AI4media

ARTIFICIAL INTELLIGENCE FOR THE MEDIA AND SOCIETY

VolEvol – D6 –

The VolEvol Software Prototype – Description and Usage

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951911

info@ai4media.eu www.ai4media.eu

Index of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951911

1 Introduction

This document describes the software prototype developed within the VolEvol project. The purpose of the application is to automatically generate sets of representations from volume datasets, that is, images detailing the spatial and geometric characteristics of surfaces and objects found in such data. The application takes as input a regular grid (3D-matrix) of numeric values representing various physical properties of a certain medium and generates images of the data based on exploring and analyzing the values, and assigning visual properties to the underlying objects. Essentially, it generates images from 3D non-visual data originating from technically any source, including various scanning devices, procedural generation, and 3D art. Consequently, the resulting software is intended to fulfill the general objective of VolEvol. Development was carried out in the Python language for portability, practicality and ease-of-use. While the software is completely automated, we provide the means to adjust multiple parameters which affect the outcome, via the command line. This allows a high degree of customization for the components of our application. The application was developed with multiple considerations in mind, which go beyond simply meeting our objectives:

- **quality-diversity output**: the application generates images from volume data given certain objective quality and diversity criteria. While there is a strong subjective component when assessing image quality, we believe to have found objectively-measurable criteria that typically result in relevant representations of the data. Note that the quality and relevance of the images is not strictly the result of these criteria, but the combined outcome of the underlying features, the design of the renderer and the behavior of the evolutionary component.
- **portability**: we provide our software primarily as a set of Python scripts. All inputs/outputs and parameters are completely controllable via the command line. This makes it trivial to run the application and generate standalone binaries in a platform-independent manner.
- **speed:** we designed the components of our application to efficiently use hardware resources. Furthermore, the application can provide a meaningful output in a reasonable amount of time, generating sets of quality images in several minutes at most. This is important for usability and practicality, since having to wait for extended periods of time for the results makes it tedious to exploit the software.
- **minimal footprint:** we developed our application so as to minimize external dependencies, in terms of the required Python modules. We implemented much of the functionality explicitly so as to not rely on many external libraries. Currently, the software can be used with the minimal number of imports and any standalone version generated by freezing the application would require the minimal amount of binaries.
- **highly-customizable:** we expose a multitude of parameters that allow the user to customize much the functionality of the application. This includes parameters which control the graphics effects and aspect of the output images, the sampling and processing used throughout the rendering pipeline, the behavior of the evolutionary optimizer etc.
- **data-agnostic approach:** out implementation is not domain-specific, i.e. it is designed to handle volume data of technically any type. Throughout development, we tested our software on data from multiple fields, such as medical imaging, climatology, industrial

branches, and art. This means that the application can handle technically any volume dataset, with the disadvantage that it is not specifically customized for any particular type of data. However, this generality of use is what we originally intended.

2 Architecture and Components

The design of the VolEvol software largely follows the schematic represnetation from Figure 1. The actual implementation closely matches the flow illustrated in the diagram.

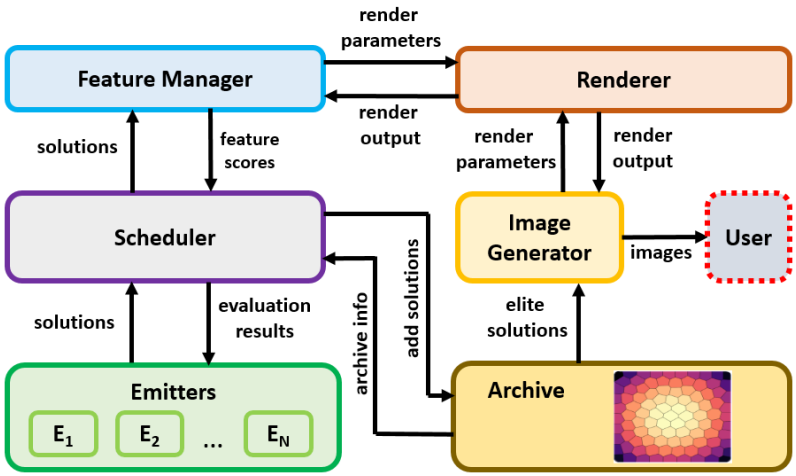

Figure 1. Diagram of the VolEvol application

Several integrated components contribute to the functionality of the application, as follows:

