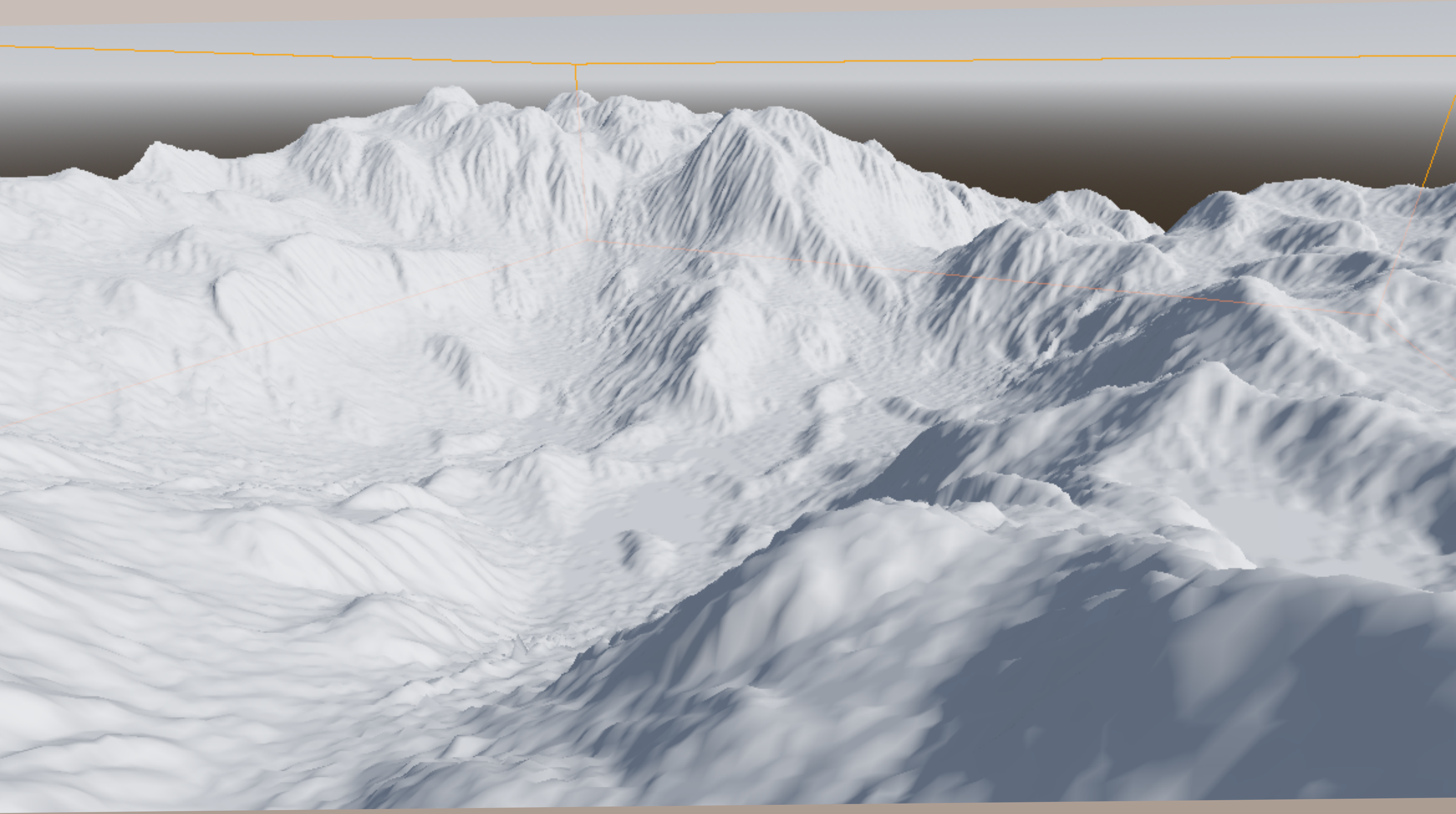
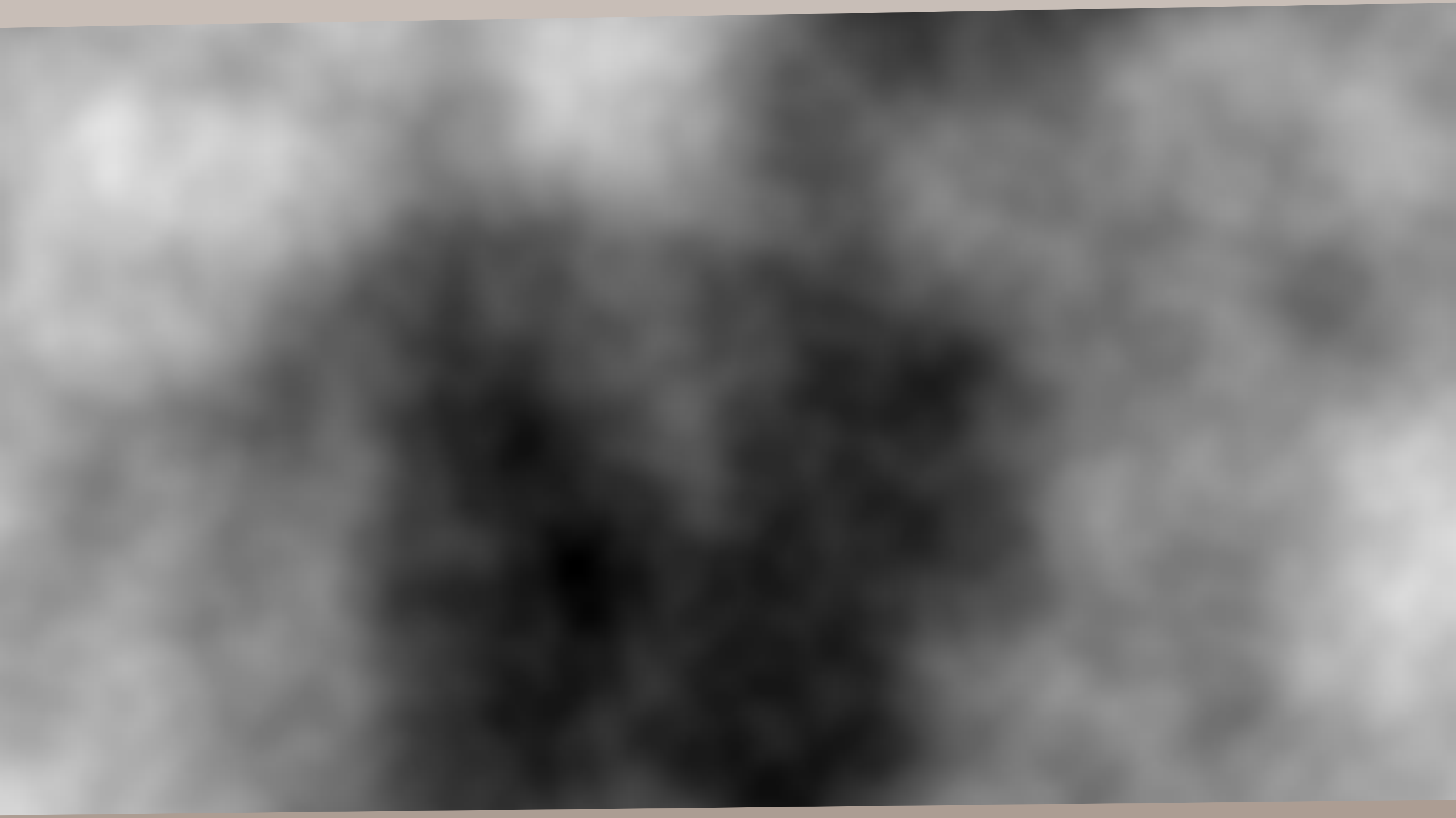
Érosion d'une montagne



