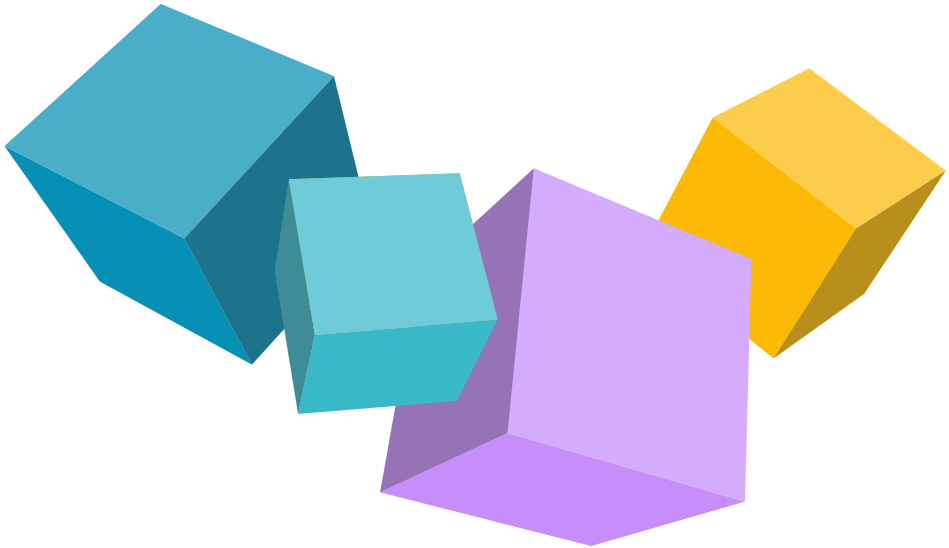
Özkan



LANDSCHAFTS-GENERATOR

Datenverarbeitung in der Medienproduktion

Gruppe 1
Özkan Y. - Justin E. - Sebastian P. - Raphael H.

GLIEDERUNG

1. PROJEKT

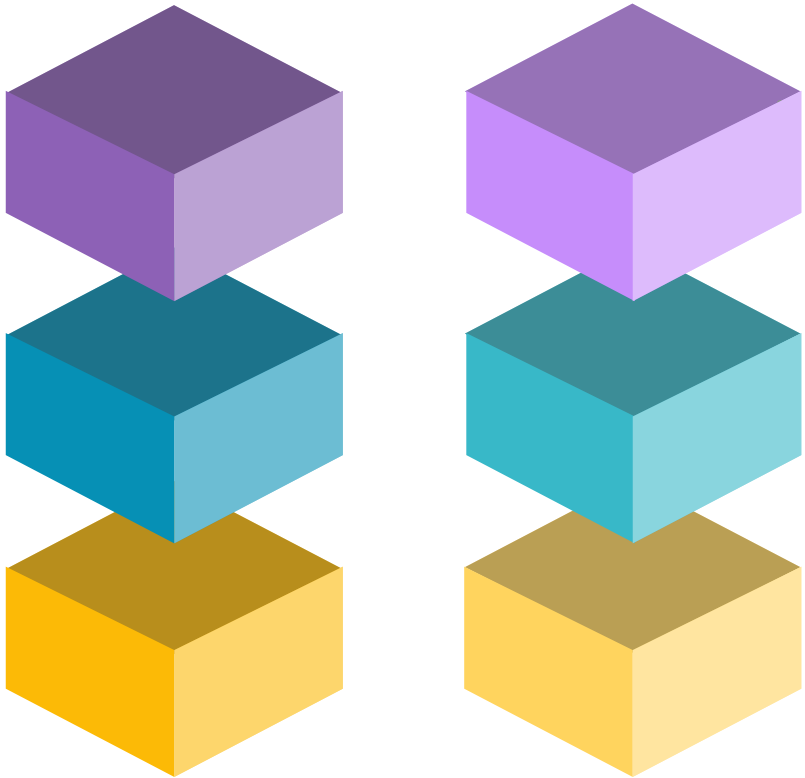
Was unser Projekt als Ziele gesetzt hatte und die Aufgabenverteilung

2. LANDSCHAFT

Generierung der Landschaft

3. BÄUME

Generierung von Bäumen



4. STEINE

Generierung von Steinen

5. BELEUCHTUNG

Generierung der Landschafts-Beleuchtung

6. FAZIT

Schwierigkeiten und weiteres Vorgehen

PROJEKT

Ziele

MUST HAVES

- ➔ Verschiedene Elemente in die Szene setzen
- ➔ Zufällige Dimensionen für die Elemente
- ➔ Verschiedene Materiale

NICE TO HAVE

- ➔ Steine
- ➔ Beleuchtung
- ➔ Berge oder Hügel
- ➔ Kreaturen und Lebewesen
- ➔ Flüsse

