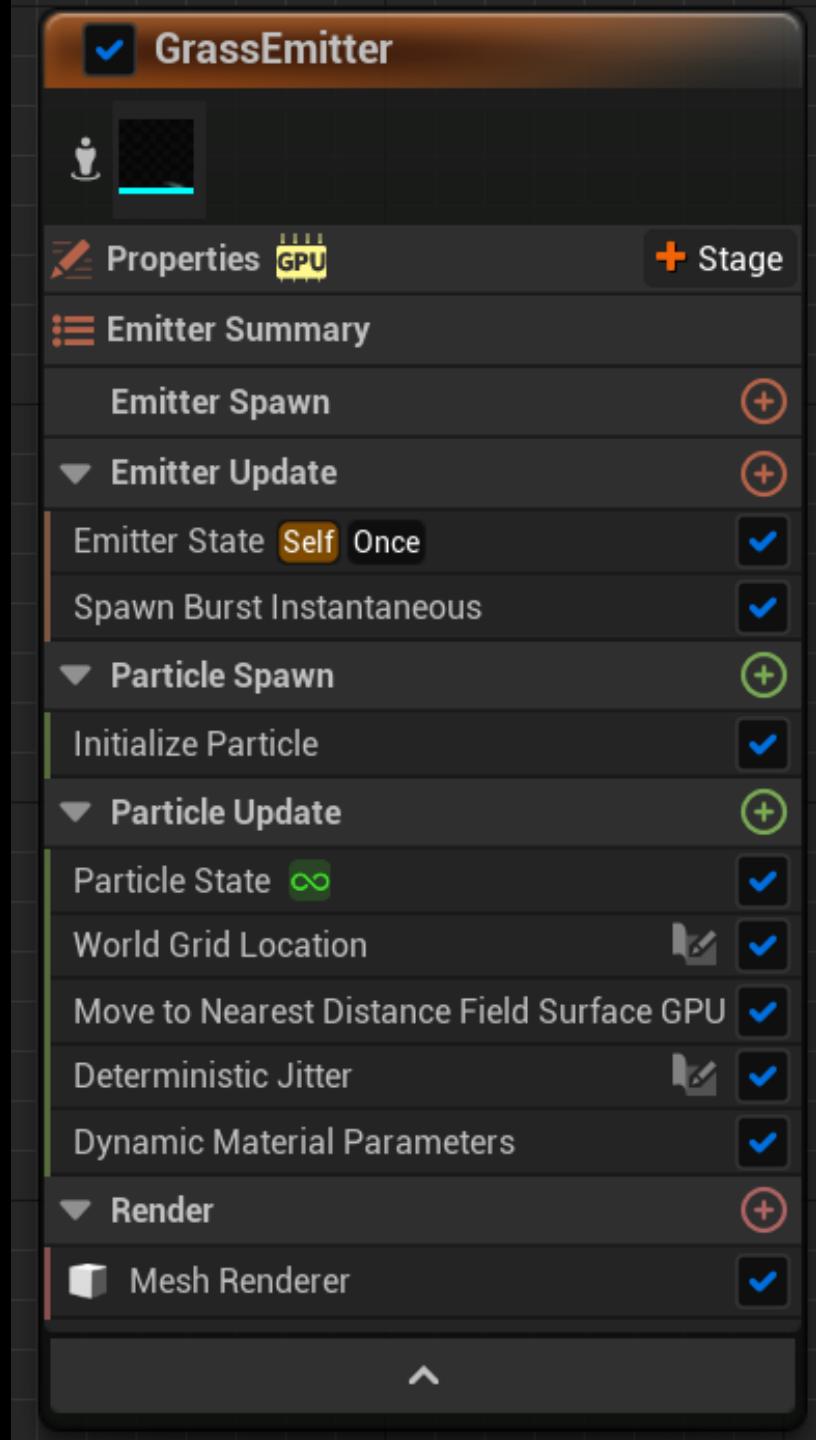
Advanced

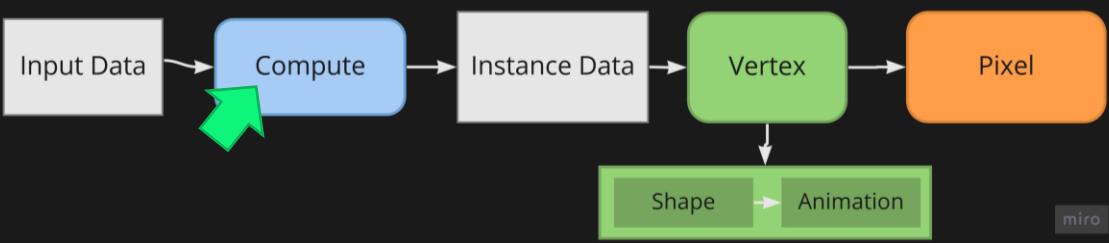
Virtual Texture Build



# COMPUTE – NIAGARA

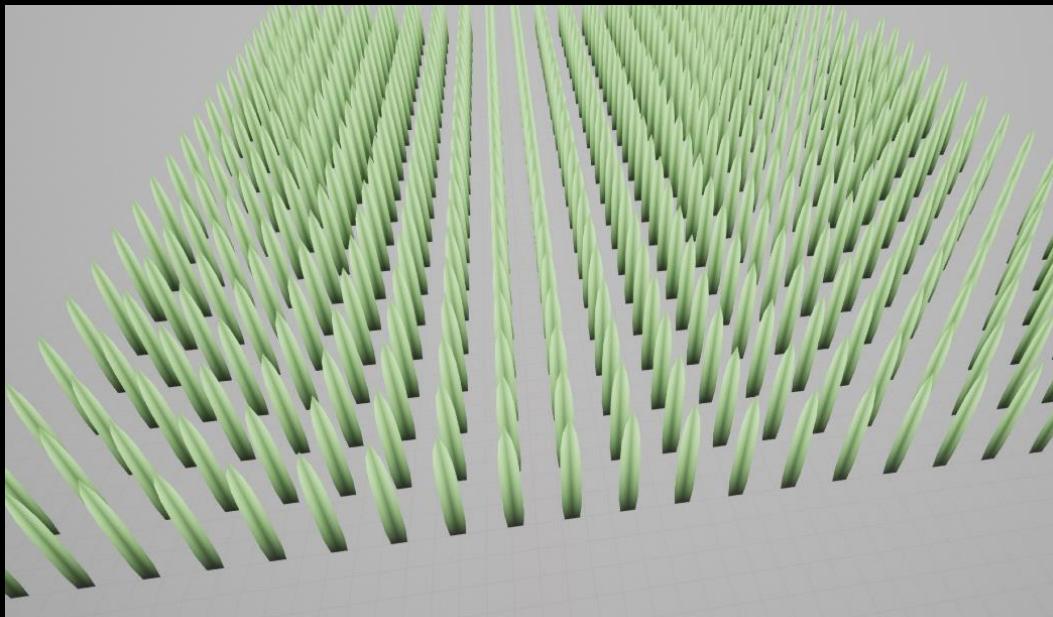
- GRASS GRID POSITION
- FACING DIRECTION
- BÉZIER CURVE PARAMETERS



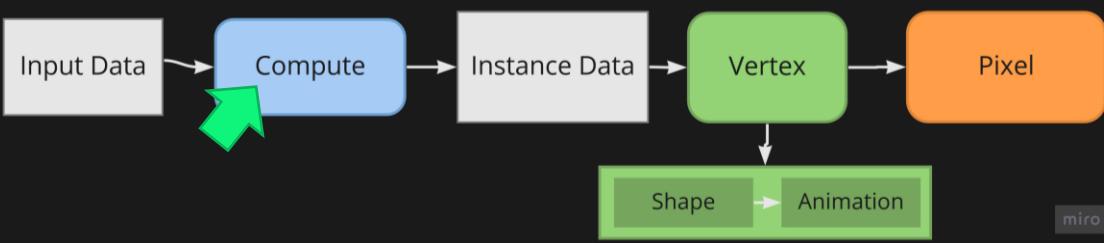


# NIAGARA PLACEMENT CALCULATION

- **GRID SIZE**
- **SPAWN COUNT**
- **UNIQUE PARTICLE ID**



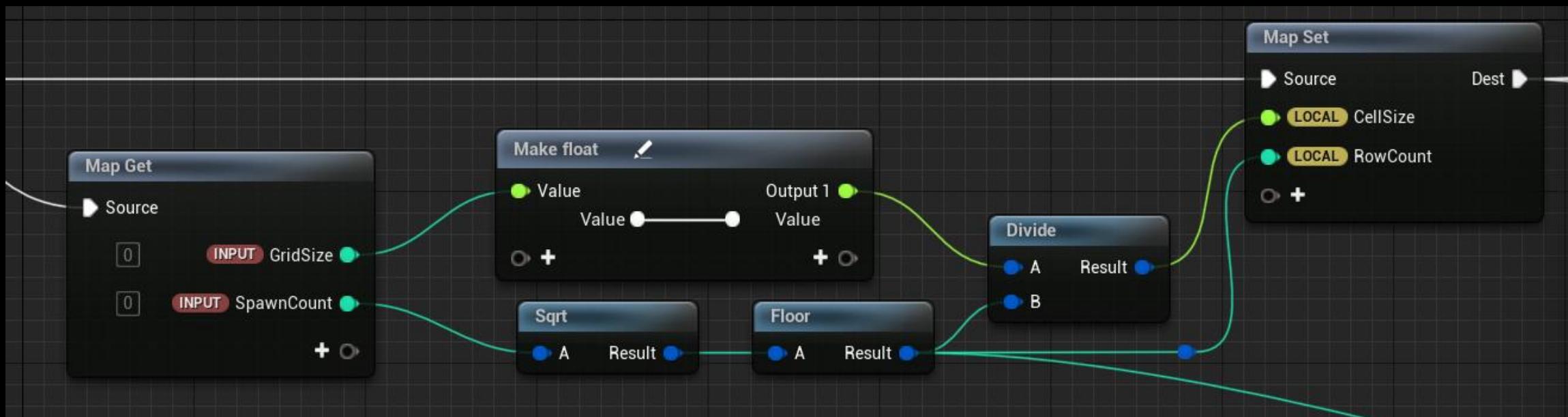
The screenshot shows the Niagara Editor interface. On the left, the 'Grass\_Test' emitter is selected, displaying its properties under 'Properties'. The 'System Update' section is expanded, showing 'Emitter Spawn' and 'Particle Spawn' settings. The 'Particle Update' section is currently active. On the right, the 'Library Filtering' sidebar is open, showing a list of available modules: Mask, Mass, Material, Math, Mesh, Orientation, Physics, Post Solve, Ribbon, Size, Skeletal Mesh, Sprite, SubUV, Texture, Utility, Vector Field, and Velocity. A context menu is open at the bottom right, with 'New Scratch Pad Module' and 'Set new or existing parameter directly' options visible.