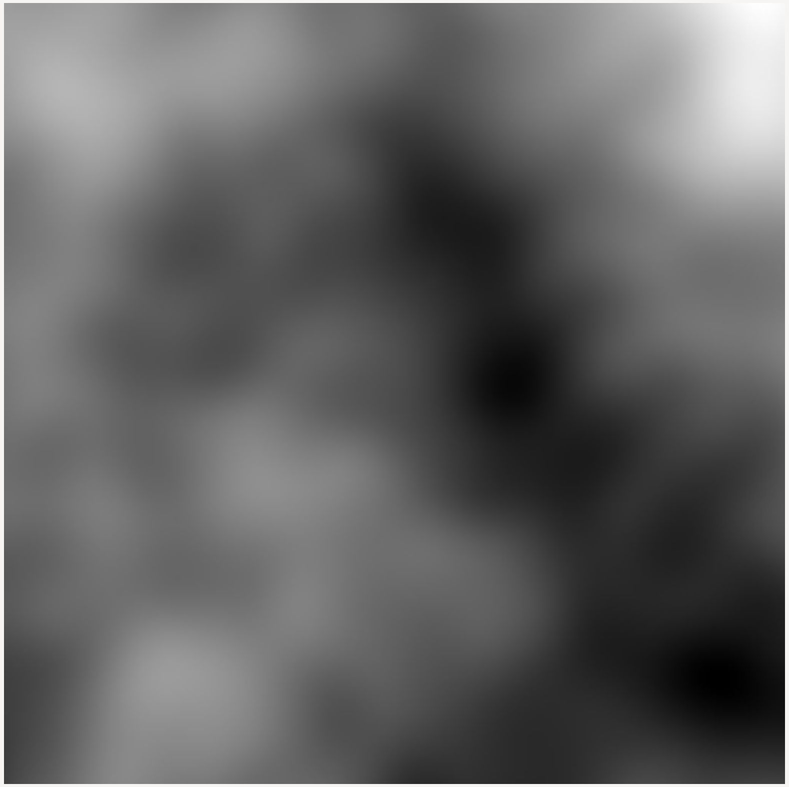
Damien LEY--THIBAULT

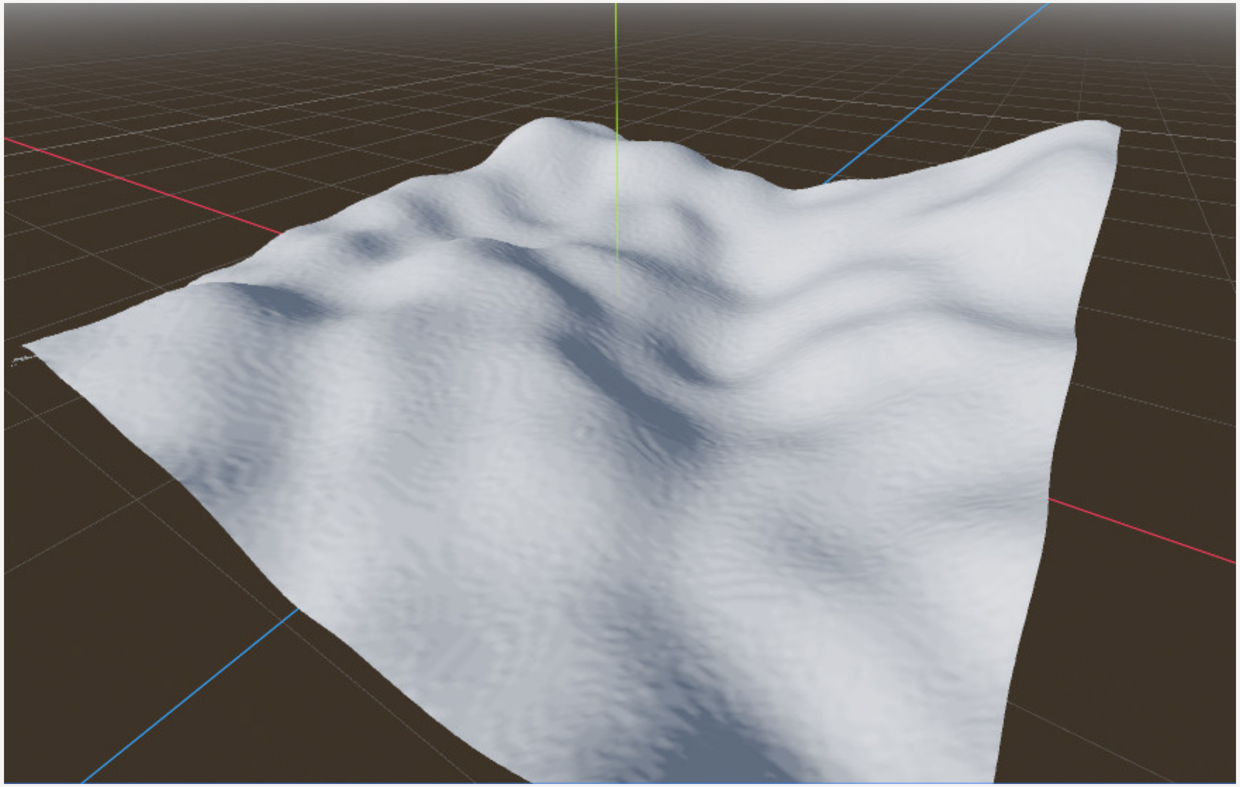
35106

Texture de hauteur



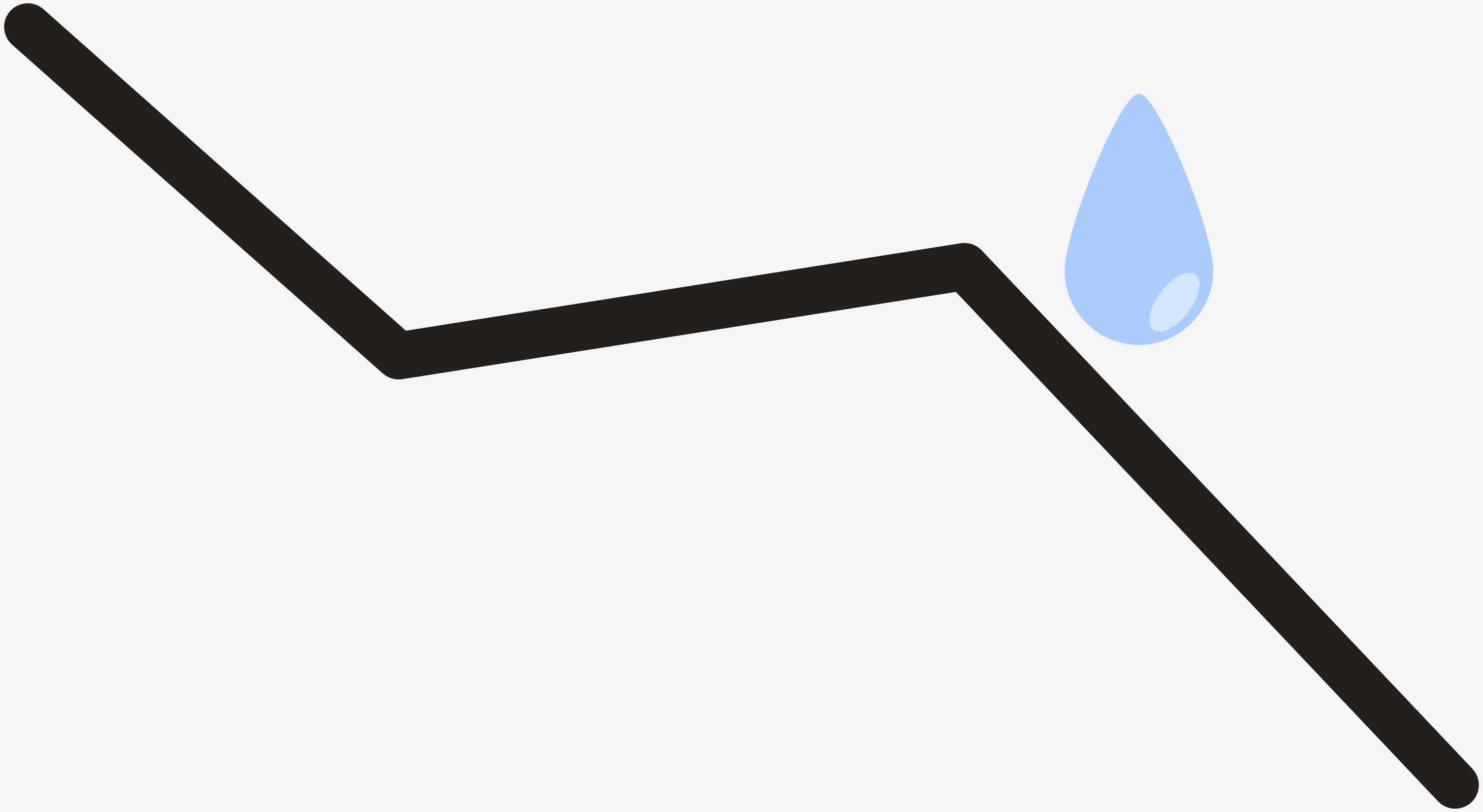
Moteur 3D



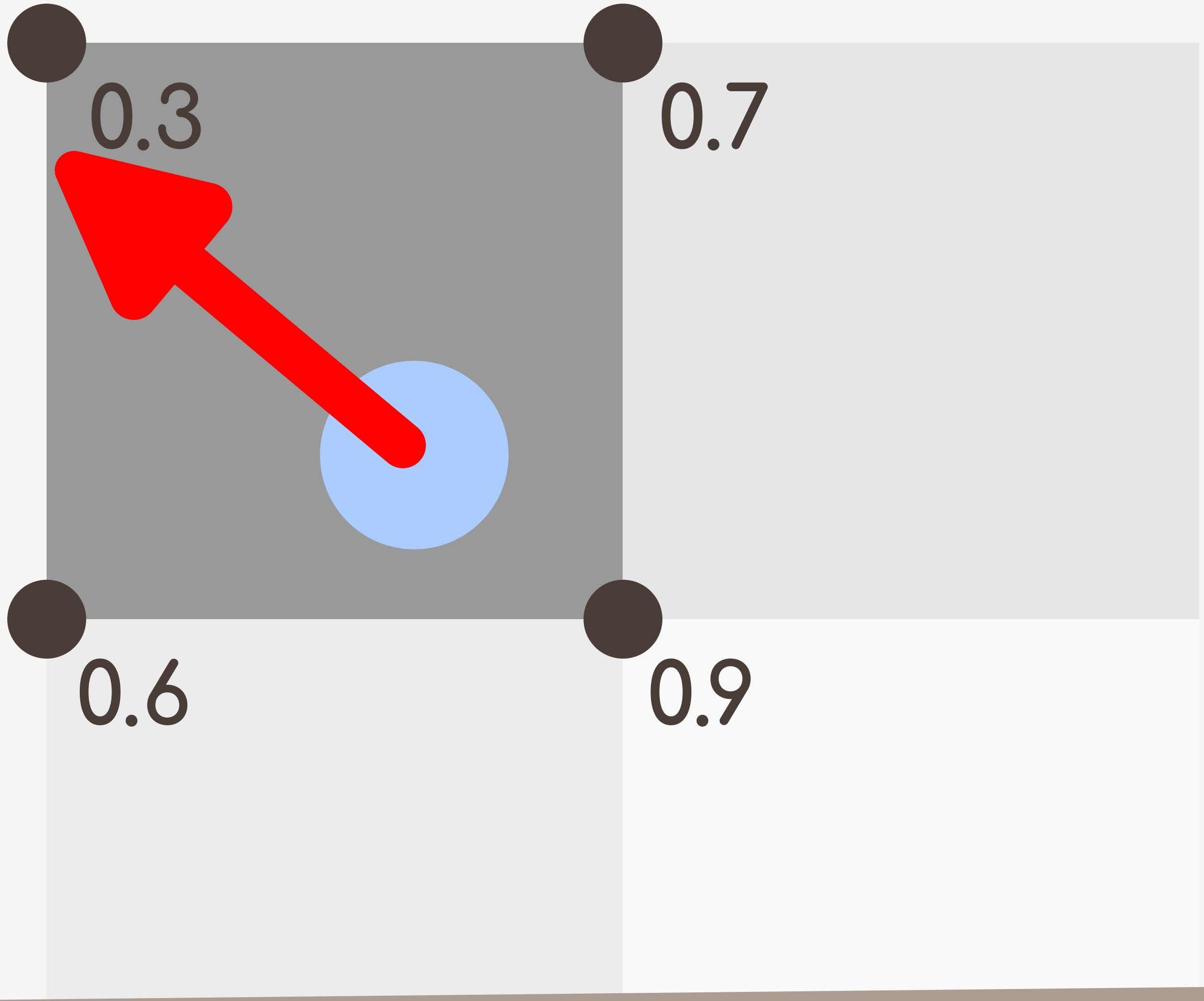


GODOT
Game engine

Déplacements



Le gradient

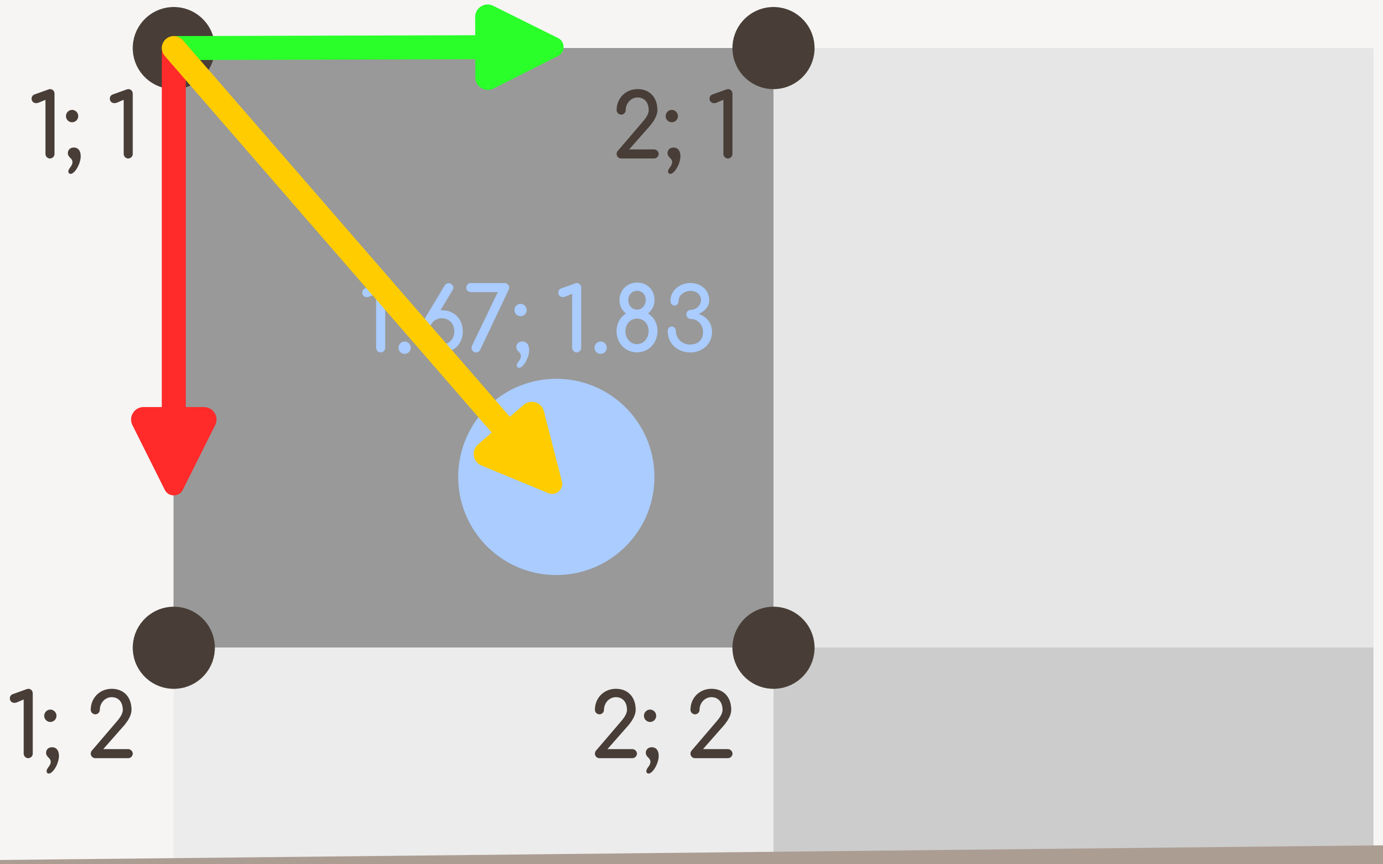


Le gradient

```
var g := Vector2(
  (Px1y - Pxy) * (1 - uv.y) + (Px1y1 - Pxy1) * uv.y,
  (Pxy1 - Pxy) * (1 - uv.x) + (Px1y1 - Px1y) * uv.x