USER INTERFACE

- ➔ Zahlenfelder
- ➔ Schieberegler
- ➔ Toggles

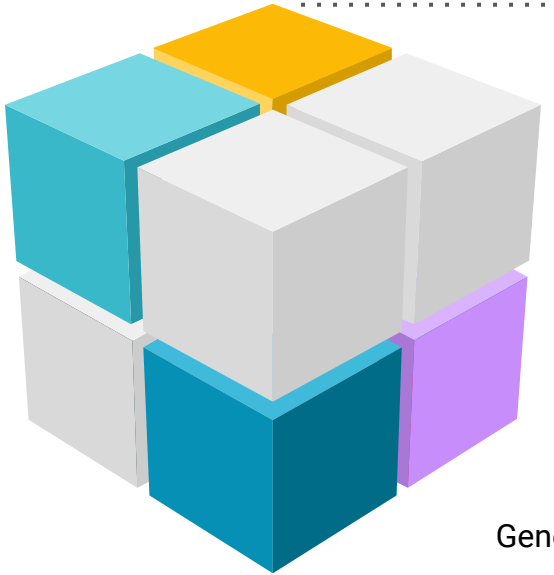
EINSTELLBARE PARAMETER

- ➔ Anzahl
- ➔ Größenskala

Justin

PROJEKT

Aufgabenverteilung



ÖZKAN

Generierung von Bäumen

- ➔ gebietsabhängig
- ➔ attributsabhängig

SEBASTIAN

Skybox

- ➔ Beleuchtungsquelle
- ➔ Skybox

JUSTIN

Generierung von Steinen

- ➔ verschiedene Größen
- ➔ attributsabhängig

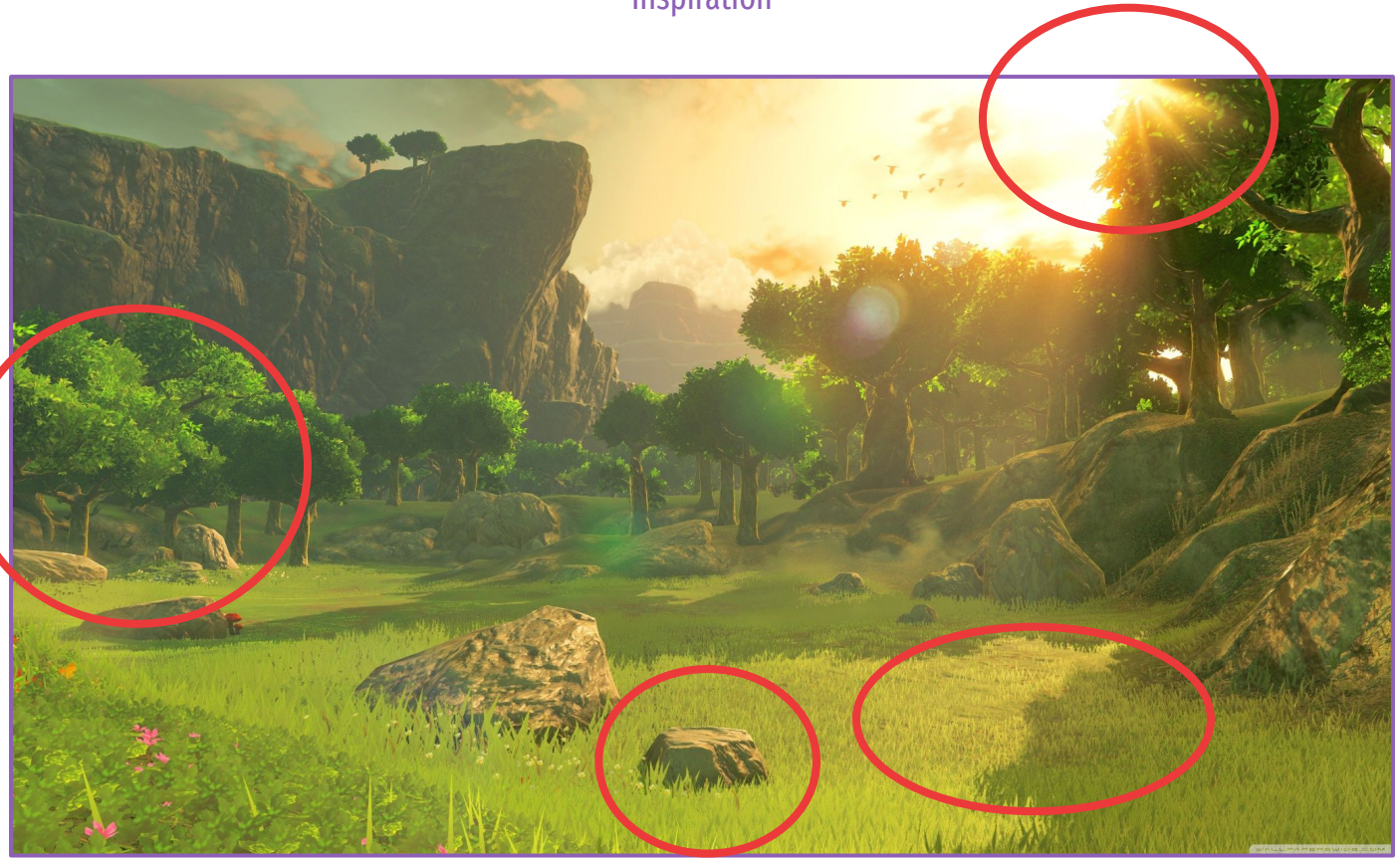
RAPHAEL

Generierung von Ebenen

- ➔ mit div. Parametern (Höhe/ Tiefe, etc.)
- ➔ mit räumlicher Begrenzung (T/ B/ H)
- ➔ Gras-Generierung

PROJEKT

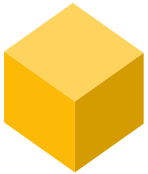
Inspiration



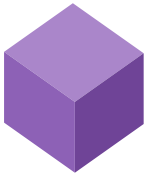
Raphael

GENERIERUNG VON EBENEN

Problemstellung



➔ mit div. Parametern (Höhe/ Tiefe, etc.)

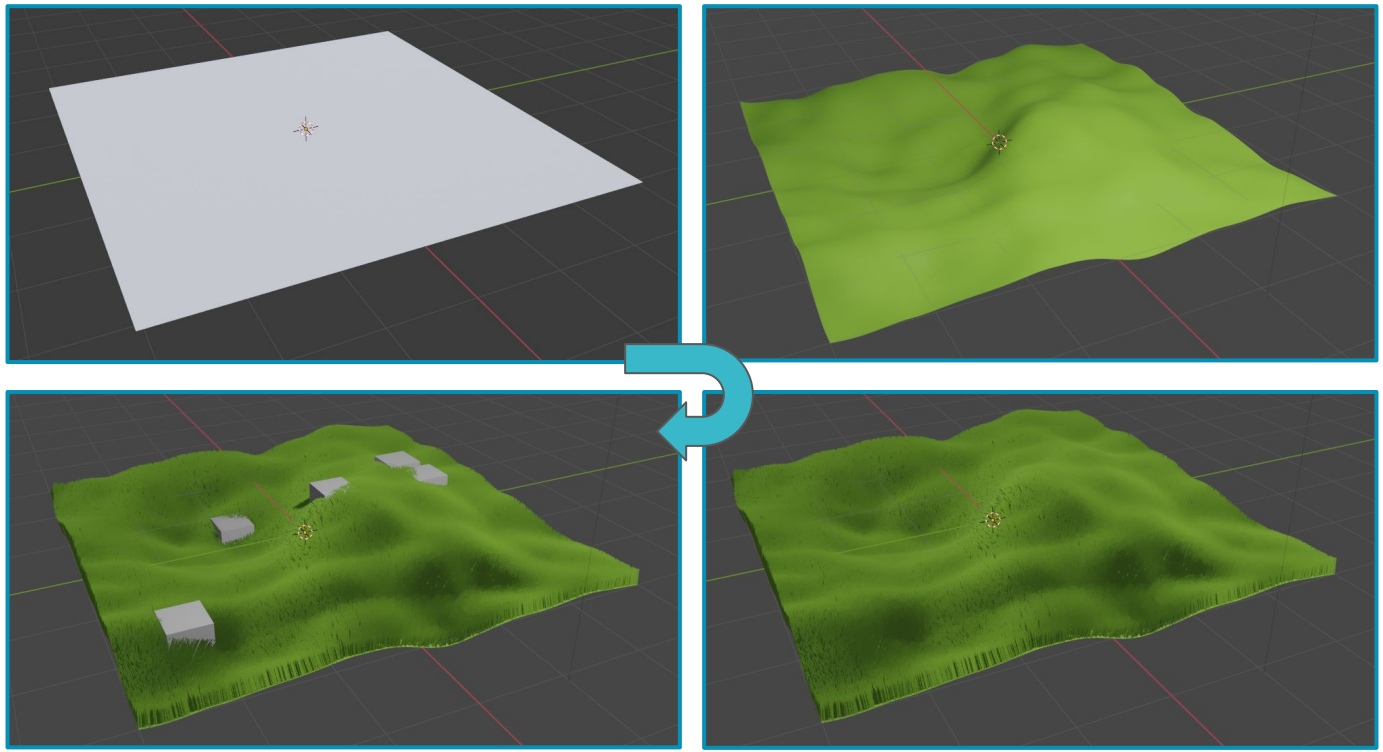


➔ mit räumlicher Begrenzung (T/ B/ H)