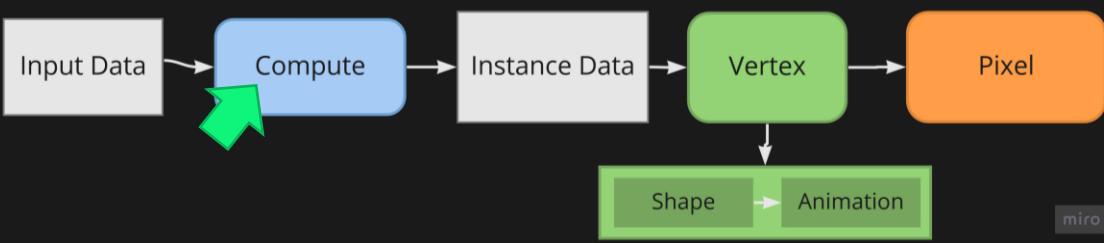


# NIAGARA PLACEMENT CALCULATION

- **GRID SIZE**
- **SPAWN COUNT**
- **UNIQUE PARTICLE ID**

$$\begin{aligned}
 LineCount (Row\&Column) &= \lfloor \sqrt{SpawnCount} \rfloor \\
 Cell Size &= \frac{GridSize}{LineCount}
 \end{aligned}$$





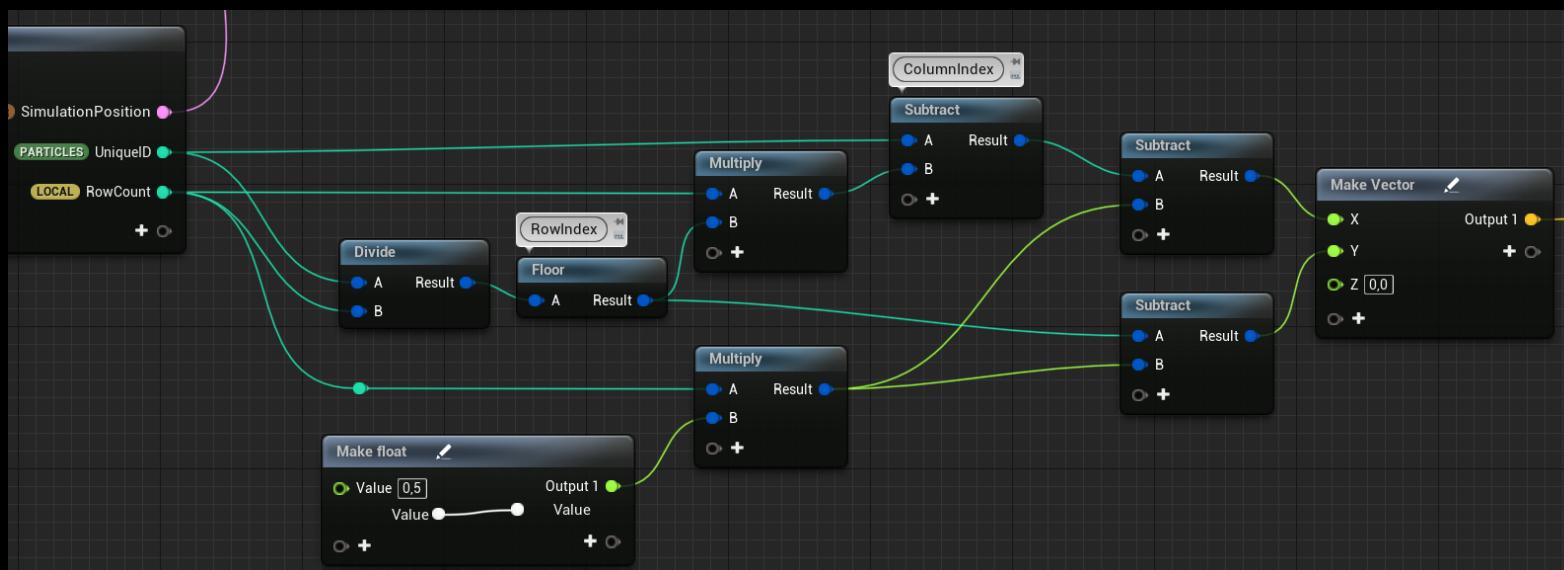
# NIAGARA PLACEMENT CALCULATION

$$\text{RowIndex} = \left\lfloor \frac{\text{ParticleID}}{\text{LineCount}} \right\rfloor$$

$$\text{Column Index} = \text{ParticleID} - \text{RowIndex} * \text{LineCount}$$

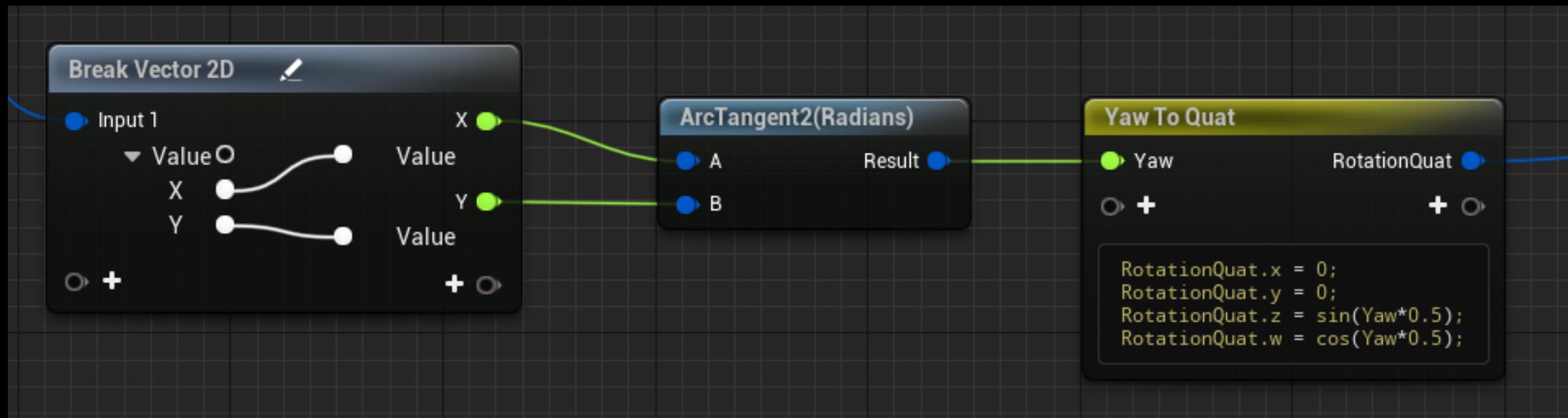
$$Position = CellSize * (ColumnIndex, RowIndex) - 0.5LineCount$$