)
```

uv ?

1.67; 1.83 => 0.67; 0.83

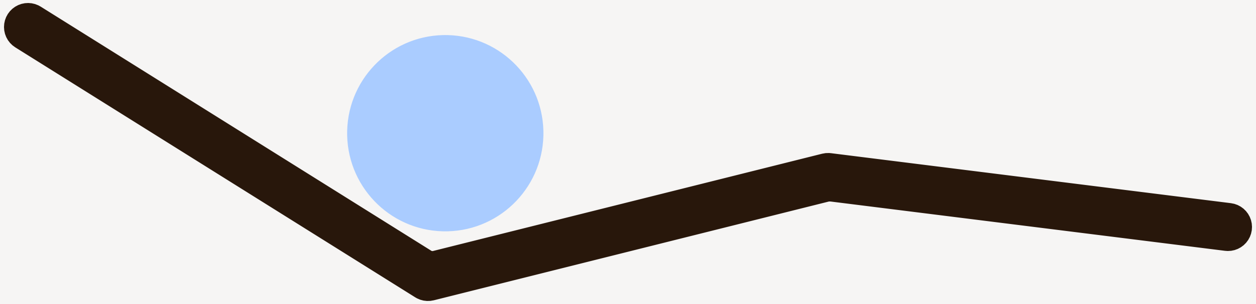


uv ?

1.67; 1.83 => 0.67; 0.83

```
var uv: Vector2 =
    position - Vector2(Vector2i(position))
```

Inertie

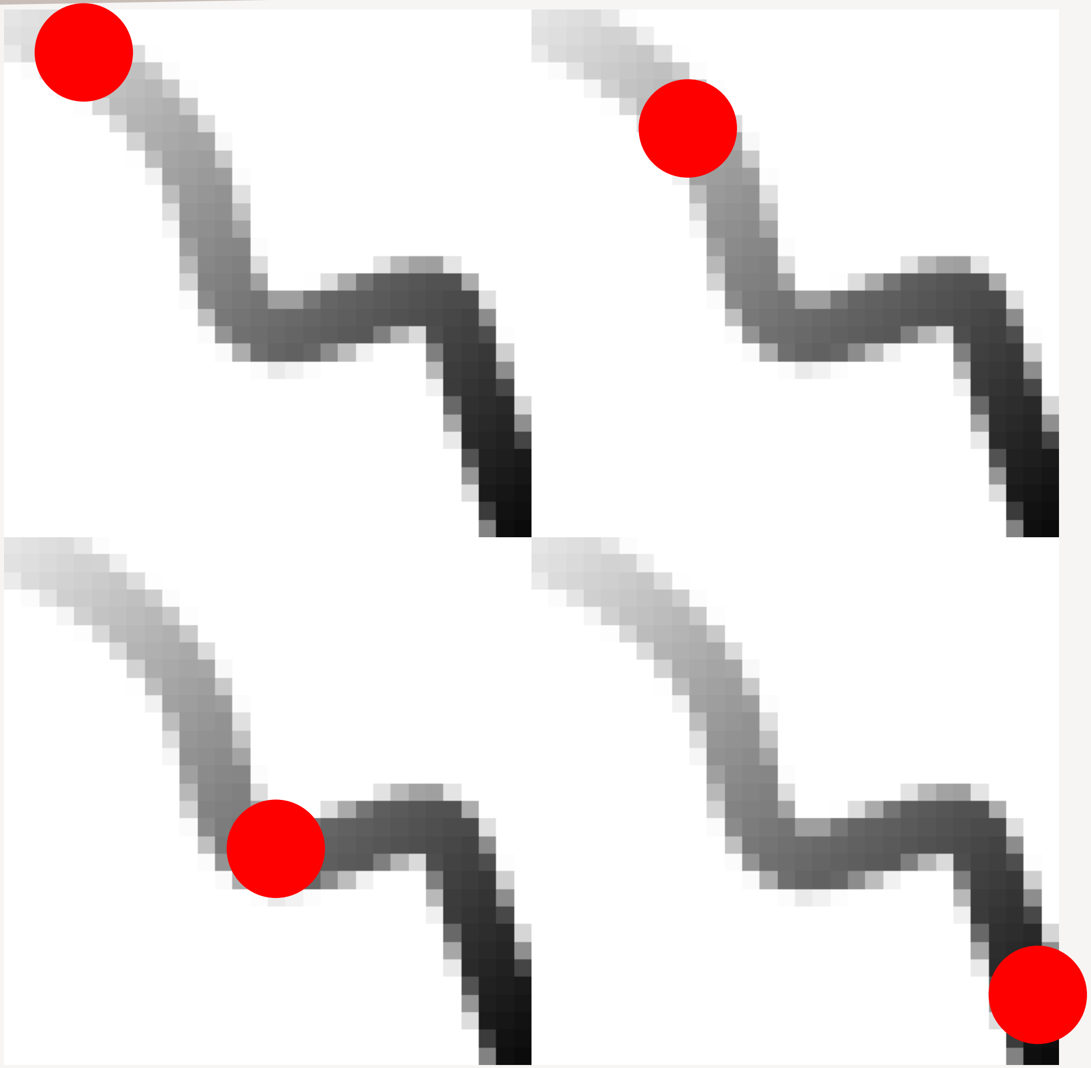


Nouvelle position

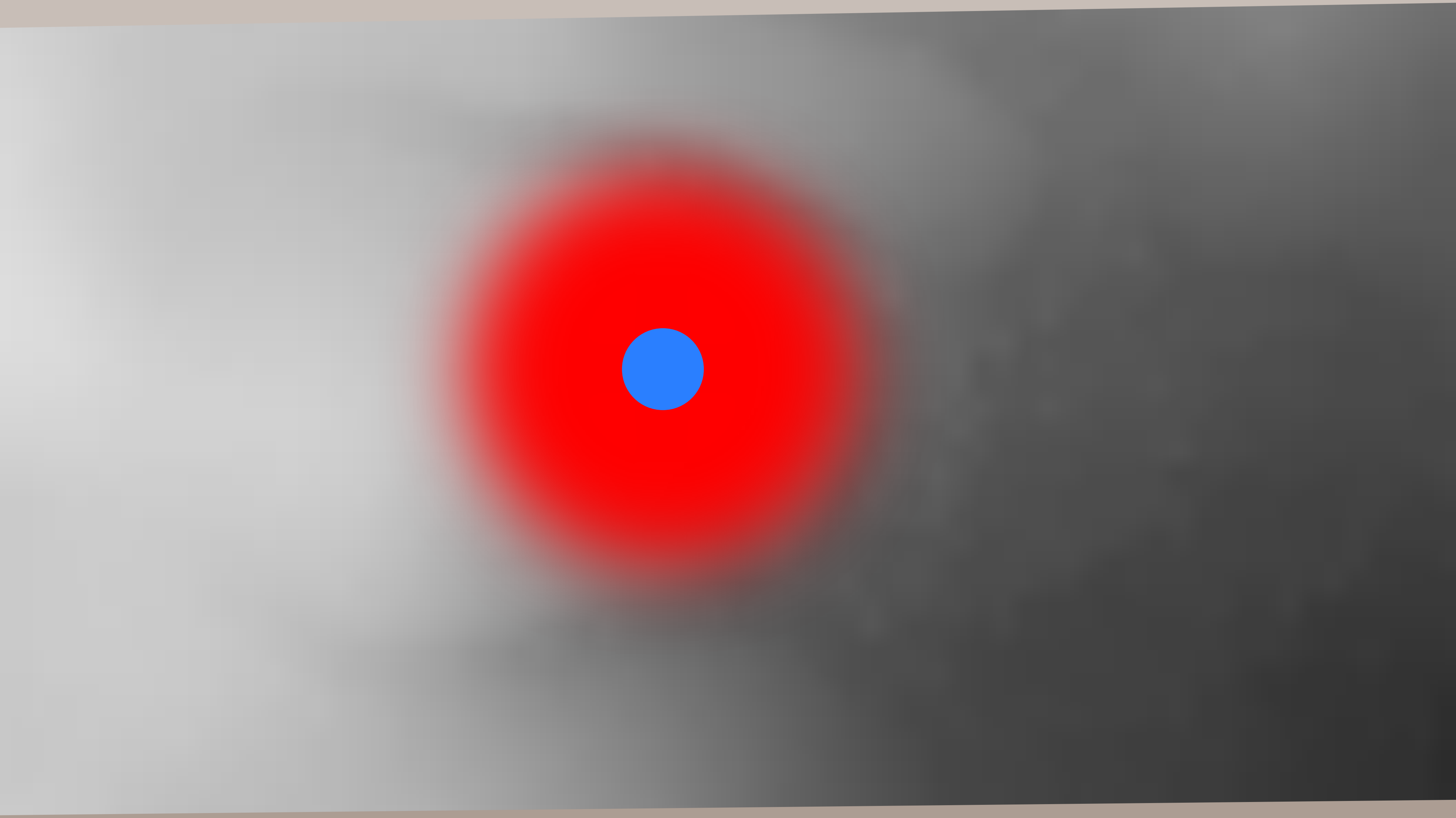
```
direction = (
    direction * inertia
    - g.normalized() * (1 - inertia)
).normalized()

position += direction
```