➔ Gras-Generierung

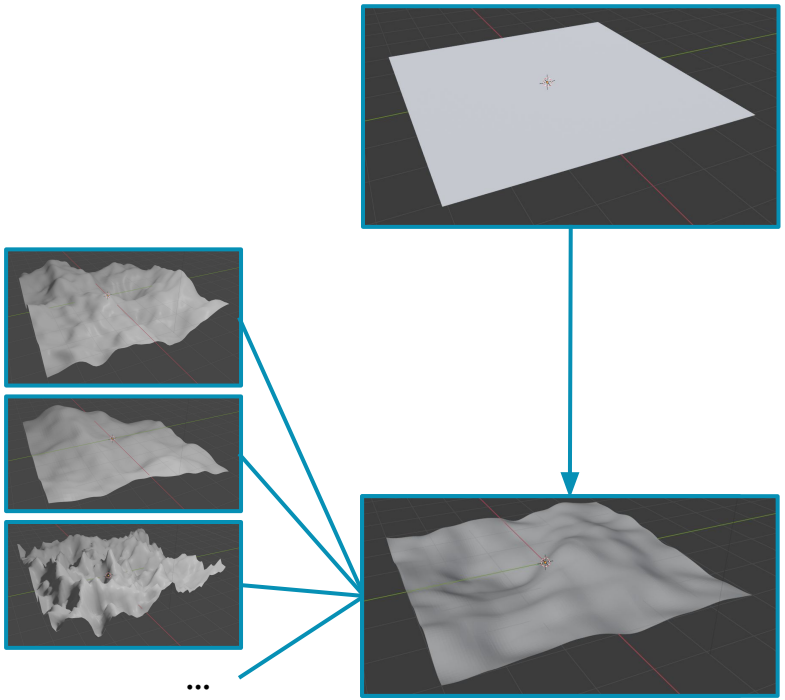
GENERIERUNG VON EBENEN

Herangehensweise



GENERIERUNG VON EBENEN

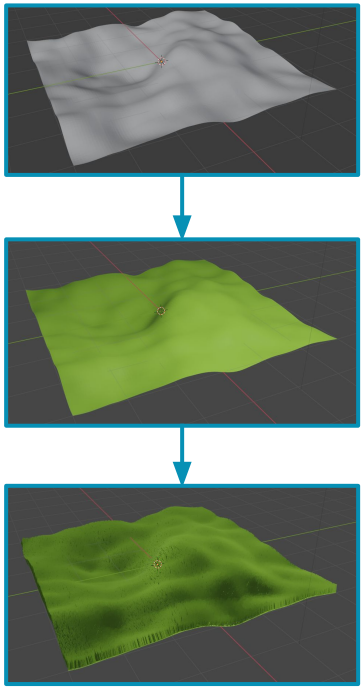
Subdivision und Displacement

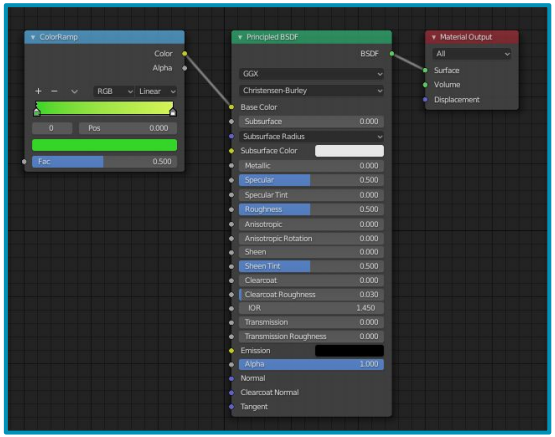



```python
# 2. Add Subdivision Surface Modifier
    context = bpy.context
    ob = context.object
    me = obj.data
    bm = bmesh.new()
    bm.from_mesh(me)

    # subdivide
    bmesh.ops.subdivide_edges(bm,
                              edges=bm.edges,
                              cuts=100,
                              use_grid_fill=True,
                              )

    bm.to_mesh(me)
    me.update()

# 3. Add Displacement Modifier
    dispMod = obj.modifiers.new("Displace", type='DISPLACE')

    tex = bpy.data.textures.new('CloudNoise', type = 'CLOUDS')
    tex.noise_scale = 1.00
    tex.noise_basis = 'ORIGINAL_PERLIN'
    tex.cloud_type = 'COLOR'
    tex.contrast = 1.050
    tex.saturation = 0.97
    tex.intensity = 1.00

    dispMod.texture = tex

    #Apply Modifier
    #TODO Apply erst am Ende?
    bpy.ops.object.modifier_apply(modifier = dispMod.name)
```

GENERIERUNG VON EBENEN

Shader und Particle System (Gras)