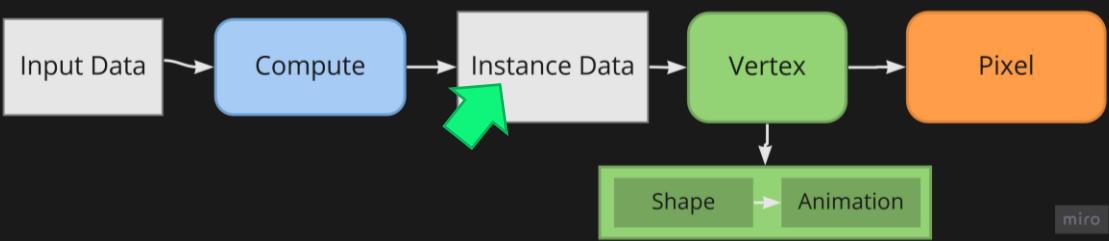
$$BladeHash = Hash(Position)$$





# 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**
  - **TIILT**
  - **BEND**
  - **MIDPOINT**
- **TWIST**

**Dynamic Parameter**  
*Input Data*

BladeHash

Twist

Width

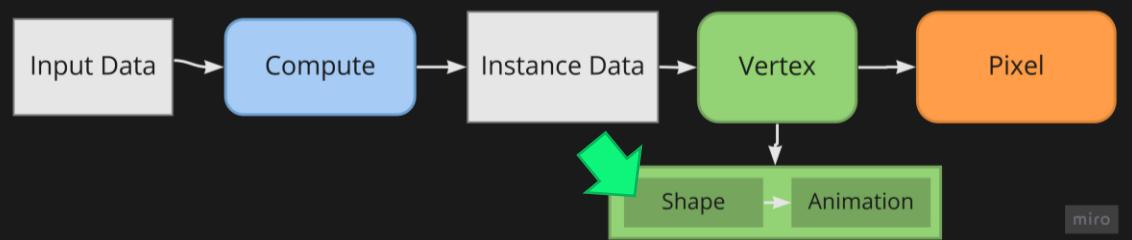
**Dynamic Parameter**  
*Input Data*

Tilt