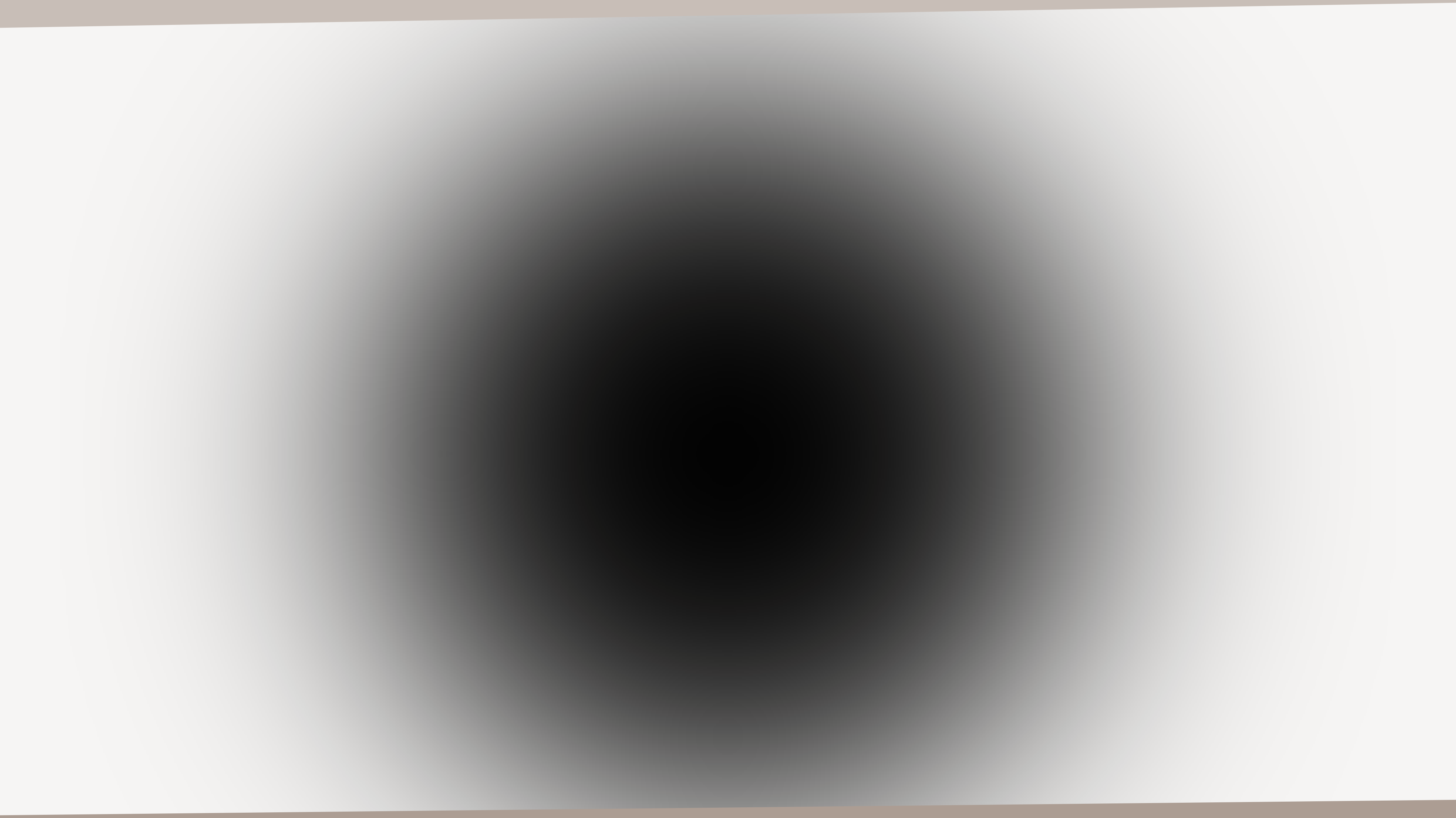
Test



Érosion



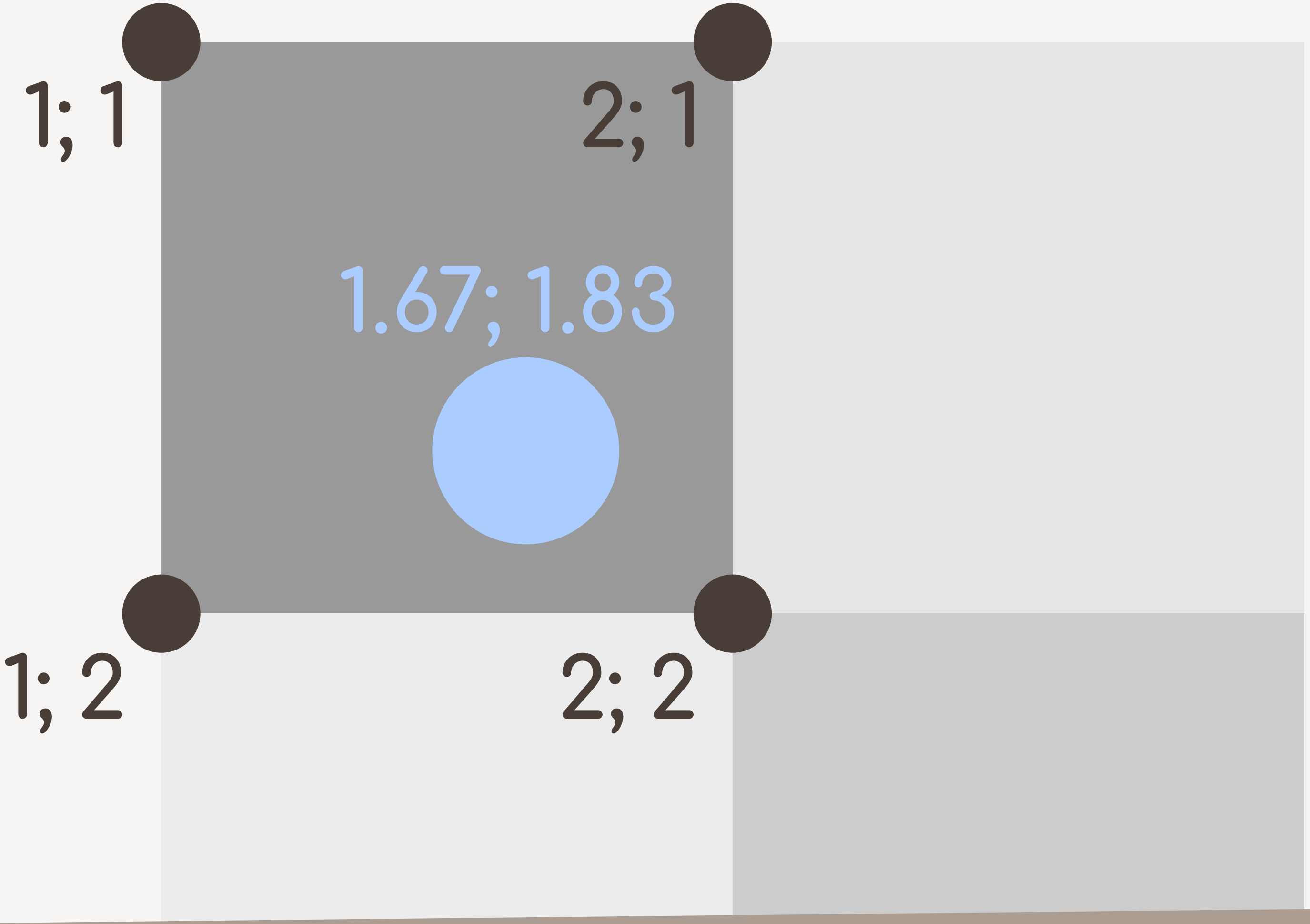
Érosion



Érosion

```
var weight := 1 - sqrt(sqr_dist) / (radius+1)
```

Dépôt de sédiment



Capacité

```
var delta_height = new_h - old_h

var carry_capacity =
max(-delta_height, min_slope)
* water * capacity_factor * velocity
```

L'eau

```
water *= (1 - evaporation)
```

evaporation $\in [0; 1]$

Vélocité

$$vel_{new} = \sqrt{vel_{old} - \Delta height * gravity}$$

On dépose, on érode ?