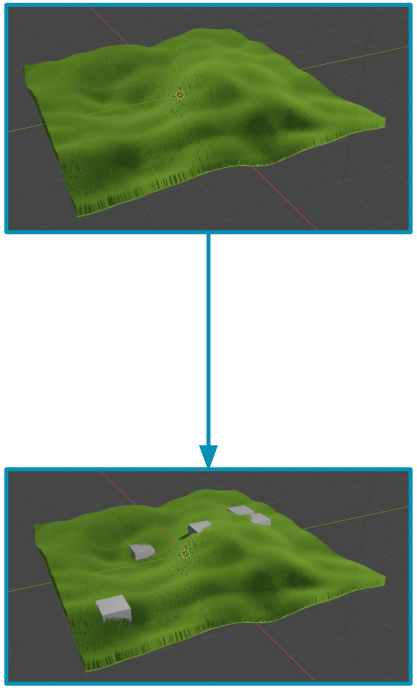



```python
# 6. Create (Node-) Material for Terrain-Plane
    terrain_material = bpy.data.materials.new(name="TerrainMaterial")
    terrain_material.use_nodes = True
    obj.active_material = terrain_material

    nodes = terrain_material.node_tree.nodes

    color_ramp = terrain_material.node_tree.nodes.new("ShaderNodeValToRGB")
    color_ramp.location = (-300, 300)

    color_ramp.color_ramp.elements.new(0.0)
    color_ramp.color_ramp.elements[0].color = (0.035, 0.666 , 0.022, 1)

    color_ramp.color_ramp.elements[1].position = (1.0)
    color_ramp.color_ramp.elements[1].color = (0.662, 0.904, 0.098, 1)

    principled_bsdf = nodes.get("Principled BSDF")
    terrain_material.node_tree.links.new(principled_bsdf.inputs["Base Color"], color_ramp.outputs["Color"])
```

```python
# 5. Add a particle system ("Grass")
    psys = obj.modifiers.new("hair", 'PARTICLE_SYSTEM').particle_system
    psys.settings.type = 'HAIR'
    psys.settings.count = 15000
    psys.settings.hair_length = 0.20
    psys.settings.child_type = 'INTERPOLATED'
```

GENERIERUNG VON EBENEN

Vorbereitung Objektplatzierung auf dem Terrain-Mesh

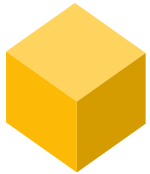


```python
# Count Amount of Vertices
sce = bpy.context.scene
me = obj.to_mesh()
countVerts = len(me.vertices)

# (Create) Object to place on Terrain
bpy.ops.mesh.primitive_cube_add(location=(0.0, 8.0, 0.0), size=0.5)
basic_cube = bpy.data.objects['Cube']

# Select Verts
bpy.ops.object.mode_set(mode = 'EDIT')
bpy.ops.mesh.select_mode(type="VERT")
bpy.ops.mesh.select_all(action = 'DESELECT')
bpy.ops.object.mode_set(mode = 'OBJECT')

ObjectsToSpawn = 5

for i in range(0, ObjectsToSpawn):
    x = randint(0, countVerts-1)
    obj.data.vertices[x].select = True

selected = [(obj.matrix_world @ v.co) for v in obj.data.vertices if v.select]
for vertex in selected:
    name = f'basic_cube {vertex}'
    new_cube = bpy.data.objects.new(name=name, object_data=basic_cube.data)
    new_cube.location = (vertex[0], vertex[1], vertex[2])   #TODO Offset berücksichtigen
    bpy.data.collections["Entities"].objects.link(new_cube)
```