- The *renderer* is based on a direct volume ray-caster and is responsible for generating images from a volume data set, using a set of rendering parameters. Examples of such parameters are the number of identifiable isosurfaces, the values of the transfer function which maps non-visual data to colors and opacities, and the directional vectors of the viewer and light-source. The renderer produces a *render output*, which contains the resulting image of the data set, as well as various properties derived during the rendering process, such as isosurface feature maps, colors and densities of individual surfaces, visibility coefficients, stencils of identified structures etc.
- The *feature manager* is responsible for generating and dispatching features to the other components. Features are measurable properties of the data from the *render output* which can be used to express quality and diversity characteristics. Each feature type can be computed from the data generated by the renderer and the result is a *feature score*, i.e. the measurement associated with the feature.
- The evolutionary optimizer is based on a variant of the MAP Elites algorithm. Its role is to search through the space of rendering parameters and to find sets of such parameters that would result in quality and diverse images. A set of such parameters is referred to as a *solution*, in optimization language. We currently use the implementation from the *ribs* library, where the algorithm consists of three components: a set of *emitters*, which initialize and update the solution population using an evolutionary strategy based on the Covariance

Matrix Adaptation method (CMA-ES), an *archive*, which is a data structure where the best solutions are retained and updated, and a *scheduler* which manages the communication and data exchanges among the other components. The resulting evolutionary algorithm explores the solution space and attempts to find solutions which maximize an objective formed from a linear combination of *objective features* within a diversity space whose axes are the domains of one or several *diversity features*. During the development of VolEvol we identified multiple such features, which we can use interchangeably for objective or diversity purposes.

- Once a set of viable solutions is available from the *archive*, an *image generator* uses the corresponding rendering parameters to produce a final set of images. These images may be of a different (typically higher) resolution than those handled within the VolEvol pipeline.

3 Inputs and Parameters

Several important aspects should be addressed beforehand, in order to understand how to run our software. The application uses two categories of inputs: volume data and parameters.

A volume dataset is usually supplied as a binary file containing raw numeric values which typically represent physical characteristics or some form of non-visual characterization of the contents. For example, in a Computed Tomography (CT) medical scan, the values represent the densities of the scanned region of space. In a meteorological radar scan, the values may represent humidity levels. There is no well-established, universally standardized format for such data, and the layout and significance of the bytes from the data file may be different from one data source to another. While the VolEvol application can read and represent volume data regardless of its origin and significance, certain characteristics of the data set have to be explicitly specified. This is done using our proprietary Volume Descriptor Format, which is a manner of defining certain properties of the data set. These characteristics are specified in a file which is supplied alongside the raw data file. The parameters that can be specified in the Volume Descriptor Format are outlined in Table 1.

Table 1. The Volume Descriptor Format

Parameter	Type	Description
`dataFile`	string	name of the raw data file
`size`	int, int, int	the resolution of the dataset (number of voxels) on each axis
`bytesPerVoxel`	int	the number of bytes used to store a data value
`bigEndian`	bool	True if the data is stored in big-endian order, False if in little-endian order
`scaleFactor`	float, float, float	scaling factors that define voxel spacings on each axis
`headerSkip`	int	number of bytes to skip from the beginning of the data file
`enableNormalization`	bool	(optional) True if the data should be normalized
`viewerPosition`	float, float, float	(optional) the default viewing position
`lightPosition`	float, float, float	(optional) the default light source position
`origin`	float, float, float	(optional) the origin of the data set in world coordinates
`mainAxis`	float, float, float	(optional) the vertical axis (“up-vector”) of the data
`name`	string	(optional) a text representing the ID of the data set

As an example of how a volume data set is defined, we consider the *bonsai* data set, a publically-available CT scan of a bonsai tree that we ship with the VolEvol application. The data consists in two files:

- the density values of the data set are stored in the *bonsai.raw* file.
- the properties of the data set are stored in the *bonsai.vdf* file, which lists the relevant parameters according to the Volume Descriptor Format.