delta_height > 0.0 ou
sediment > carry_capacity

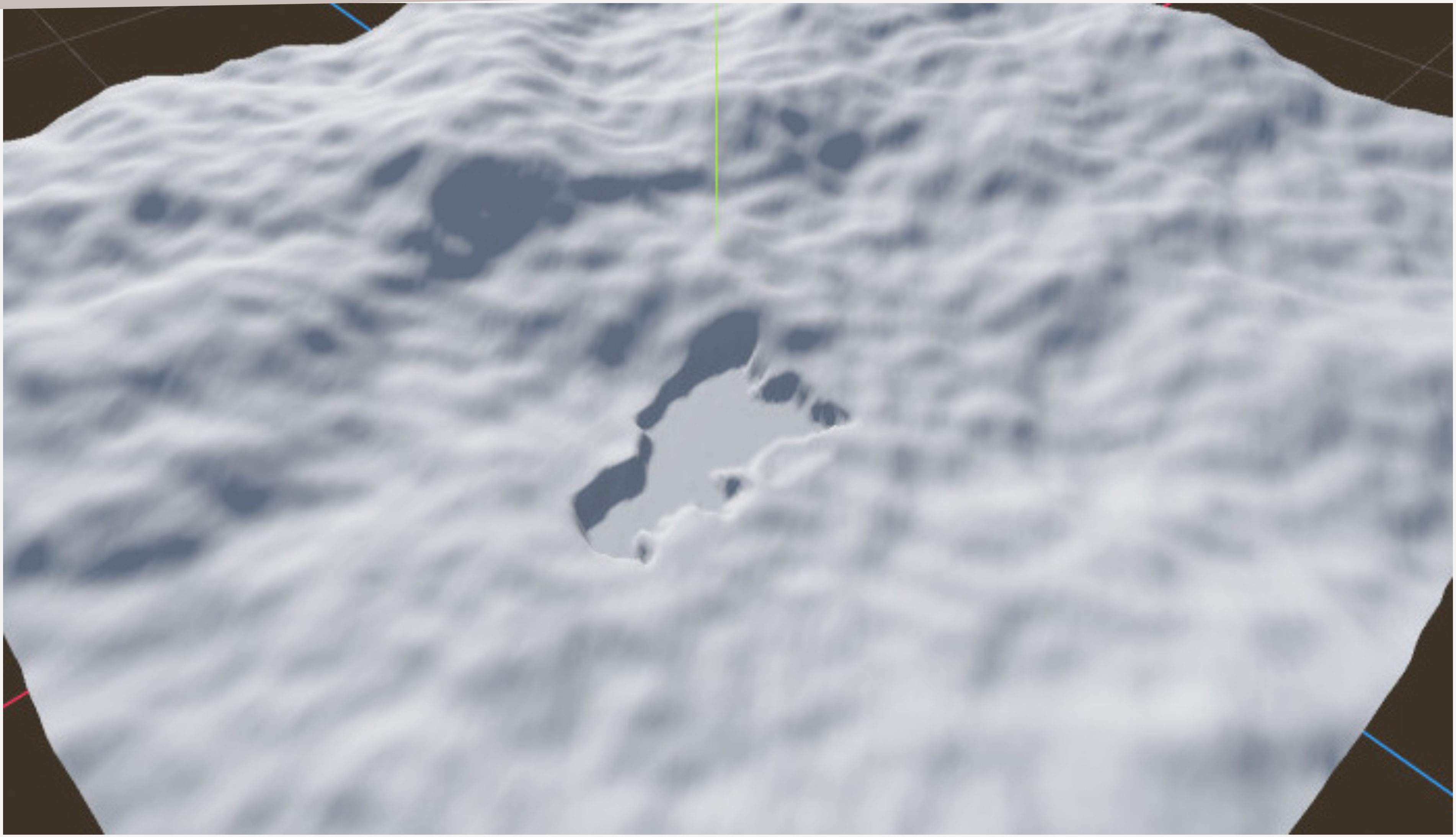
on dépose
sinon, on érode

Attention

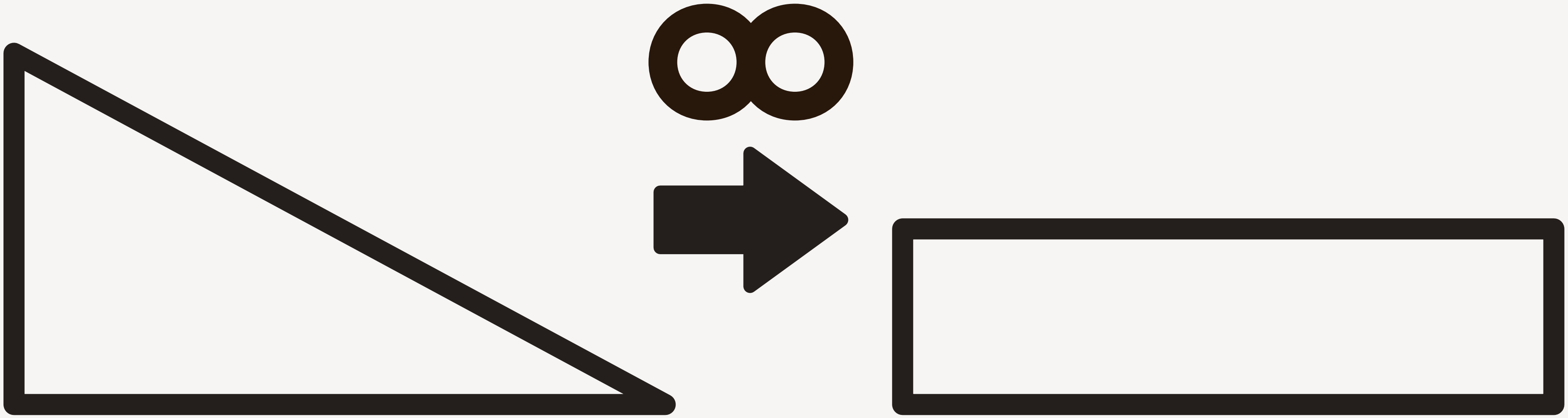
ne pas trop éroder

ne pas trop déposer

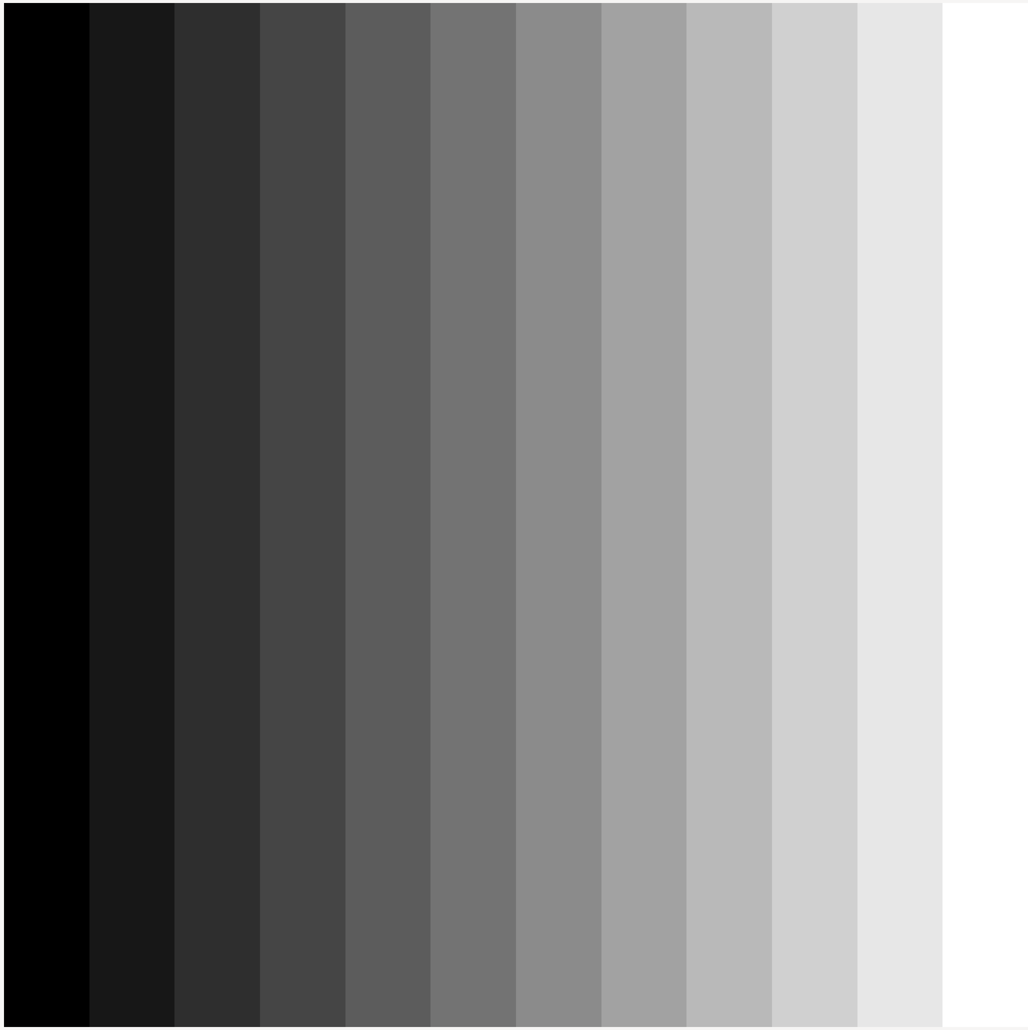
Test



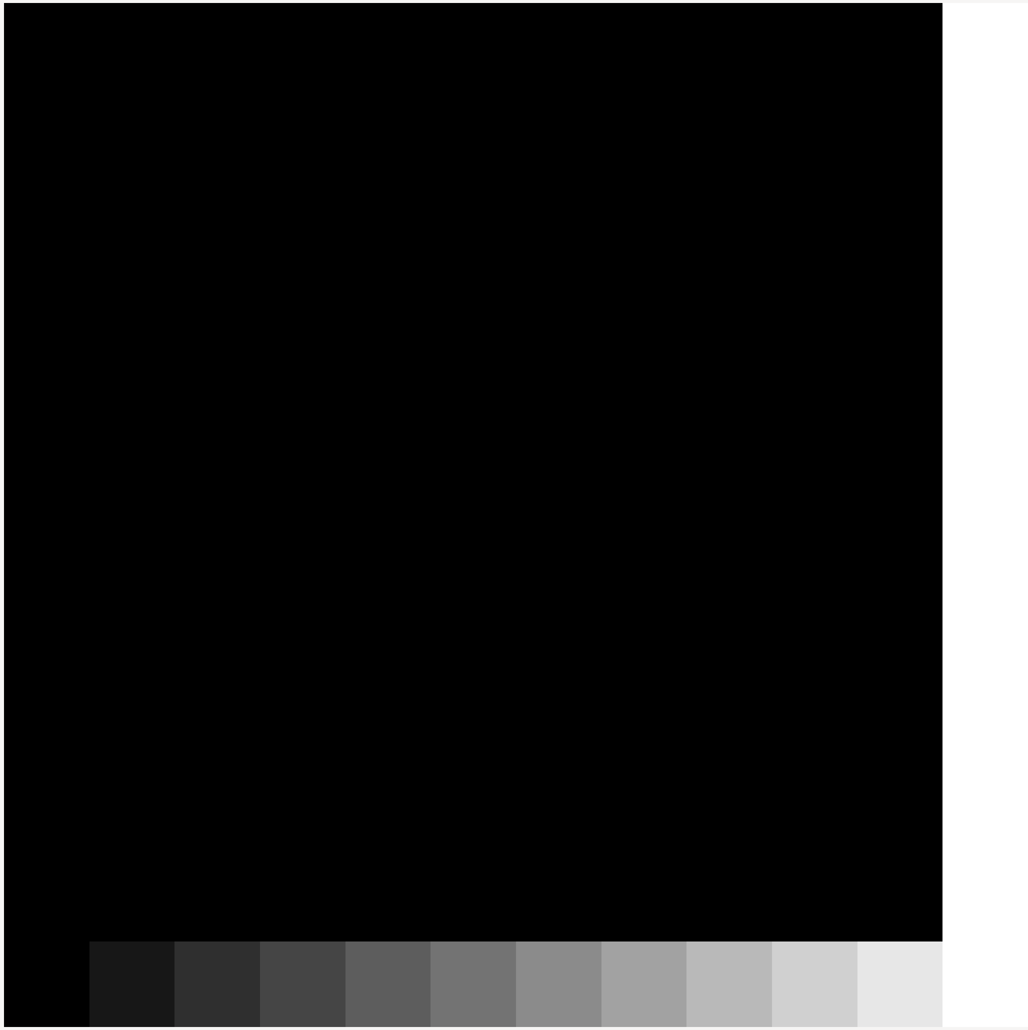
Enquête



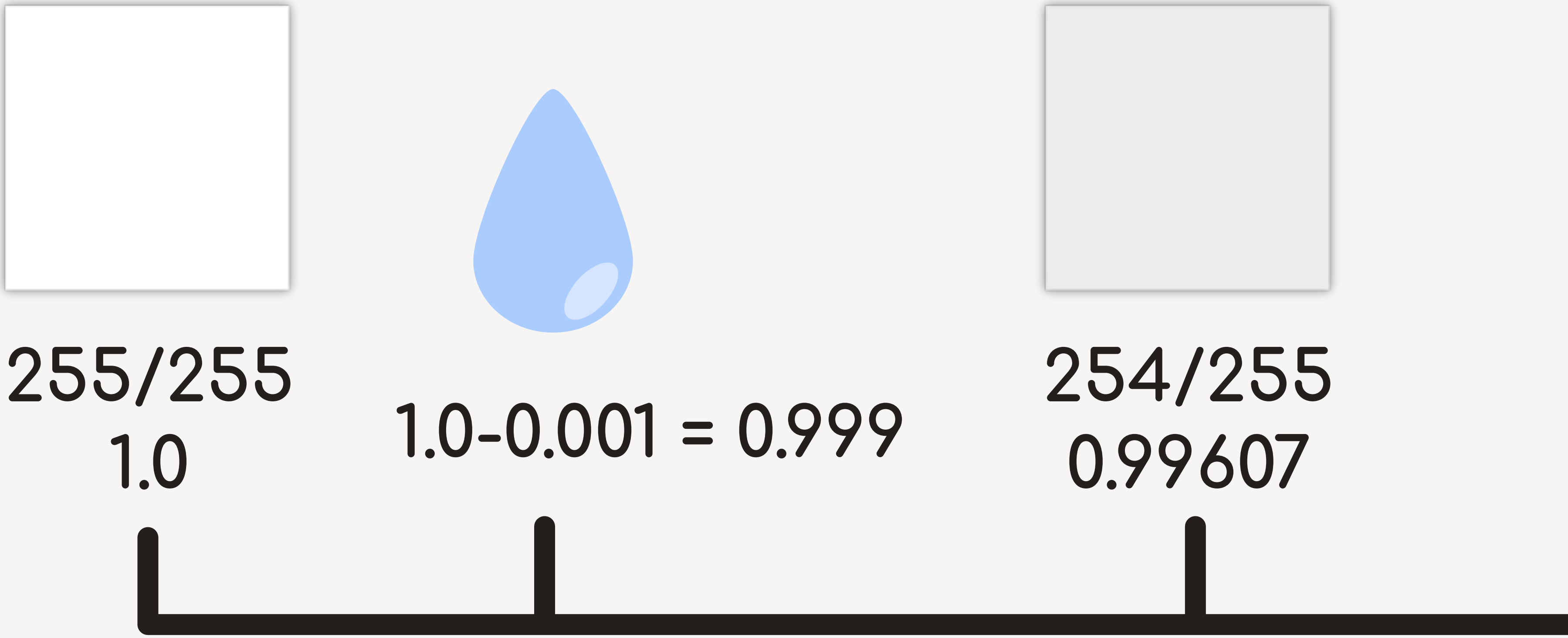
Enquête



Enquête



Problème

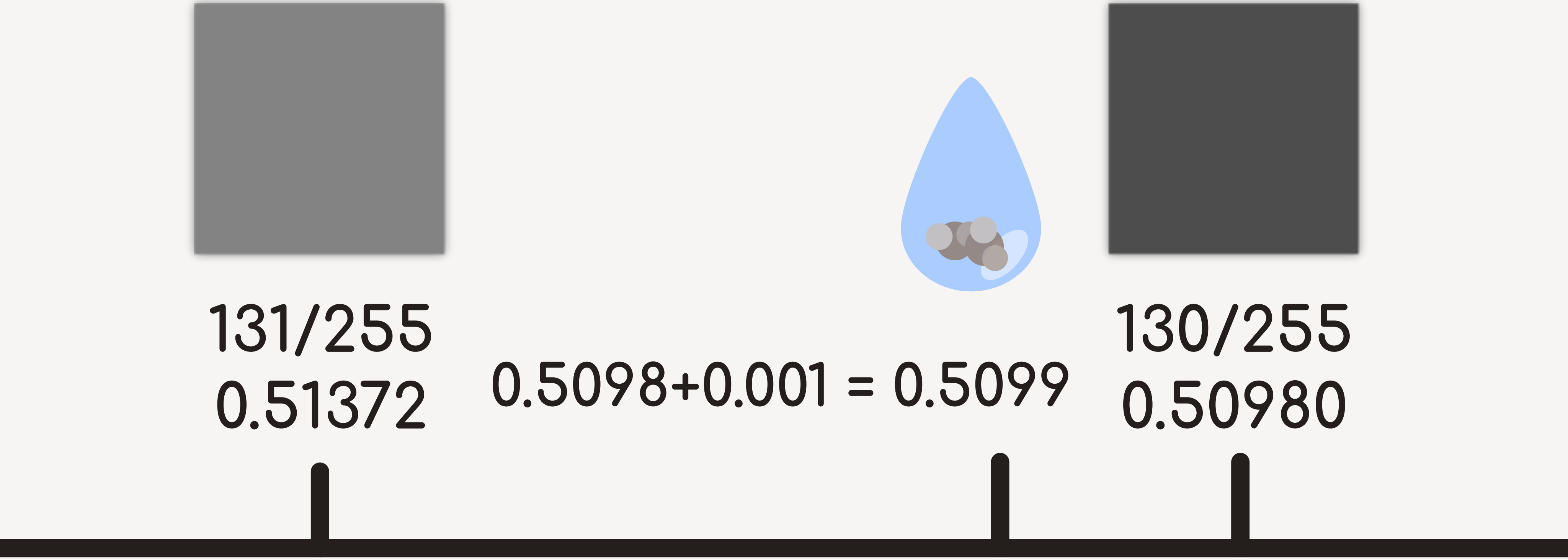