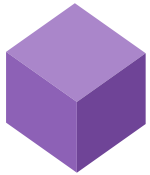
Özkan

GENERIERUNG VON BÄUMEN

Problemstellung



➔ Steuerung der Anzahl der Bäume

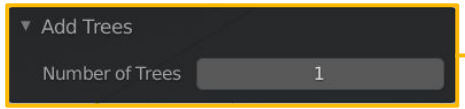


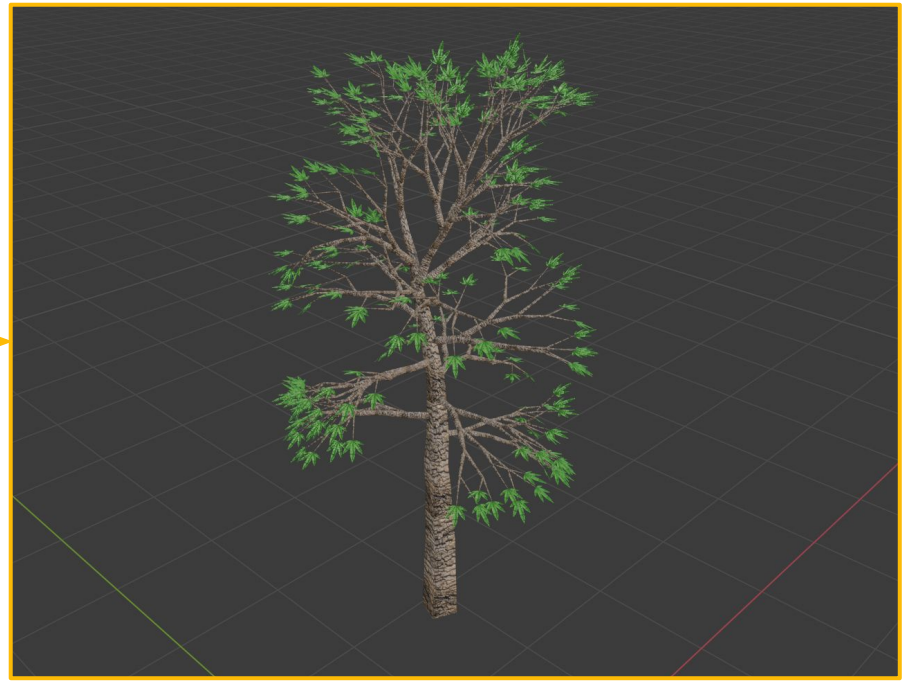
➔ Automatische Anbindung zur Landschaft

➔ Unterschiedliche Formen und Größen

GENERIERUNG VON BÄUMEN

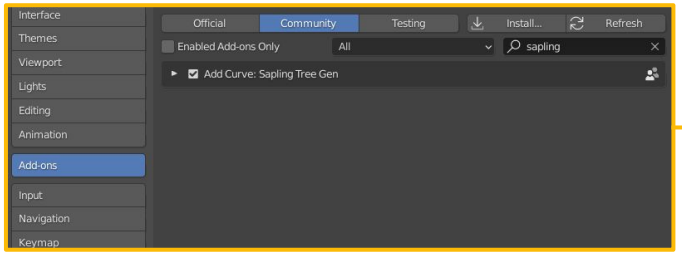
Beispiel aus Ergebnis

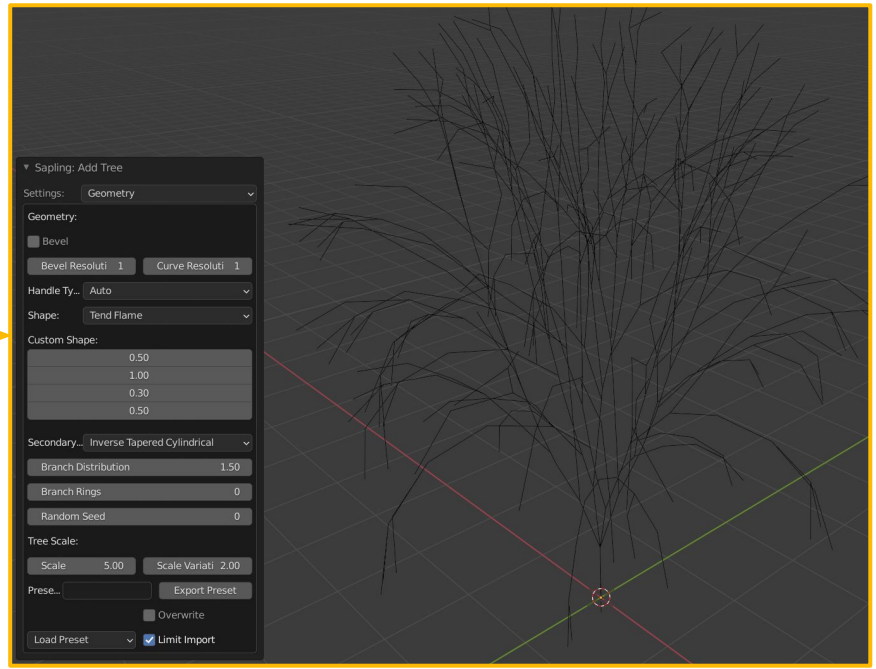




GENERIERUNG VON BÄUMEN

1. Sapling Tree Gen

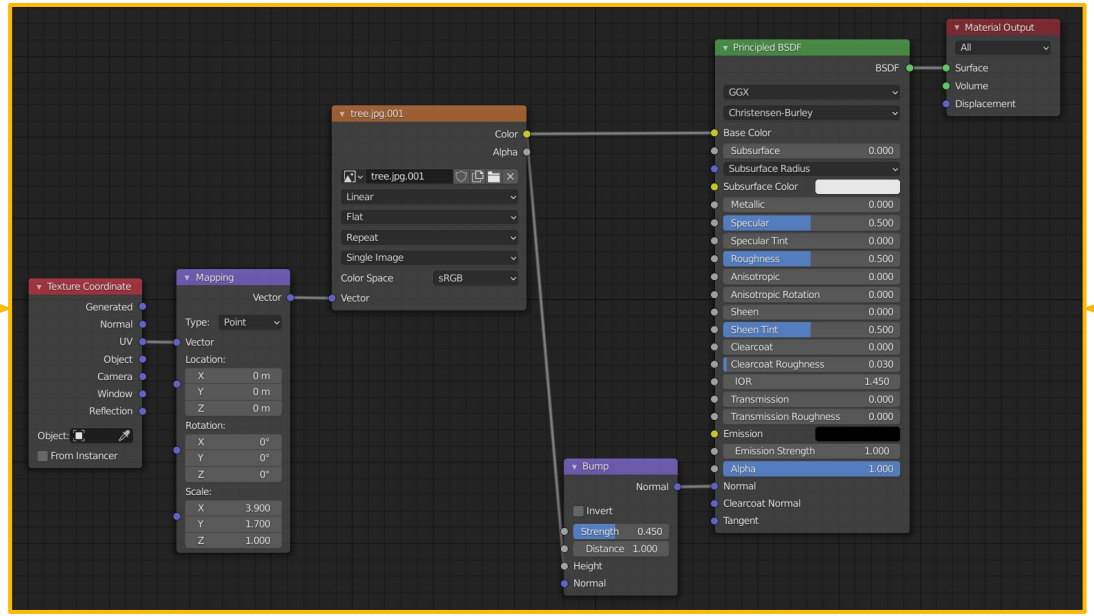




Nur ein Beispiel

Generierung von Bäumen

2. Material - Holz




```python
# Node List
nodes_list: typing.List[bpy.types.Node] = material.node_tree.nodes

# Nodes
node_texCoord: bpy.types.Node = nodes_list.new("ShaderNodeTexCoord")
node_mapping: bpy.types.Node = nodes_list.new("ShaderNodeMapping")
node_texImage: bpy.types.Node = nodes_list.new("ShaderNodeTexImage")
node_bump: bpy.types.Node = nodes_list.new("ShaderNodeBump")
node_bsdf: bpy.types.Node = nodes_list["Principled BSDF"]

# Connect Nodes
material.node_tree.links.new(
    node_texCoord.outputs[2],
    node_mapping.inputs[0]
)
material.node_tree.links.new(
    node_mapping.outputs[0],
    node_texImage.inputs[0]
)
material.node_tree.links.new(
    node_texImage.outputs[0],
    node_bsdf.inputs[0]
)
material.node_tree.links.new(
    node_texImage.outputs[0],
    node_bump.inputs[2]
)
material.node_tree.links.new(
    node_bump.outputs[0],
    node_bsdf.inputs[20]
)

# Manipulate Nodes
print(node_texImage.inputs)
node_mapping.inputs[3].default_value[0] = 3.9
node_mapping.inputs[3].default_value[1] = 1.7
node_bump.inputs[0].default_value = 0.45

# Add Texture
image: bpy.types.Image = bpy.data.images.load(
    os.path.dirname(os.path.realpath(__file__)).replace(
        'main.blend',
        'textures\\tree.jpg'
    )
)
node_texImage.image = image
```

GENERIERUNG VON BÄUMEN

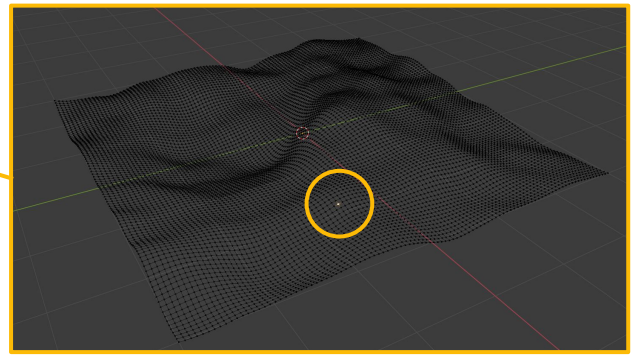
3. Material - Blätter

GENERIERUNG VON BÄUMEN

4. Landschaft

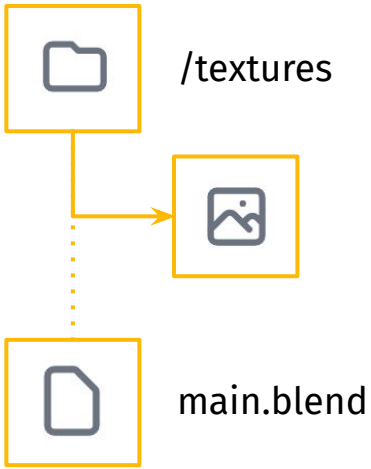
```python
# 1. Get Terrain
terrain: bpy.types.Object = bpy.data.objects.get('Terrain-Plane')

# 2. Extract Vertices
vertices = [(terrain.matrix_world @ v.co) for v in terrain.data.vertices]

# 3. Select random Vertex
rand_vert = vertices[int(self.getRandom(0, len(vertices)))]

# 4. Set Tree Location
tree.location = (rand_vert[0], rand_vert[1], rand_vert[2])
```