Midpoint

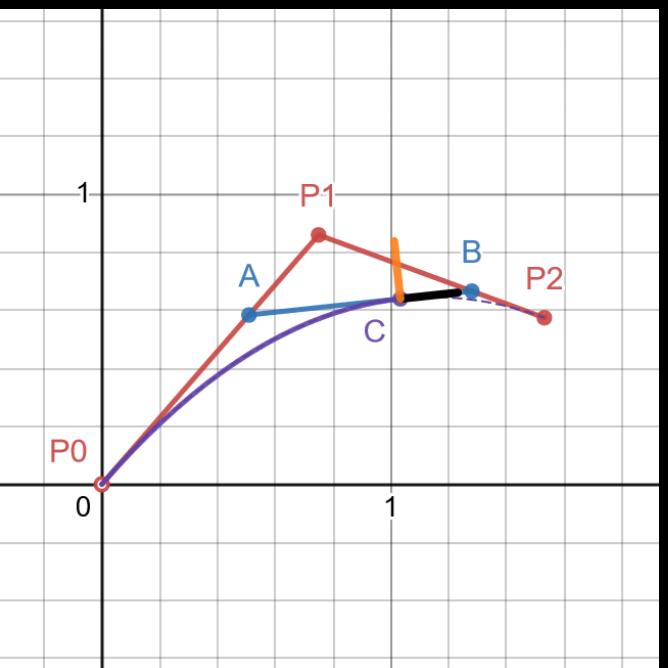
Bend

Height

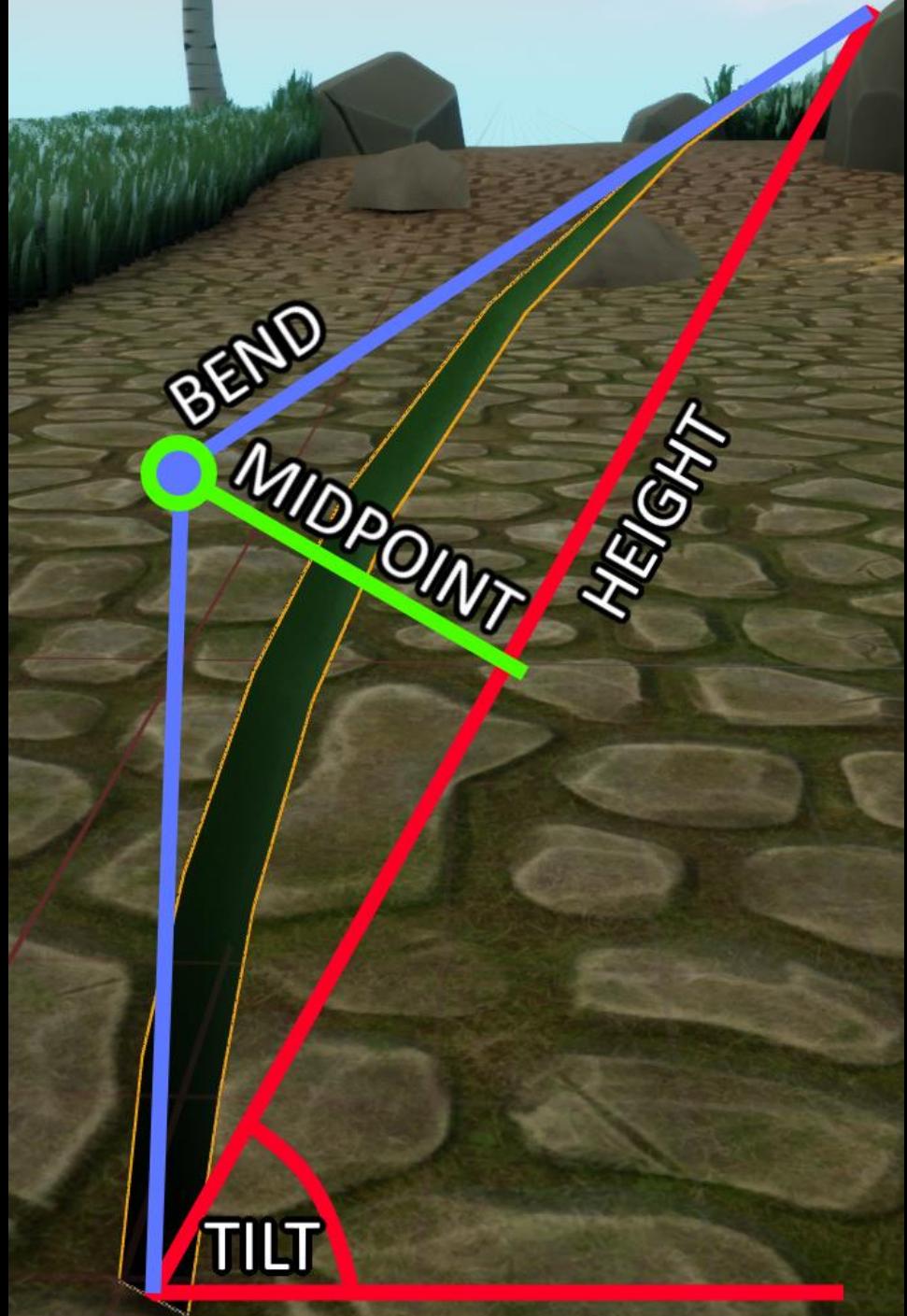


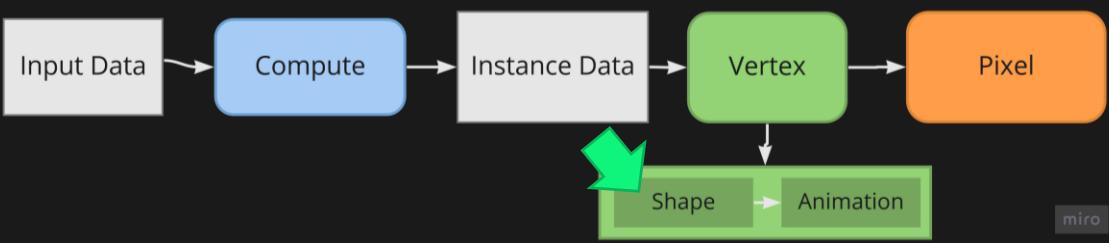
# SHAPE - BEZIER CURVE

- 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;`

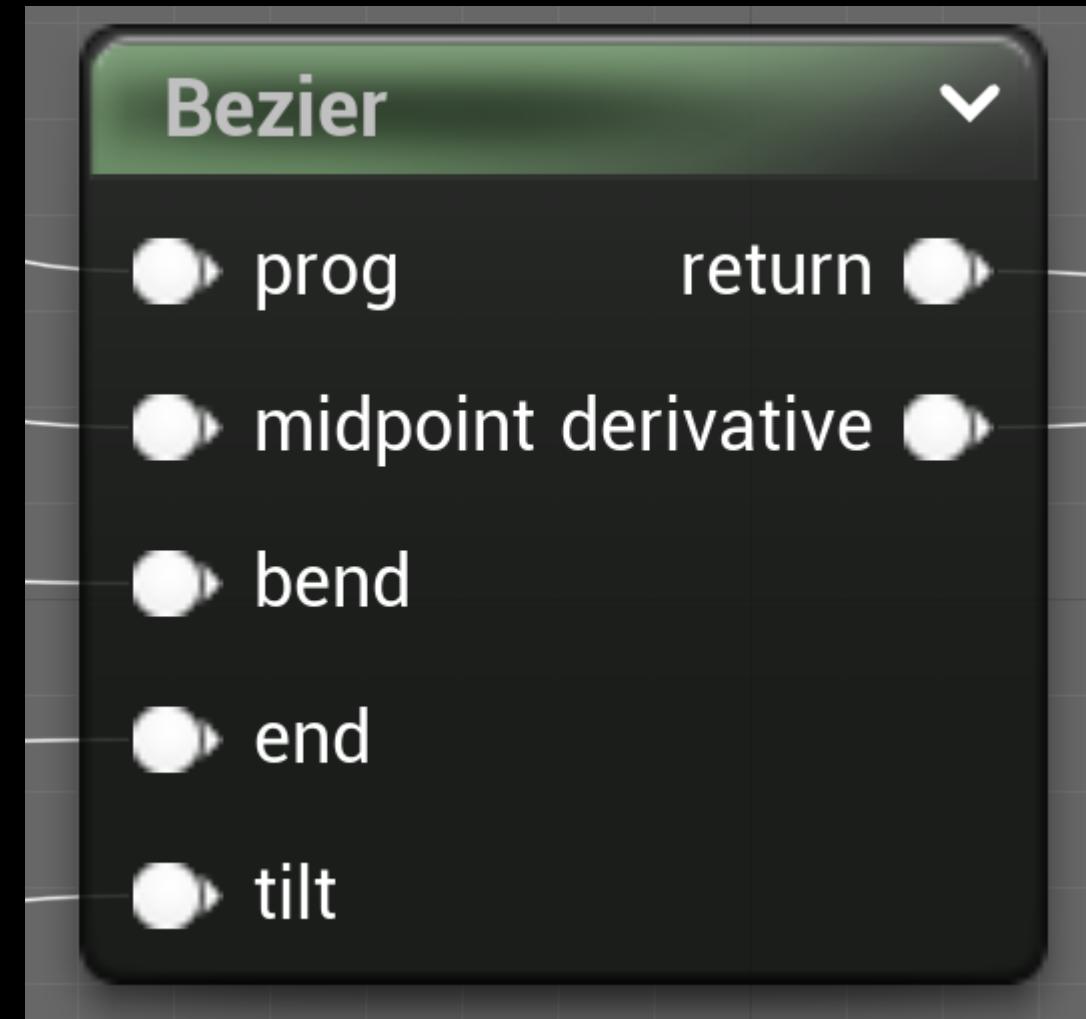
`bezier = (2.0 * (1 - prog) * prog) * mid + prog * prog * end;`