255/255
1.0

1.0-0.001 = 0.999

254/255
0.99607

Gros problème



131/255
0.51372

0.5098+0.001 = 0.5099

130/255
0.50980

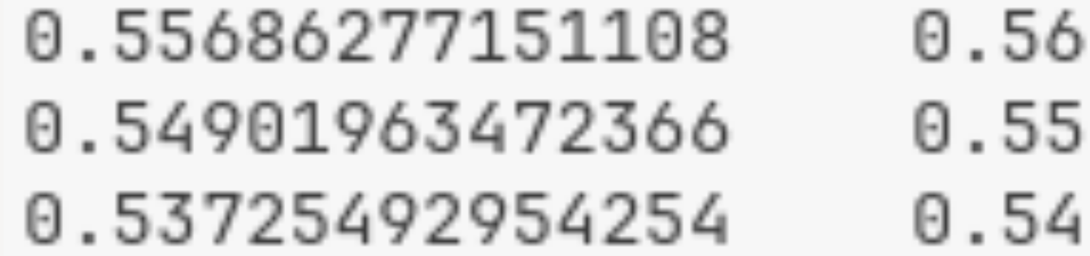
Solution

Image en 32 bits

2^8=256 2^{32}=4 294 967 296

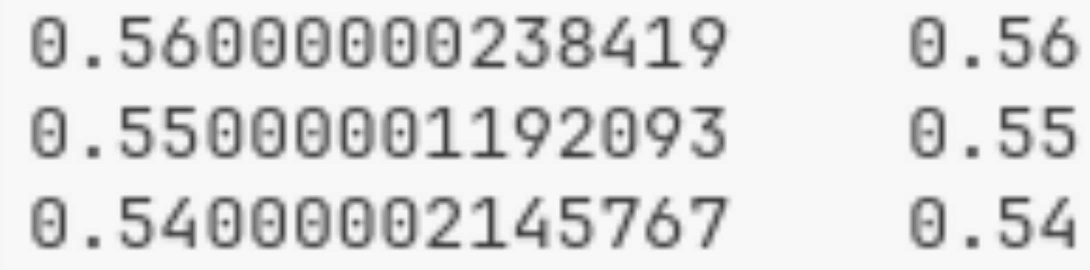
Solution

8 bits



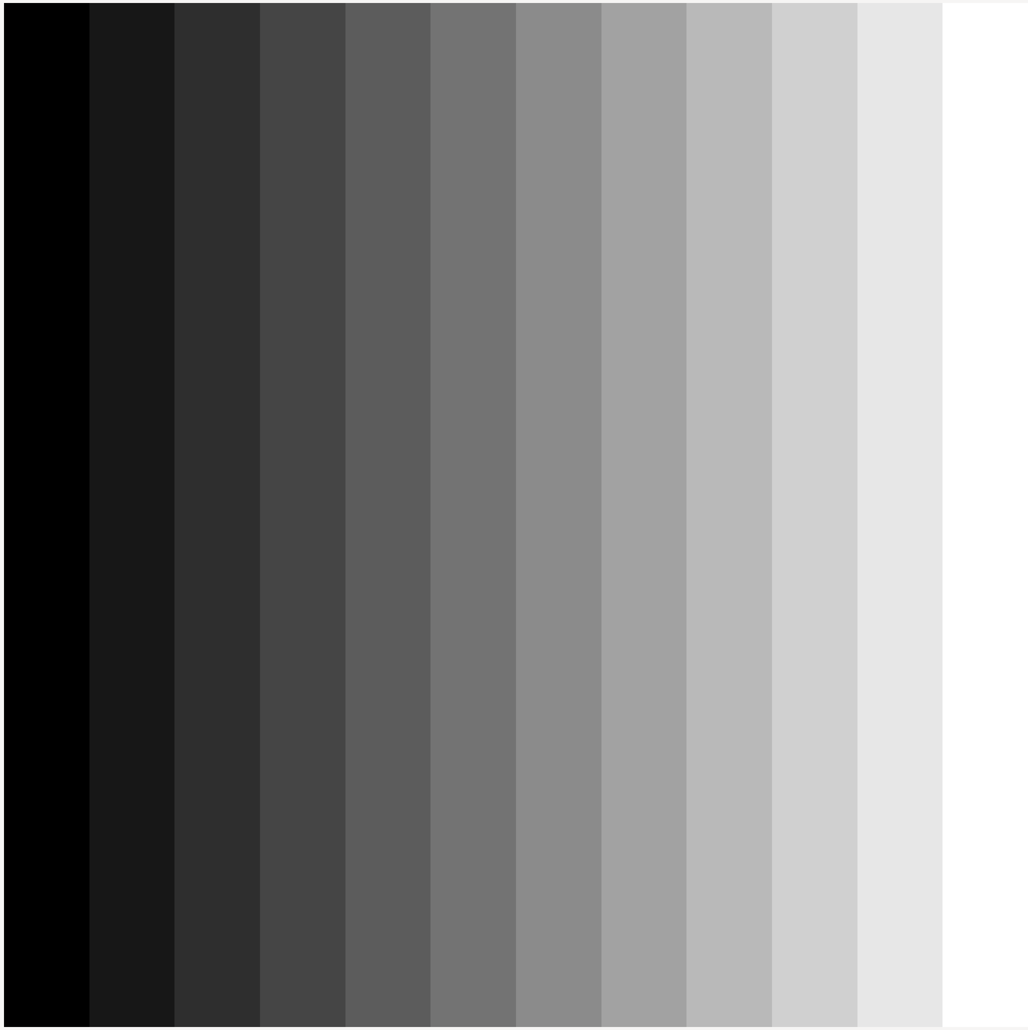
```
0.55686277151108     0.56
0.54901963472366     0.55
0.53725492954254     0.54
```