Code ist abstrahiert



GENERIERUNG VON BÄUMEN

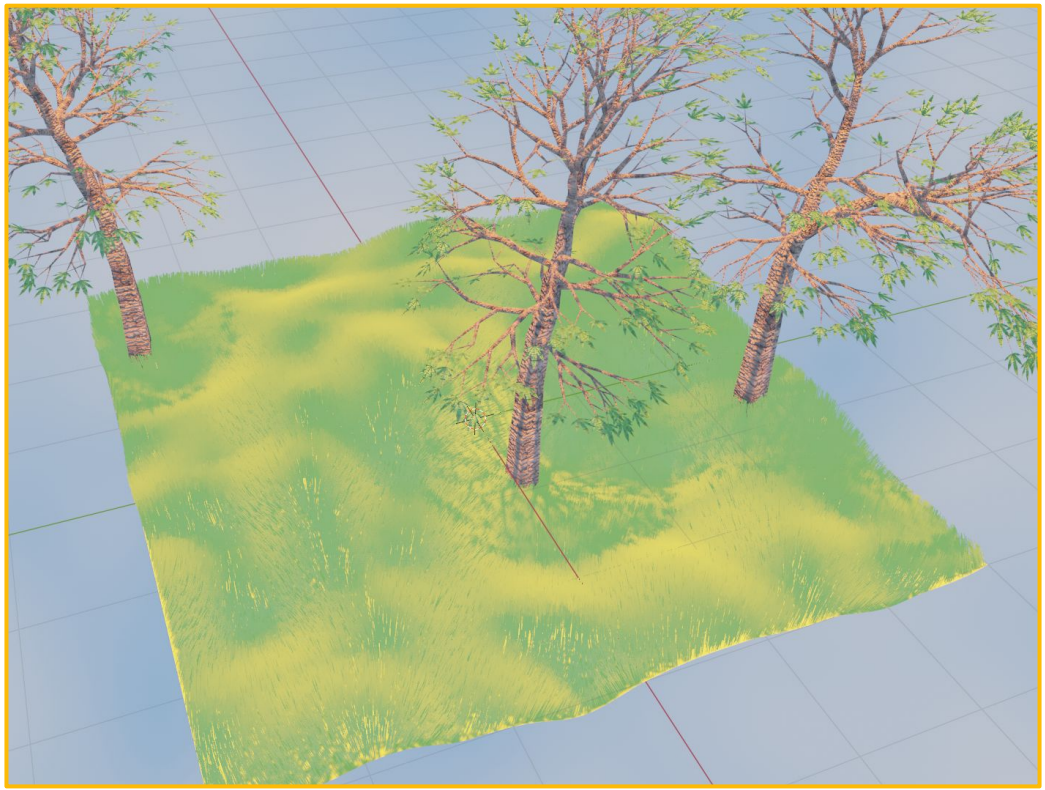
Probleme: Pfad




```python
# 1. Texture Image Node
node_texImage: bpy.types.Node = nodes_list.new("ShaderNodeTexImage")

# 2. Load Texture
image: bpy.types.Image = bpy.data.images.load(
    os.path.dirname(os.path.realpath(__file__)).replace(
        'main.blend',
        'textures\\tree.jpg'
    )
)

# 3. Set Texture to Node
node_texImage.image = image
```

GENERIERUNG VON BÄUMEN

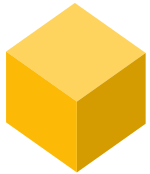
Ergebnis



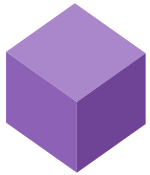
Justin

GENERIERUNG VON STEINEN

Problemstellung



➔ Anzahl der Steine steuern



➔ Anbindung der Steine an die Landschaft

➔ Unterschiedliche Formen und Größen

GENERIERUNG VON STEINEN

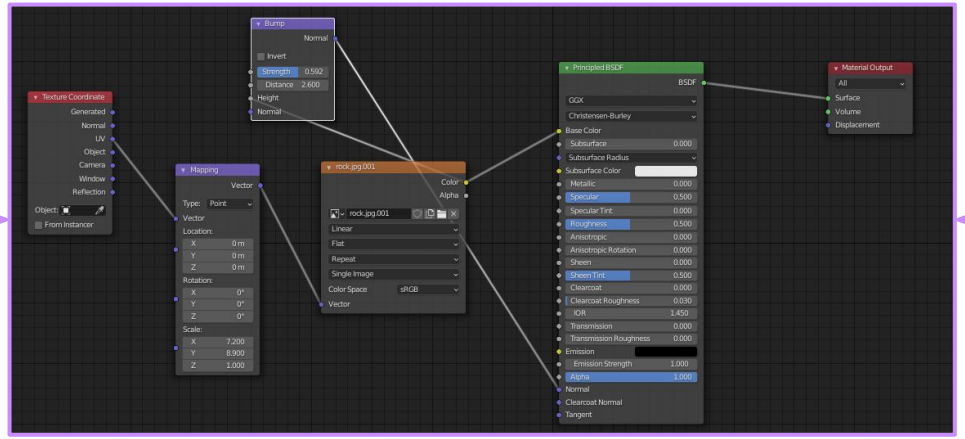
Einstellbare Parameter

- ➔ Anzahl der Steine
- ➔ Minimale Größe eines Steins
- ➔ Maximale Größe eines Steins


```python
# define amount of stones that should be generated
amount_of_stones: bpy.props.IntProperty(
    name="Amount Of Stones",
    description="Changes the amount of stones",
    default=1
)

# define min and max size of stones
size_min: bpy.props.FloatProperty(
    name="Size Min",
    description="Changes the Min Size of a Stone",
    default=0.4
)

size_max: bpy.props.FloatProperty(
    name="Size Max",
    description="Changes the Max Size of a Stone",
    default=2.5
)
```

