Program Examples

Voxel Generator for a Torus object

This program generates a volumetric Torus object:

```
import voxelfarm as vf
import math



def length2d(p):
    return math.sqrt(p[0] * p[0] + p[1] * p[1])

def torus(p, r1, r2):
    d = length2d((p[0], p[2]))
    q = (d - r1, p[1])
    return length2d(q) - r2

vf.init() # necessary for Voxel Generator programs

xo = vf.input("x", "Torus Center X")
yo = vf.input("y", "Torus Center Y")
zo = vf.input("z", "Torus Center Z")
r1 = vf.input("r1", "Torus Radius")
r2 = vf.input("r2", "Torus Inner Radius")

vf.set_entity_bounds_x(xo - r1 - r2, xo + r1 + r2)
vf.set_entity_bounds_z(zo - r1 - r2, zo + r1 + r2)
vf.set_entity_bounds_y(yo - r2, yo + r2)

for v in vf.field:
    voxel_p = vf.get_field_origin(v)
    p = (voxel_p[0] - xo, voxel_p[1] - yo, voxel_p[2] - zo)
    f = torus(p, r1, r2)
    vf.set_field(v, f)
```

The program produces the following output:

Depletion Report

The following program computes a depletion report using two date ranges and a block model:

```
import voxelfarm as vf

block_model = vf.input_entity("bm", "Block Model", vf.type.block_model)
attributes = vf.input_attributes("attrs", "Attribute", vf.type.value, block_model)

terrainAD1 = vf.input_date("terrainA_date_start", "Terrain A Start Date")
terrainAD2 = vf.input_date("terrainA_date_end", "Terrain A End Date")
folderA = vf.input_entity("folderA", "Terrain A Parent Folder", vf.type.folder)
terrainBD1 = vf.input_date("terrainB_date_start", "Terrain B Start Date")
terrainBD2 = vf.input_date("terrainB_date_end", "Terrain B End Date")
folderB = vf.input_entity("folderB", "Terrain B Parent Folder", vf.type.folder)

surfaceA = vf.start_composite()
surfaceB = vf.start_composite()

for i in vf.entities:
    if vf.get_entity_type(i) & vf.type.voxel_terrain != 0:
        date = vf.get_entity_date(i)
        if date >= terrainAD1 and date <= terrainAD2 and vf.folder_contains(folderA, i):
            vf.add_layer(surfaceA, i)
        if date >= terrainBD1 and date <= terrainBD2 and vf.folder_contains(folderB, i):
            vf.add_layer(surfaceB, i)

terrain_voxelsA = vf.load_voxels(surfaceA, vf.attribute.volume, None)
terrain_voxelsB = vf.load_voxels(surfaceB, vf.attribute.volume, None)
voxel_data = vf.load_voxels(block_model, vf.attribute.volume, attributes)

attribute_list = vf.get_attributes(attributes)
```

```
for attr in attribute_list:
    vf.start_sum("Depleted", vf.get_attribute_name(attributes, attr))

for v in vf.voxels:
    volume = vf.get_volume(voxel_data, v)
    if volume > 0.0:
        for attr in attribute_list:

            # get block model attribute value
            value = vf.get_attribute_value(voxel_data, attr, v)

            # get terrain volume
            volumeTerrainA = vf.get_volume(terrain_voxelsA, v)
            volumeTerrainB = vf.get_volume(terrain_voxelsB, v)

            # clip affected volume to block model volume
            volumeA = min(volume, volumeTerrainA)
            volumeB = min(volume, volumeTerrainB)

            # sum attribute weight
            if volumeA > volumeB:

                # this was cut
                volume = min(volume, volumeA - volumeB)
                vf.sum(attr, volume * value)
```

The program will compute the tonnage for a user-supplied list of attributes.

The two date ranges are used to compute two surfaces. All terrain datasets within the first date range will be composited together in a single surface. The program does the same with the second date range, obtaining a second surface.

The program will assume the second composite is the most recent one.

Block Model with Terrain View

This program overlays a semi-transparent terrain over a block model:

```
import voxelfarm as vf

terrain = vf.input_entity("terrain", "Terrain", vf.type.voxel_terrain)
```

```
bm = vf.input_entity("bm", "Block Model", vf.type.block_model)

mesh1 = vf.load_mesh(terrain, vf.attribute.none, None)
mesh2 = vf.load_mesh(bm, vf.attribute.none, None)

mat1 = vf.new_material()
mat1.color = [1, 1, 1, 0.9]
mat1.gradient = "dirt"

mat2 = vf.new_material()
mat2.color = [1, 1, 1, 1]
mat2.gradient = "color"

vf.render(mesh1, mat1)
vf.render(mesh2, mat2)
```

Revision #1
Created 17 March 2025 19:13:07 by admin
Updated 17 March 2025 19:19:30 by admin