32 bits

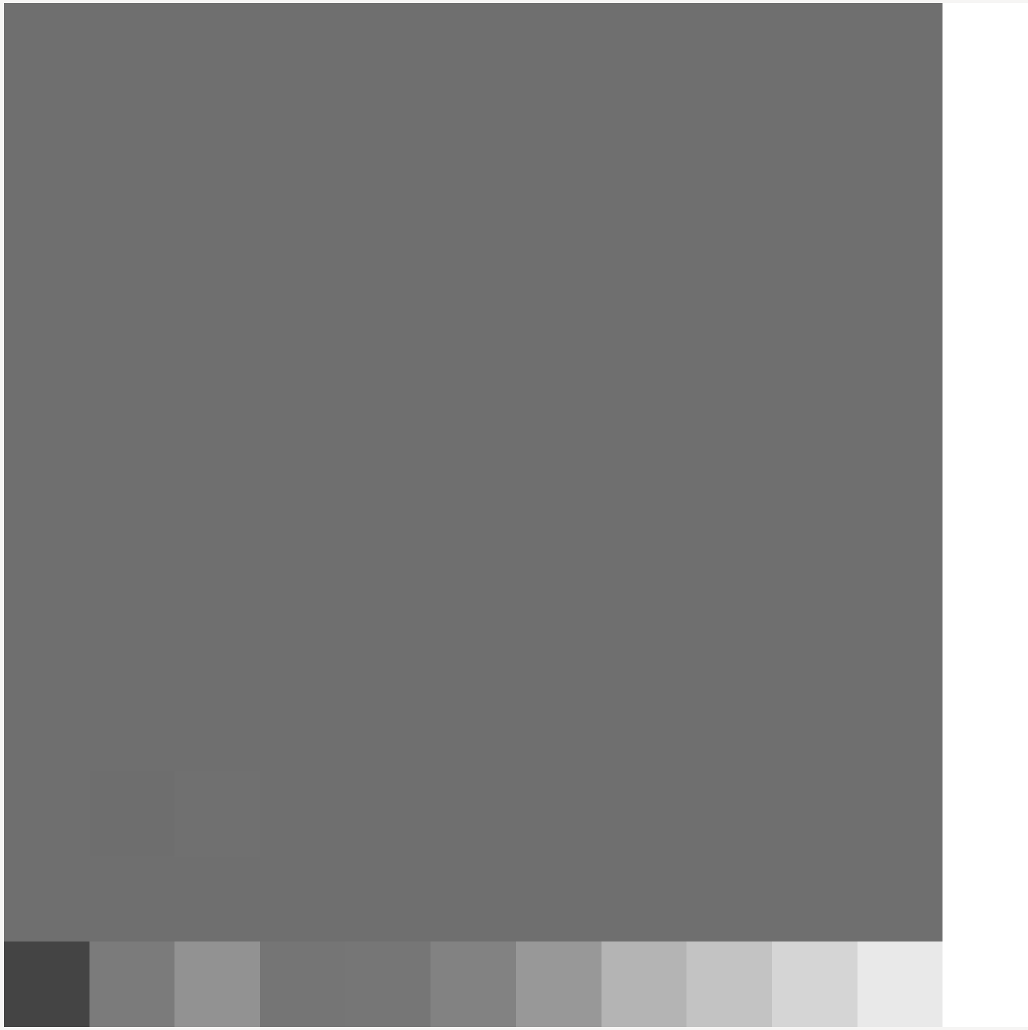


```
0.56000000238419     0.56
0.55000001192093     0.55
0.54000002145767     0.54
```