GENERIERUNG VON STEINEN

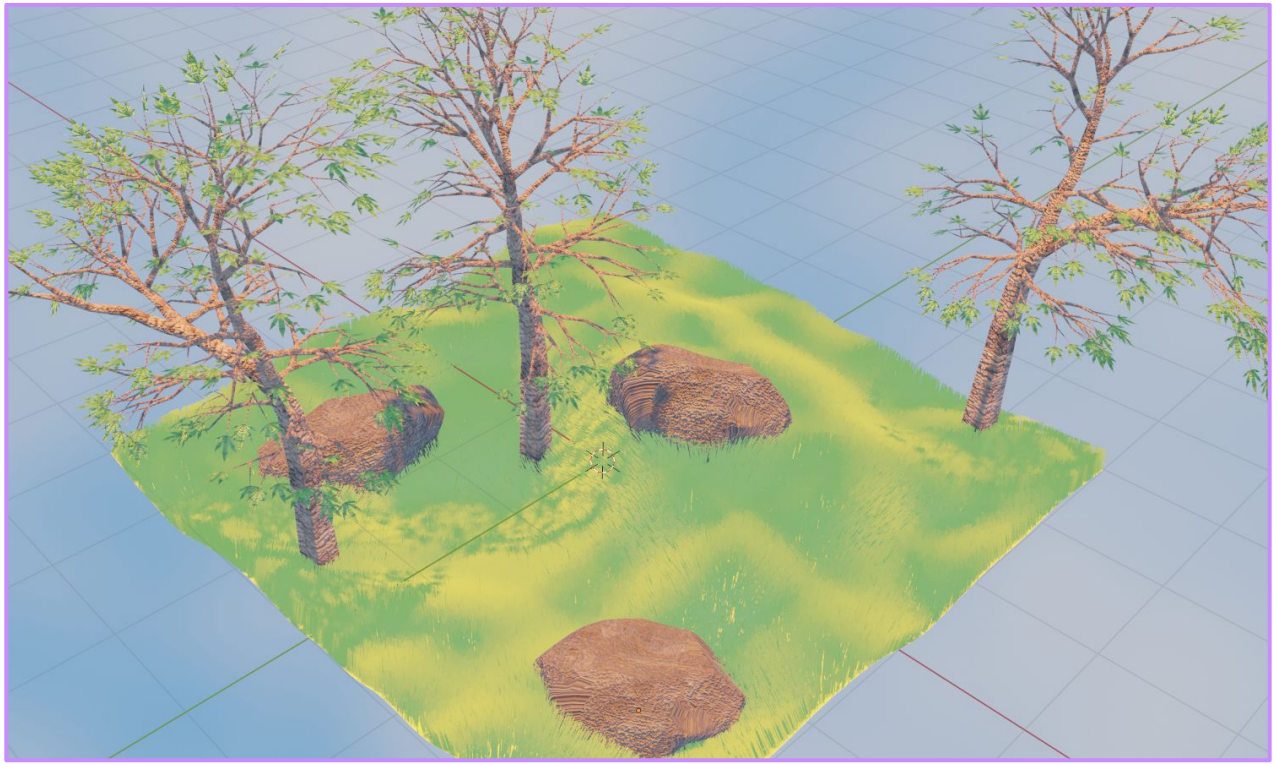
Material - Stein




```python
# Connect Nodes
mat.node_tree.links.new(
    node_texCoord.outputs[2],
    node_mapping.inputs[0]
)
mat.node_tree.links.new(
    node_mapping.outputs[0],
    node_texImage.inputs[0]
)
mat.node_tree.links.new(
    node_texImage.outputs[0],
    node_bsdf.inputs[0]
)
mat.node_tree.links.new(
    node_texImage.outputs[0],
    node_bump.inputs[2]
)
mat.node_tree.links.new(
    node_bump.outputs[0],
    node_bsdf.inputs[20]
)

node_bump.inputs[0].default_value = 0.592
node_bump.inputs[1].default_value = 2.6

node_mapping.inputs[3].default_value[0] = 7.2
node_mapping.inputs[3].default_value[1] = 8.9
```

GENERIERUNG VON STEINEN

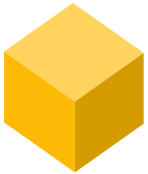
Ergebnis



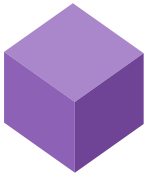
Sebastian

GENERIERUNG VON HINTERGRUND & LICHT

Problemstellung



➔ Platzierung und steuerung des Lichts



➔ Farbtemperatur der Sonne

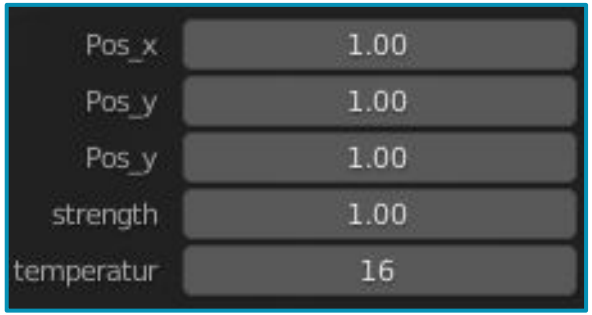
➔ Erstellung der Hintergründe

GENERIERUNG VON HINTERGRUND & LICHT

1. Parameter und Umsetzung

Parameter der Sonne:

- ➔ Position frei wählbar
- ➔ Stärke Frei wählbar
- ➔ Farbtemperatur frei wählbar




```
bpy.ops.object.light_add(type='SUN', align='WORLD', location=(self.pos_x  ,self.pos_y  ,self.pos_z)  ,scale=(1,1,1),)
bpy.context.object.data.energy = self.strength
```

GENERIERUNG VON HINTERGRUND & LICHT

2. Farbtemperatur der Sonne

Parameter für Farbgebung

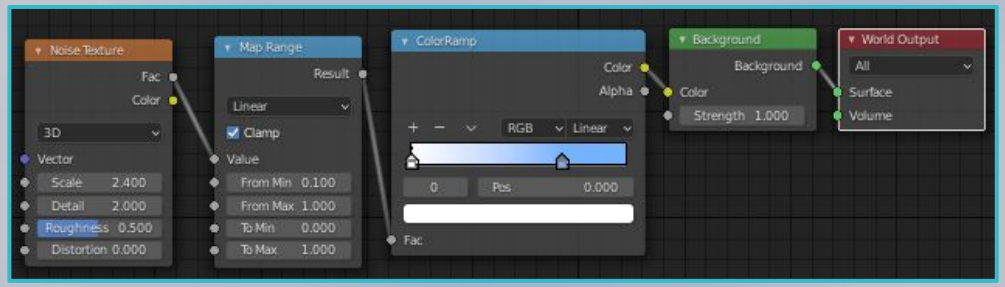
```
temp = self.temperatur *100
redSun: int = 0
greenSun: int = 0
blueSun: int = 0
bpy.context.object.data.color = (redSun,greenSun,blueSun)
```

Umwandlung von Farbtemperatur zu RGB

```
if temp == 1000:
    redSun = 255
    greenSun = 56
    blueSun = 0
elif temp == 1100:
    redSun = 255
    greenSun = 71
    blueSun = 0
```