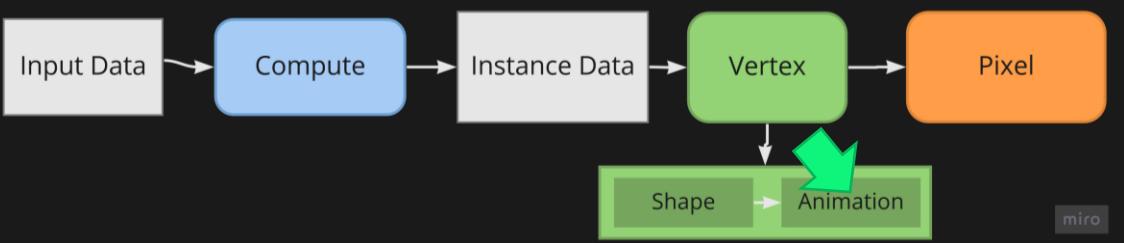
`derivative = 2.0 * (1.0 - 2.0 * prog) * mid + 2.0 * prog * end;`

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2P_2$$

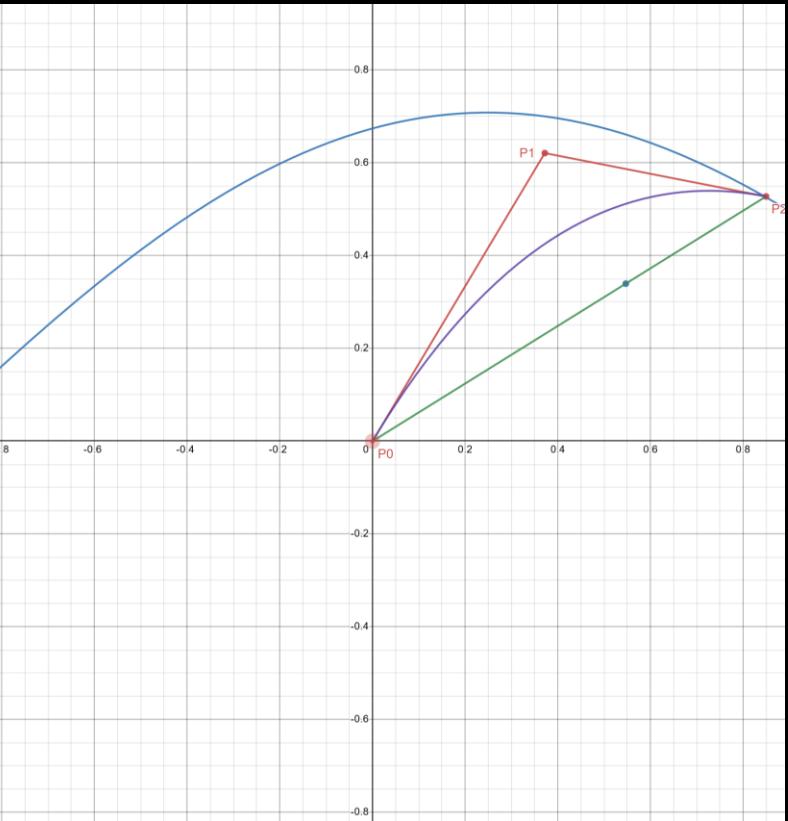
$$B'(t) = 2(t-1)P_0 + 2(1-2t)P_1 + 2tP_2$$