The VolEvol application reads the parameters from the .vdf file and uses them to interpret the binary data from the .raw file. To exemplify, we provide the contents of the bonsai.vdf file in the following listing:

```
dataFile: bonsai.raw
size: 256, 256, 256
bytesPerVoxel: 1
bigEndian: True
scaleFactor: 1.0, 1.0, 1.0
headerSkip: 0
enableNormalization: True
viewerPosition: 0.9, 0, -0.7
lightPosition: 0.9, 0, -0.6
origin: 0, 0, 0
mainAxis: 0, 1, 0
name: bonsai
```

The parameters of the application are used to control various functionalities of the software components. Each parameter can be specified through a configuration file and/or using command-line arguments. A full listing of the user-editable parameters is available in Table 2.

Table 2. User-editable parameters of the VolEvol application

Parameter	Type	Domain	Description
`dataset`	string	-	volume dataset provided via a Volume Descriptor Format (.vdf) file
`outputDir`	string	-	output folder to store generated images
`archiveSize`	int	[10, 10000]	maximum number of elite solutions in the archive
`batchSize`	int	[5, 100]	number of solutions managed by each emitter
`emitterMean`	float	[0.0, 1.0]	initial mean value used by emitters
`emitterSigma`	float	[0.01, 1.0]	standard deviation used by emitters
`lightSourceMaxAngle`	float	[1, 180]	maximum angle to rotate light source in a cone around viewer direction (degrees)
`mainAxisMaxAngle`	float	[1, 360]	maximum angle to rotate viewer around main volume axis (degrees)
`maxNoIsosurfaces`	int	[1, 7]	maximum number of isosurfaces to search for
`minNoIsosurfaces`	int	[1, 7]	minimum number of isosurfaces to search for
`noEmitters`	int	[1, 32]	number of emitters used to generate and update solutions
`noIterations`	int	[10, 1000]	number of evolutionary iterations
`noWorkers`	int	[1, 16]	number of workers to use when computing features
`secondaryAxisMaxAngle`	float	[1, 360]	maximum angle to rotate viewer around secondary volume axes

Parameter	Type	Domain	Description
useCVTArchive	bool	{True, False}	use CVT archive instead of grid archive
diversityFeatures	list	-	features that define the diversity space
objectiveFeatures	list	-	features to be used as objectives
computeWidth	int	[32, 1024]	width of images used for feature computation
computeHeight	int	[32, 1024]	height of images used for feature computation
imageWidth	int	[32, 4096]	width of generated images, in pixels
imageHeight	int	[32, 4096]	height of generated images, in pixels
noDesiredImages	int	[1, 256]	number of desired output images (actual number may be lower)
showGeneratedImages	bool	{True, False}	display generated images in Pyplot window
ambientLightIntensity	float	[0.0, 1.0]	intensity of ambient light
diffuseLightIntensity	float	[0.0, 1.0]	intensity of diffuse light
enableLighting	bool	{True, False}	enable illumination when rendering
enableShadows	bool	{True, False}	enable casting of shadows by objects
enableSurfaceHighlights	bool	{True, False}	enable the highlighting of variations on object surfaces
lightAmplification	float	[0.0, 1.0]	amplification of overall lightsource brightness
rayJitterStrength	float	[0.0, 1.0]	amount of randomization of sampling ray starting positions
rayStepSize	float	[0.001, 0.1]	distance between consecutive samples along sampling ray
shadowOpacity	float	[0.0, 1.0]	opacity of casted shadows
shadowRayStepSize	float	[0.001, 0.1]	distance between consecutive samples along shadow ray
specularLightIntensity	float	[0.0, 1.0]	intensity of specular light
specularPower	float	[0.0, 1.0]	glossiness of surfaces with specular highlights
surfaceHighlightStrength	float	[0.0, 1.0]	visibility of surface highlights

Any/all parameters can be defined in a configuration file. In such files, parameter definitions are provided according to the following syntax:

- parameter assignment: `parameter_name: value`
- the parameters may be divided into categories. These are generally ignored by the parser, but they may help better organize the configuration file. Categories are specified on a separate line as follows: `[category_name]`
- in any particular line, any text after # is ignored (in other words, # may be used to add comments)
- lines consisting only of whitespaces are ignored