GENERIERUNG VON HINTERGRUND & LICHT

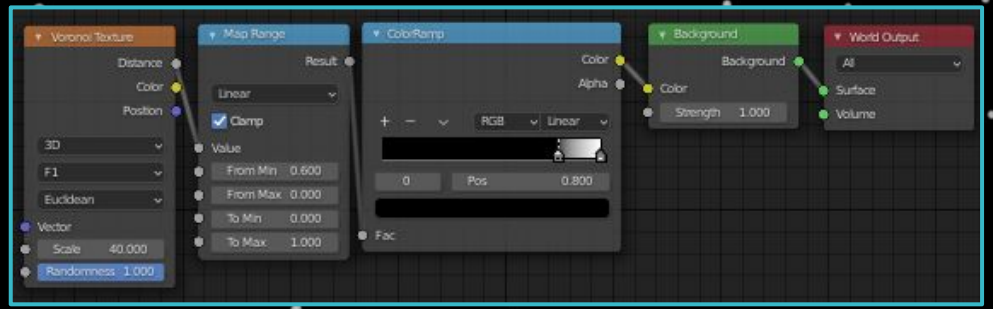
3. Hintergrund Tag




```python
nodes = bpy.data.worlds["World"].node_tree.nodes

my_node: bpy.types.Node = nodes.new("ShaderNodeValToRGB")
bpy.data.worlds["World"].node_tree.nodes["ColorRamp"].color_ramp.elements[0].position = (0.8)
links =  bpy.data.worlds["World"].node_tree.links
links.new(my_node.outputs[0], nodes["Background"].inputs[0])

my_node: bpy.types.Node = nodes.new("ShaderNodeMapRange")
links.new(my_node.outputs[0], nodes["ColorRamp"].inputs[0])
bpy.data.worlds["World"].node_tree.nodes["Map Range"].inputs[2].default_value = 0
bpy.data.worlds["World"].node_tree.nodes["Map Range"].inputs[1].default_value = 0.6

my_node: bpy.types.Node = nodes.new("ShaderNodeTexVoronoi")
links.new(my_node.outputs[0], nodes["Map Range"].inputs[0])
bpy.data.worlds["World"].node_tree.nodes["Voronoi Texture"].inputs[2].default_value = 40
```

GENERIERUNG VON HINTERGRUND & LICHT

4. Nacht Tag




```python
nodes = bpy.data.worlds["World"].node_tree.nodes

my_node: bpy.types.Node = nodes.new("ShaderNodeValToRGB")
bpy.data.worlds["World"].node_tree.nodes["ColorRamp"].color_ramp.elements[1].position = (0.7)
bpy.data.worlds["World"].node_tree.nodes["ColorRamp"].color_ramp.elements[1].color = (0.18,0.45,1,1)
bpy.data.worlds["World"].node_tree.nodes["ColorRamp"].color_ramp.elements[0].color = (1,1,1,1)
links =  bpy.data.worlds["World"].node_tree.links
links.new(my_node.outputs[0], nodes["Background"].inputs[0])

my_node: bpy.types.Node = nodes.new("ShaderNodeMapRange")
links.new(my_node.outputs[0], nodes["ColorRamp"].inputs[0])
bpy.data.worlds["World"].node_tree.nodes["Map Range"].inputs[2].default_value = 1
bpy.data.worlds["World"].node_tree.nodes["Map Range"].inputs[1].default_value = 0.1

my_node: bpy.types.Node = nodes.new("ShaderNodeTexNoise")
links.new(my_node.outputs[0], nodes["Map Range"].inputs[0])
bpy.data.worlds["World"].node_tree.nodes["Noise Texture"].inputs[2].default_value = 2.4
bpy.data.worlds["World"].node_tree.nodes["Noise Texture"].inputs[3].default_value = 2
```

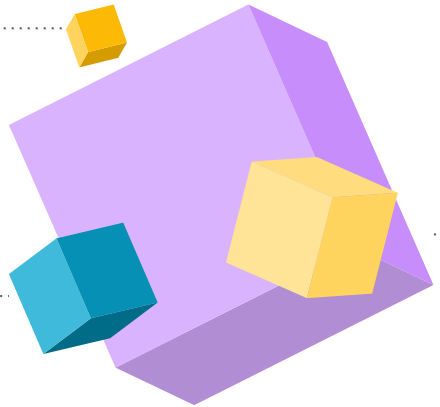
6. FAZIT

Schwierigkeiten & Positives

1.

KOMPL. MESH GENERIERUNG

Generierung von Hügeln, Bergen, Erhebungen und Senkungen (zzgl. Entities etc.)



2.

CODE-COLLABORATION

Struktur und Versionierung von Python / Blender Projekten

3.

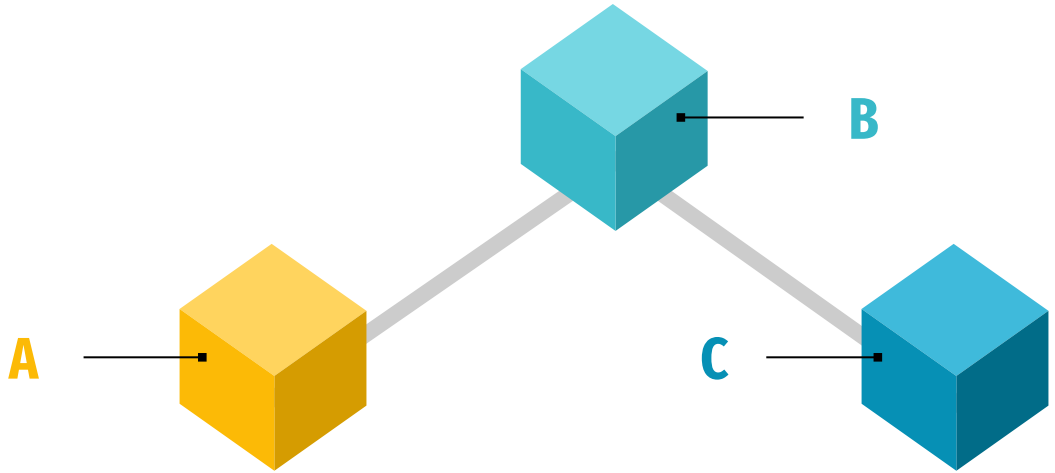
BELEUCHTUNG

Tag/Nacht Implementierung

Raphael

6. FAZIT

Weiteres Vorgehen (bis Abgabe)



CODE-MERGE

Teil-Plugins in ein Haupt-Plugin zusammenführen

UI-DESIGN

UI-Design aus Konzeption implementieren

TESTEN

Plugin auf Funktionalität und Qualität testen.