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

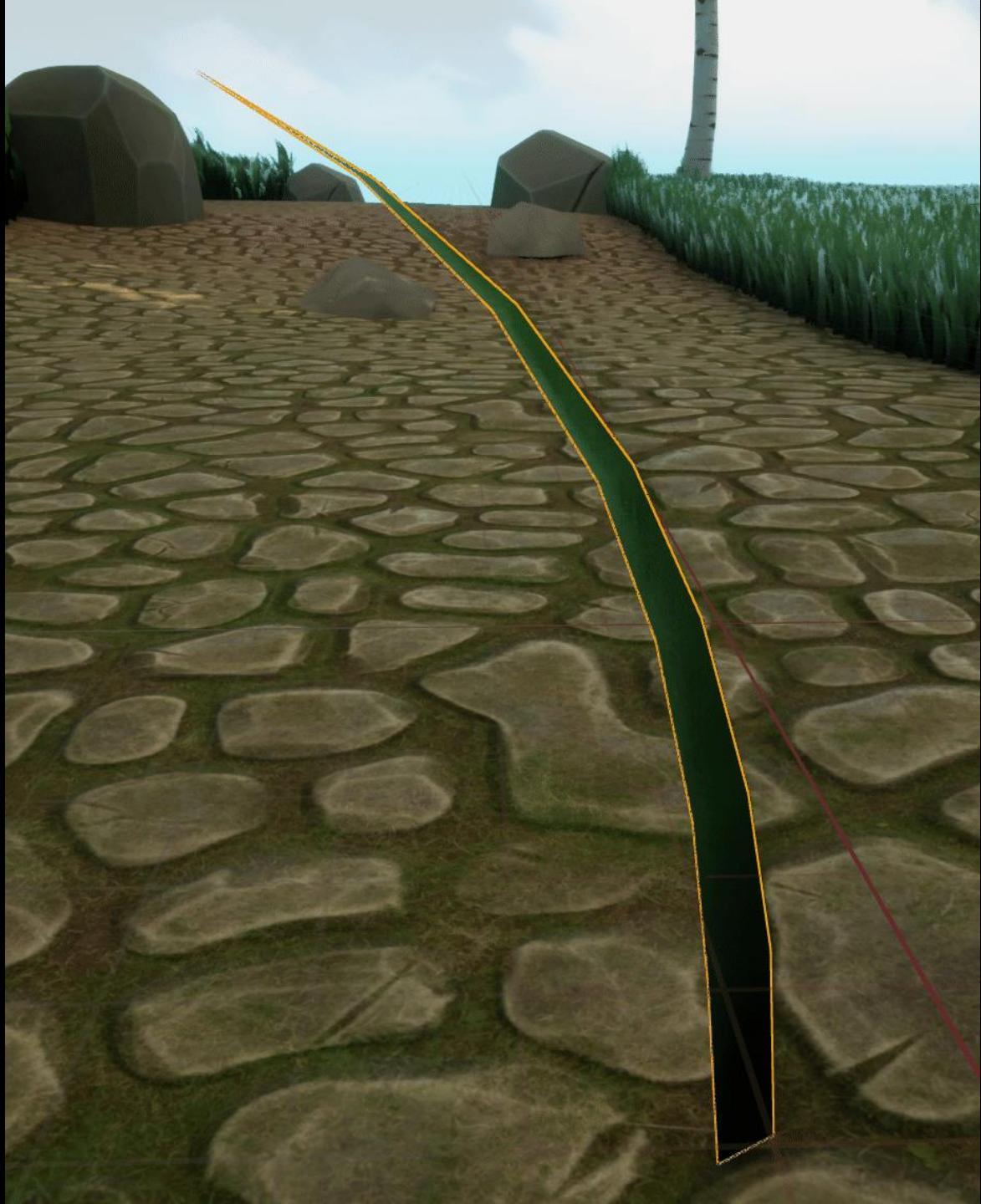


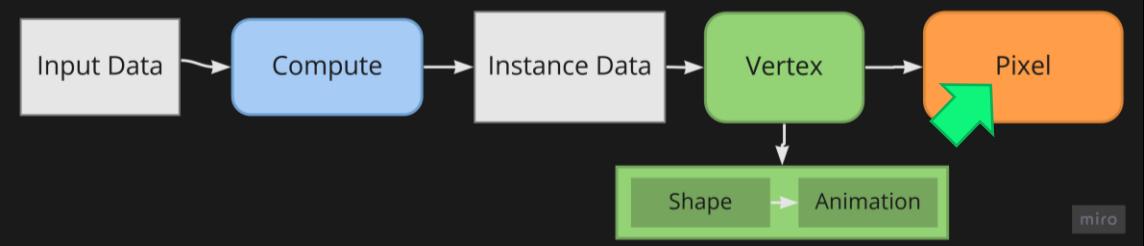


# ANIMATION



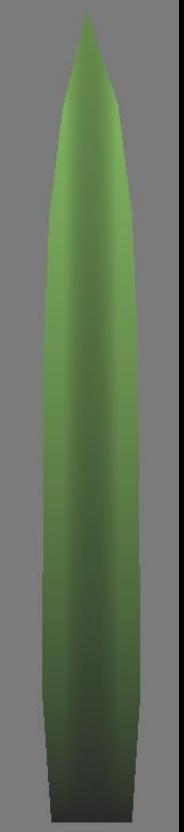
Desmos | Bézier Curve Animation



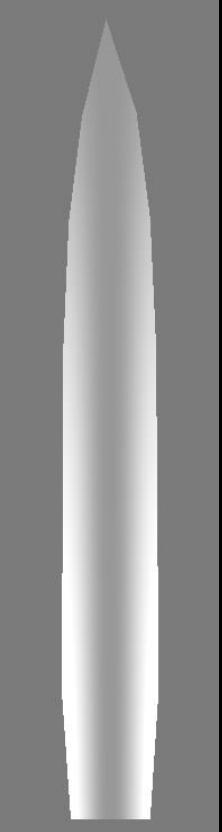


# PIXEL

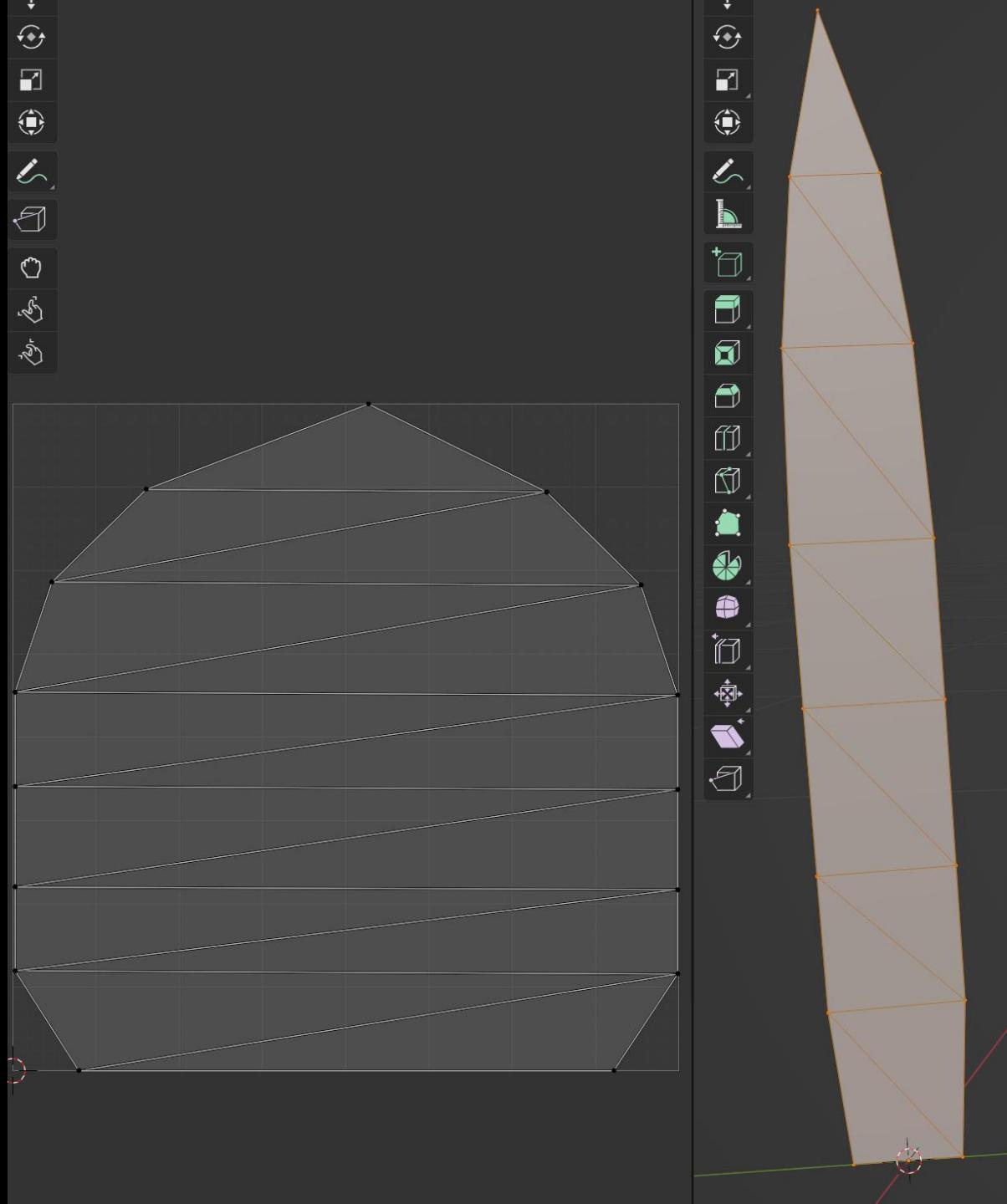
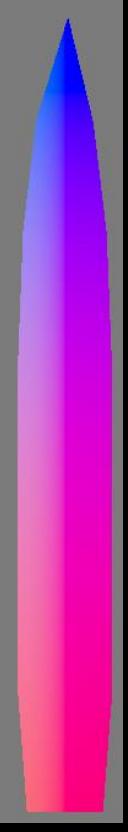
COLOR

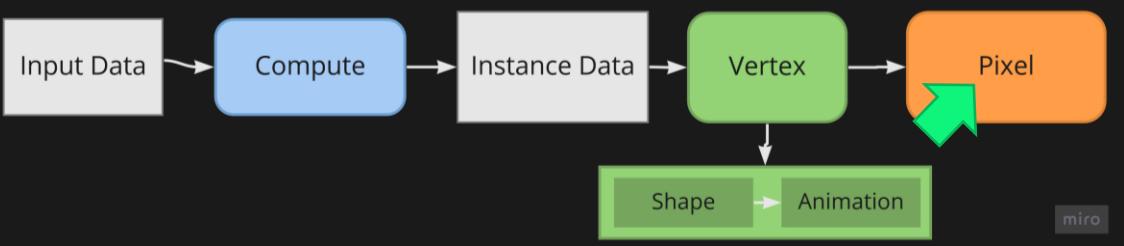


ROUGHNESS  
(INVERSE GLOSS)



NORMAL





# PIXEL

**Color and gloss**

This screenshot shows a complex node-based shading graph in Miro, likely for a game engine like Unity or Unreal Engine. The graph is titled "Color and gloss".

**Inputs:**

- TexCoord[0]
- Particle Color (Input Data)
- GlossIntensity (Param (0.4))

**Operations:**

- Mask (R) and Mask (G) nodes extract red and green components from TexCoord[0].
- Cosine and Divide(2) nodes calculate the dot product of the normal and view direction.
- Add(1) and Divide(2) nodes calculate the cosine of the angle between the normal and the light direction.
- Power(X, 1.4) and Power(X, 1.2) nodes calculate non-linear power terms.
- Multiply nodes calculate the product of the base values and the gloss intensity.
- 1-x nodes invert the base values.
- Add(0.5) and Add nodes calculate the final color components.
- Saturate nodes clamp the color values between 0 and 1.
- Final output nodes produce the R, G, B, and A channels.

**Outputs:**

The final output consists of four channels: Red, Green, Blue, and Alpha (A). The Red and Green channels show a vertical gradient from dark green at the bottom to bright yellow-green at the top. The Blue channel is mostly black. The Alpha channel shows a smooth transition from black at the bottom to white at the top.



**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.



A scenic view of a green hillside under a clear blue sky. A dirt path winds its way up the hill. On the left, there's a small, stylized figure sitting on the grass. In the center, a large tree with green leaves stands next to a grey structure. The hillside is covered in vibrant green grass.

**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](https://www.youtube.com/watch?v=NFs4D_G7t_Q)

- **INTRODUCTION TO NIAGARA**

<https://www.youtube.com/watch?v=GlxwxwrM9Mk>

- **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=xYulDFzKaF4>

- **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](https://tinyurl.com/Grass2022)



