Report Programs

Reports are special entities that contain tabular results, like a CSV table. These results are obtained by running a custom program over the spatial datasets available in the project. It is up to the report program to select which datasets should be used, and how the data they contained should be mixed and interpreted. Once the program completes execution, its results are stored in the report entity.

A report program will typically go over the following stages:

1. Request user input. Report programs usually ask the user which datasets and attributes should be used and any other information required for the program to run.
2. Load the data. Once input is gathered, the program decides which data to load, and how it should be combined. The data can be a mashup of different datasets.
3. Look at the data. The report program can iterate over the data. If it decides it has found a relevant fact it can emit it as an RDF triplet.

The same report program is executed by the system in many threads at the same time, but over a different region of space in each case. The system will add the results of each individual run, integrating over all emitted RDF triplets. The triplets are then used to compose a table, which is stored in the report entity as a CSV file attachment.

Report Example

The following report program computes the volume of an object:

```
import voxelfarm as vf

entity = vf.input_entity("entity", "Entity", vf.type.voxel_terrain | vf.type.voxel_generator)

voxels = vf.load_voxels(entity, vf.attribute.volume, None)
```

```
volume = 0.0

for v in vf.voxels:

    volume += vf.get_volume(voxels, v)



sum = vf.start_sum("Item", "Object Volume")

vf.sum(sum, volume)
```

The program first asks the user for an entity by calling “input_entity()”. The program will compute the volume of this entity. The program requires this entity to be a Voxel Terrain or a Voxel Generator.

Next, the program calls “load_voxels”. This tells the system the program intends to use the voxel data associated with the input entity. The last two parameters to the function specify which attributes should appear in the voxel data. The first parameter is to select attributes from the default attribute set, like volume in this case. The second parameter is the set of custom attributes that will be loaded.

Then, for each voxel, the program calls the “get_volume()” function. This returns the volume for the voxel, a value between zero and the voxel size at the third power. The program proceeds to add up this value.

At the end, the program declares a new RDF triplet by calling “start_sum()”, the two parameters to this function contain the resource and property used for the triplet. The property value is incremented by the call to the “sum()” function. This is how every voxel’s volume is accounted for.

After running, this program generates a CSV file for the report that looks like this:

Item,Object Volume
Item,523605.137331239

Loading voxel attributes

The “load_voxels()” function instructs the system which voxel data must be loaded for the entity.

voxel_data = voxelfarm.load_voxels(entity, default_attributes, custom_attributes)

The last two parameters to the function specify which attributes we want to load from the voxel data.

The first parameter is to select attribute from the default attribute set, like volume in this case. This parameter is composed by performing a bitwise OR using the constants designated for each default attribute. The following table lists the possible values:

voxelfarm.attribute.none	Constant used for requesting no attributes
voxelfarm.attribute.volume	Requests the volume for each voxel

The second parameter is the set of custom attributes that will be loaded.

Volumetric accuracy

The report system in Voxel Farm is voxel-based. Voxels imply a discretization of space. A voxel may contain one or more samples of some object.

Voxel size must be equal or less than half the smallest distance from any two features in the object in order to be able to fully reconstruct the original object. Any pair of features less than the voxel size apart could erode, introducing an error comparable to the voxel size. This problem is generally referred to as aliasing.

Voxel Farm will use a precise sub-voxel geometric model to compute the volume value for the voxel. For instance, consider a sphere (in blue) that marginally intersects one voxel (in red):

The volume computed for this voxel will account only for the portion of the sphere that is really inside the voxel.

For volume computations, any surface inside the voxel is approximated to four quads. This is a close approximation, but it does introduce some error. This error is negative for convex surfaces and positive for concave ones.

At voxel size 0.5m, the earlier report program computes the volume of a 50m sphere to be: 523,605.137331239 cubic meters. The analytical result for the volume of a 50m sphere is: 523,598.7755983 cubic meters. The error introduced by the discretization of the sphere is 0.0012%

Revision #1
Created 17 March 2025 19:08:29 by admin
Updated 17 March 2025 19:10:48 by admin