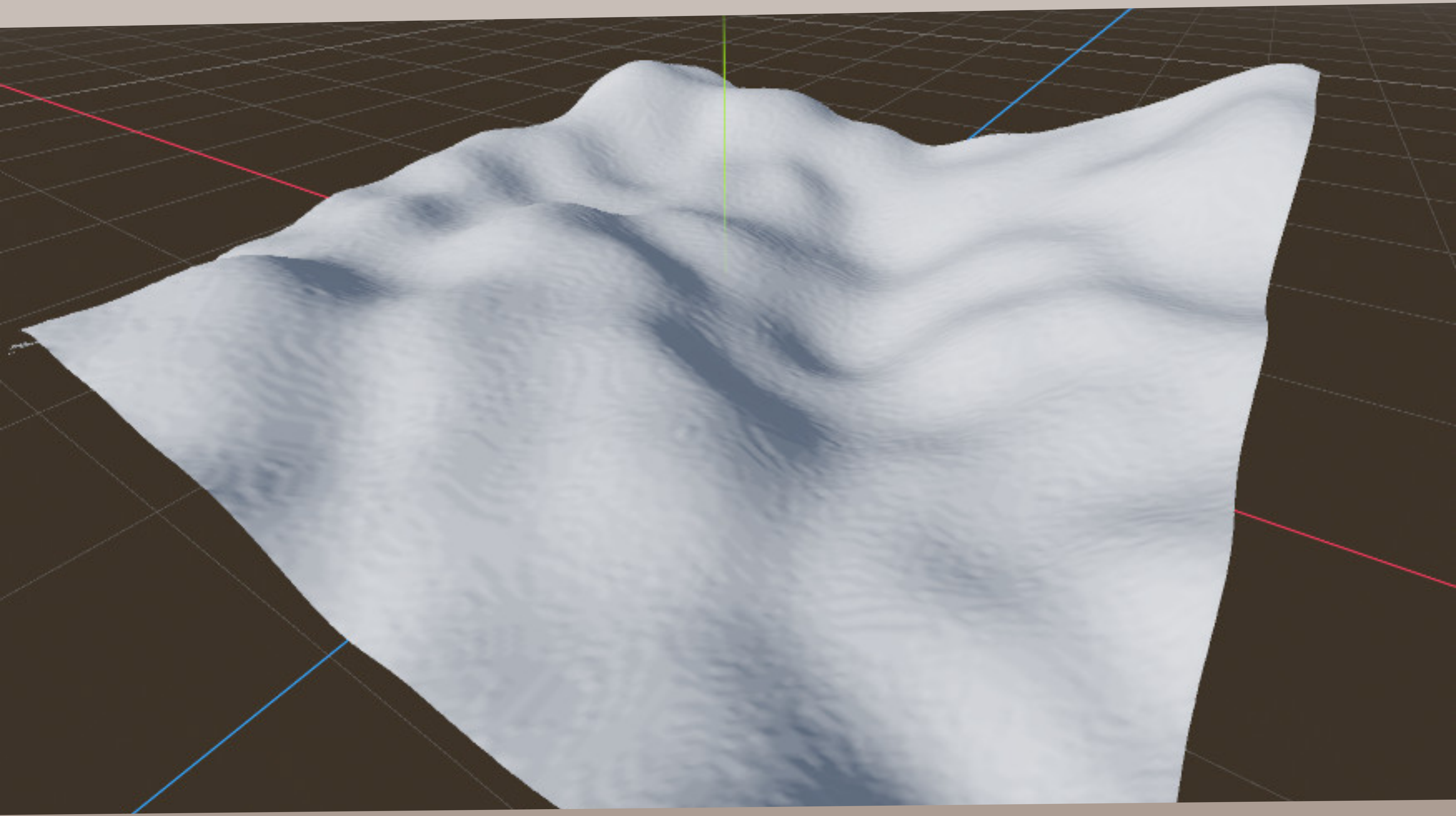
Vérification



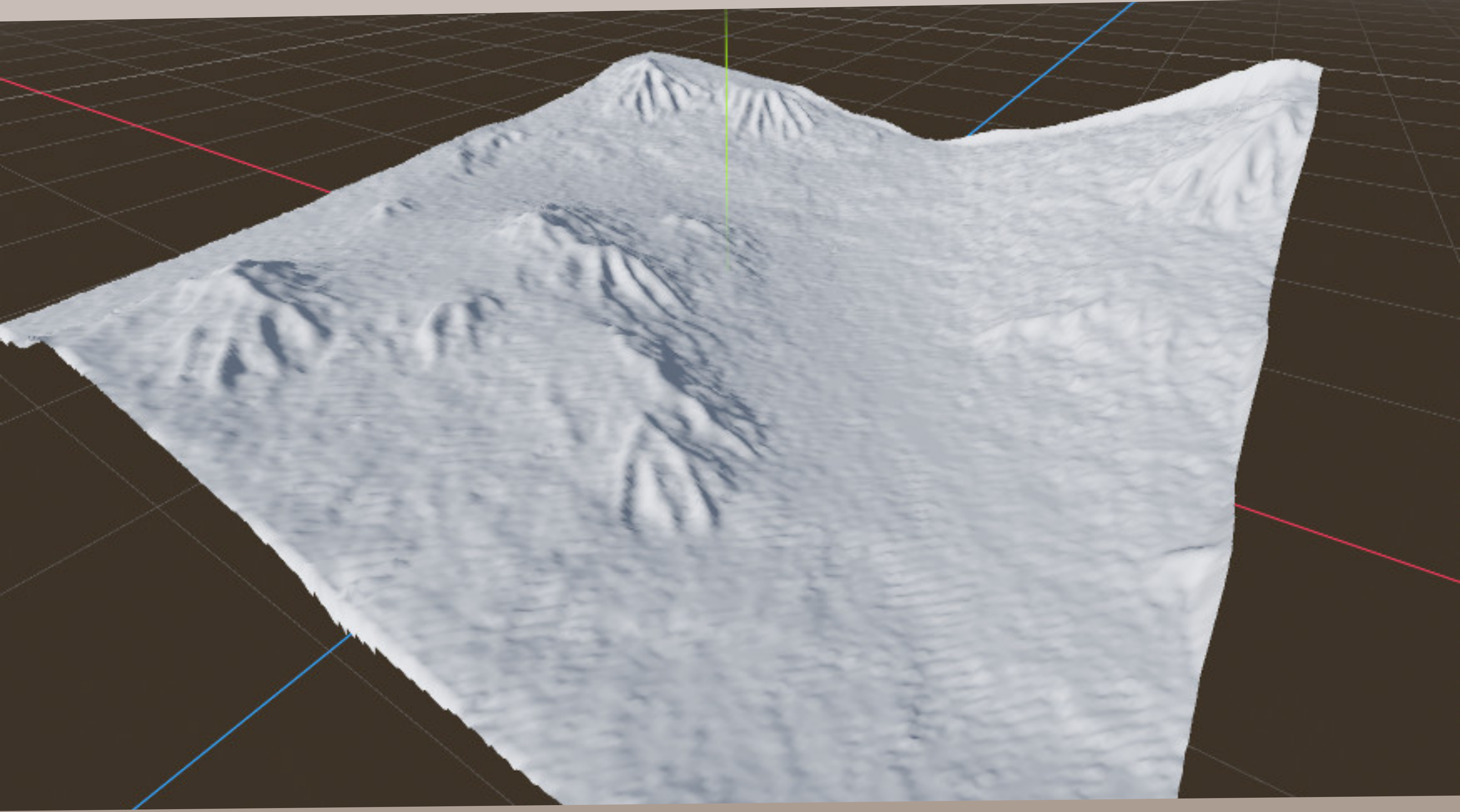
Vérification



Visuel 3D



Visuel 3D



Annexe

```
extends RefCounted
class_name Droplet

const MAX_ITERATIONS := 600

static var movements_image: Image
static var image: Image

var position: Vector2 = Vector2.ZERO

var direction: Vector2 = Vector2(0, 0)
var inertia: float = 0.1
var velocity: float = 1.0
var gravity: float = 0.098

var min_slope: float = 0.01

var sediment: float = 0.0
var capacity: float = 3.0
var erosion: float = 0.03
var deposition: float = 0.03

var water: float = 1.0
var evaporation: float = 0.1

var radius: int = 1

var iteration := 0

static var brush_weights: Dictionary = {}

static func generate_weights(radius: int):
  print("Start generating")
  var weights := {}
  var weight_sum = 0

  for j in range(-radius, radius+1):
    for i in range(-radius, radius+1):
      var pos := Vector2i(i, j)
      var sqr_dist := pos.length_squared()
      if sqr_dist > radius**2:
        continue

      var weight := 1 - sqrt(sqr_dist) / (radius+1)

      weight_sum += weight
      weights[pos] = weight

  var calculated_weights := {}
  for pos in weights:
    calculated_weights[pos] = weights[pos] / weight_sum

  Droplet.brush_weights = calculated_weights
  print(Droplet.brush_weights)
  print("Generated")
```

```
func move():
  movements_image.set_pixelv(position, Color(.0, .0, .0 ,.0))

  if out_of_bounds():
    return

  var Pxy : float =  image.get_pixelv(position).r
  var Px1y: float =  image.get_pixelv(position + Vector2.RIGHT).r
  var Pxy1: float =  image.get_pixelv(position + Vector2.DOWN).r
  var Px1y1: float = image.get_pixelv(position + Vector2.ONE).r

  # u = uv.x
  # v = uv.y
  # values between [0, 1[
  var uv: Vector2 = position - Vector2(Vector2i(position))

  var g := Vector2(
    (Px1y - Pxy) * (1 - uv.y) + (Px1y1 - Pxy1) * uv.y,
    (Pxy1 - Pxy) * (1 - uv.x) + (Px1y1 - Px1y) * uv.x
  )

  direction = (direction * inertia - g.normalized() * (1 - inertia)).normalized()
  position += direction

  if not out_of_bounds():
    movements_image.set_pixelv(position, Color.GREEN)

func update():
  if iteration > MAX_ITERATIONS:
    die()
    return false

  iteration += 1

  var old_position = position
  var old_height = image.get_pixelv(old_position).r
  move()

  if out_of_bounds():
    position = old_position
    die()
    return false


  var new_height = image.get_pixelv(position).r
  var delta_height = new_height - old_height
  if old_position == position:
    die()
    return false

  var carry_capacity = max(-delta_height, min_slope) * water * capacity * velocity
  if delta_height > 0.0 or sediment > carry_capacity:
    var amount_to_depose: float = min(delta_height, sediment) if delta_height > 0
                                                    else (sediment - carry_capacity) * deposition
    depose(amount_to_depose, old_position)
  else:
    var amount_to_erode: float = min(-delta_height, (carry_capacity-sediment) * erosion)
    erode(amount_to_erode, old_position)

  velocity = sqrt(max(0, velocity**2 - delta_height * gravity))
  water *= (1 - evaporation)

  return true
```

```
func die():
  movements_image.set_pixelv(position, Color(0.0, 0.0, 0.0, 0.0))
  depose(sediment, position)

func erode(amount: float, old_position: Vector2):
  var i_old_position := Vector2i(old_position)
  for offset in brush_weights.keys():
    var sub_position := Vector2i(offset) + i_old_position
    if sub_position.x < 0 or sub_position.y < 0 or sub_position.x >= image.get_width() or sub_position.y >= image.get_height():
      continue

    var previous_amount := image.get_pixelv(sub_position).r
    var weighed_amount = amount * brush_weights[offset]
    var delta_amount = previous_amount if previous_amount < weighed_amount else weighed_amount

    sediment += delta_amount

    var new_color = Color(previous_amount - delta_amount, previous_amount - delta_amount, previous_amount - delta_amount, 1.0)
    image.set_pixelv(sub_position, new_color)

func depose(amount: float, old_position: Vector2):
  var uv: Vector2 = old_position - Vector2(Vector2i(old_position))

  var xy = Vector2i(old_position)
  var x1y = Vector2i(old_position) + Vector2i.RIGHT
  var xy1 = Vector2i(old_position) + Vector2i.DOWN
  var x1y1 = Vector2i(old_position) + Vector2i.RIGHT + Vector2i.DOWN

  var xy_amount = amount * (1 - uv.x) * (1 - uv.y)
  var x1y_amount = amount * uv.x * (1 - uv.y)
  var xy1_amount = amount * (1 - uv.x) * uv.y
  var x1y1_amount = amount * uv.x * uv.y

  depose_pixel(xy, xy_amount)
  depose_pixel(x1y, x1y_amount)
  depose_pixel(xy1, xy1_amount)
  depose_pixel(x1y1, x1y1_amount)

func depose_pixel(pixel_pos: Vector2i, amount: float):
  var previous_amount = image.get_pixelv(pixel_pos).r
  var deposed_amount = previous_amount + amount
  sediment -= amount
  image.set_pixelv(pixel_pos, Color(deposed_amount, deposed_amount, deposed_amount, 1.0))

func out_of_bounds() -> bool:
  return position.x < 0 or position.y < 0 or position.x >= image.get_width()-1 or position.y >= image.get_height()-1
```

```
extends Sprite2D

const N = 10_000

@export var timer: Timer
@export var movements: Sprite2D
@onready var base_texture: Texture2D = texture

var droplets: Array[Droplet] = []
var total_droplets: int = 0

func _ready() -> void:
  randomize()
  await base_texture.changed

  var movements_image = Image.create(base_texture.get_width(), base_texture.get_height(), false, Image.FORMAT_RGBA8)
  var movements_image_texture: ImageTexture = ImageTexture.create_from_image(movements_image)

  var img = base_texture.get_image()
  var img32 := Image.create(img.get_width(), img.get_height(), false, Image.FORMAT_RGBF)
  for i in range(img.get_width()):
    for j in range(img.get_height()):
      img32.set_pixel(i, j, img.get_pixel(i, j))

  Droplet.movements_image = movements_image_texture.get_image()
  Droplet.image = img32
  movements.texture = movements_image_texture

  Droplet.generate_weights(2)

  timer.timeout.connect(update)

func _process(delta: float) -> void:
  update()
  for i in range(droplets.size(), 10_000):
    var rx = randf_range(0.0, base_texture.get_width()-1.0)
    var ry = randf_range(0.0, base_texture.get_height()-1.0)
    var droplet = Droplet.new(Vector2(rx, ry))
    droplets.append(droplet)
    total_droplets += 1

func _input(event: InputEvent) -> void:
  if event is InputEventMouseButton and event.pressed:
    var droplet = Droplet.new(get_local_mouse_position())
    droplets.append(droplet)
    total_droplets += 1

  if event is InputEventKey and event.pressed:
    if event.keycode == KEY_SPACE:
      for i in range(N):
        var rx = randf_range(0.0, base_texture.get_width()-1.0)
        var ry = randf_range(0.0, base_texture.get_height()-1.0)
        var droplet = Droplet.new(Vector2(rx, ry))
        droplets.append(droplet)
      total_droplets += N
    elif event.keycode == KEY_S:
      texture.get_image().save_jpg("res://results/after_%s_droplets.jpg" % total_droplets, 1.0)
```

```
func update():
  var q = len(droplets)
  var droplets_alive: Array[Droplet] = []
  for droplet: Droplet in droplets:
    var is_alive = droplet.update()
    if is_alive:
      droplets_alive.append(droplet)

  droplets = droplets_alive

  movements.texture = ImageTexture.create_from_image(Droplet.movements_image)
  texture = ImageTexture.create_from_image(Droplet.image)
```

```
@tool
extends Node3D

@export_tool_button("Update") var update = update_mountain
@export var max_height := 1.0
@export var texture: Texture2D

@onready var plane_3d: MeshInstance3D = $Plane3D

func update_mountain():
  build_plane()

func build_plane():
  var img = texture.get_image()
  if img.is_compressed():
    img.decompress()
    texture = ImageTexture.create_from_image(img)

  var plane := PlaneMesh.new()
  plane.size = texture.get_size()-Vector2.ONE
  plane.subdivide_width = texture.get_size().x-2
  plane.subdivide_depth = texture.get_size().y-2

  var mesh = ArrayMesh.new()
  mesh.add_surface_from_arrays(Mesh.PRIMITIVE_TRIANGLES, plane.get_mesh_arrays())
  var mdt = MeshDataTool.new()
  mdt.create_from_surface(mesh, 0)
  for i in range(mdt.get_vertex_count()):
    var vertex: Vector3 = mdt.get_vertex(i)
    var tex_position := Vector2(vertex.x, vertex.z) + texture.get_size() / 2

    vertex.y = max_height * texture.get_image().get_pixelv(tex_position).r
    mdt.set_vertex(i, vertex)

  mesh.clear_surfaces()
  mdt.commit_to_surface(mesh)

  var st = SurfaceTool.new()
  st.create_from(mesh, 0)
  st.generate_normals()
  st.generate_tangents()

  plane_3d.mesh = st.commit()
```