Multiple such files may be created to store different parameter configurations. We supply a default configuration file named *config.cfg* with the VolEvol software, which we show in the listing below:

```
[data]
dataset: datasets/bonsai.vdf   #volume dataset provided via a Volume Descriptor Format (.vdf)
file
outputDir:  #output folder to store generated images

[evolutionary]
archiveSize: 200   #max number of elite solutions in archive
batchSize: 10   #number of solutions managed by each emitter
emitterMean: 0.2   #initial mean value used by emitters
emitterSigma: 0.3   #standard deviation used by emitters
lightSourceMaxAngle: 60   #maximum angle to rotate light source in a cone around viewer direction
mainAxisMaxAngle: 180   #maximum angle to rotate viewer around main volume axis
maxNoIsosurfaces: 5   #maximum number of isosurfaces to search for
minNoIsosurfaces: 1   #minimum number of isosurfaces to search for
noEmitters: 3   #number of emitters used to generate and update solutions
noIterations: 10   #number of evolutionary iterations
noWorkers: 1   #number of workers to use when computing features
secondaryAxisMaxAngle: 60   #maximum angle to rotate viewer around secondary volume axes
useCVTArchive: False   #use CVT archive instead of grid archive

[feature]
diversityFeatures: ['VisibilityBalance', 'DensitySpread', 'CurvatureEntropy']   #features that
define the diversity space
objectiveFeatures: ['Pique', 'ColorSpread', 'GradientShift']   #features to be used as objectives

[image]
computeHeight: 256   #height of images used for feature computation
computeWidth: 256   #width of images used for feature computation
imageHeight: 512   #height of generated images, in pixels
imageWidth: 512   #width of generated images, in pixels
noDesiredImages: 25   #number of desired output images (actual number may be lower)
showGeneratedImages: True   #display generated images in Pyplot window

[rendering]
ambientLightIntensity: 0.3   #intensity of ambient light
diffuseLightIntensity: 0.6   #intensity of diffuse light
enableLighting: True   #enable illumination when rendering
enableShadows: False   #enable casting of shadows by objects
enableSurfaceHighlights: False   #enable the highlighting of variations on object surfaces
lightAmplification: 0.2   #amplification of overall lightsource brightness
rayJitterStrength: 0.0   #amount of randomization of sampling ray starting positions
rayStepSize: 0.001   #distance between consecutive samples along sampling ray
shadowOpacity: 1.0   #opacity of casted shadows
shadowRayStepSize: 0.005   #distance between consecutive samples along shadow ray
specularLightIntensity: 0.8   #intensity of specular light
specularPower: 0.6   #glossiness of surfaces with specular highlights
surfaceHighlightStrength: 0.3   #visibility of surface highlights
```

4 Running the application

The VolEvol application is designed to run via the command line. This decision is mostly for portability, compatibility and for easier integration into other, larger frameworks.

The easiest way to run the software is by executing the *volevol.py* script. The application was tested using Python 3.8. It should work with any later version, and possibly with "slightly" earlier versions (Python 3.5+). We went to significant lengths to minimize the number of dependencies and imports, so as to minimize the footprint of the software and make is as convenient as possible to exploit and integrate with other projects. The current dependencies are listed in Table 3.

Table 3. Dependencies of the VolEvol software

Library	Installation using pip	Important notes
numpy	pip install numpy	Numpy arrays are used extensively. Many array-oriented operations are much faster in numpy than equivalent explicit implementations.
glm	pip install pyglm	Allows working with vectors and matrices in a manner similar to GLSL (the language used for writing shaders).
OpenGL	pip install pyopengl	The graphics API used for the volume renderer.
GLFW	pip install glfw	Only used for creating an OpenGL context and an off-screen rendering surface in a platform-agnostic manner.
ribs	pip install ribs	The library used for the MAP-Elites implementation. Significantly faster and more configurable that the original "home-brew" version used in early versions of our application.
matplotlib	pip install matplotlib	Only used for displaying image galleries in a pyplot window. Can be skipped if `--showGeneratedImages` is always set to False.

Executing the application is as simple as:

```
python volevol.py
```

This command will run the application with the default parameter values and using the default volume data set *bonsai.vdf*. Expect to see a progress bar that tracks the application's execution:

```
Working:  [|||||||||||||||----------------------------------] 30.0% Complete
```

Once the application finishes processing the data, the resulting images are displayed in a *matplotlib* window by default (Figure. 2).

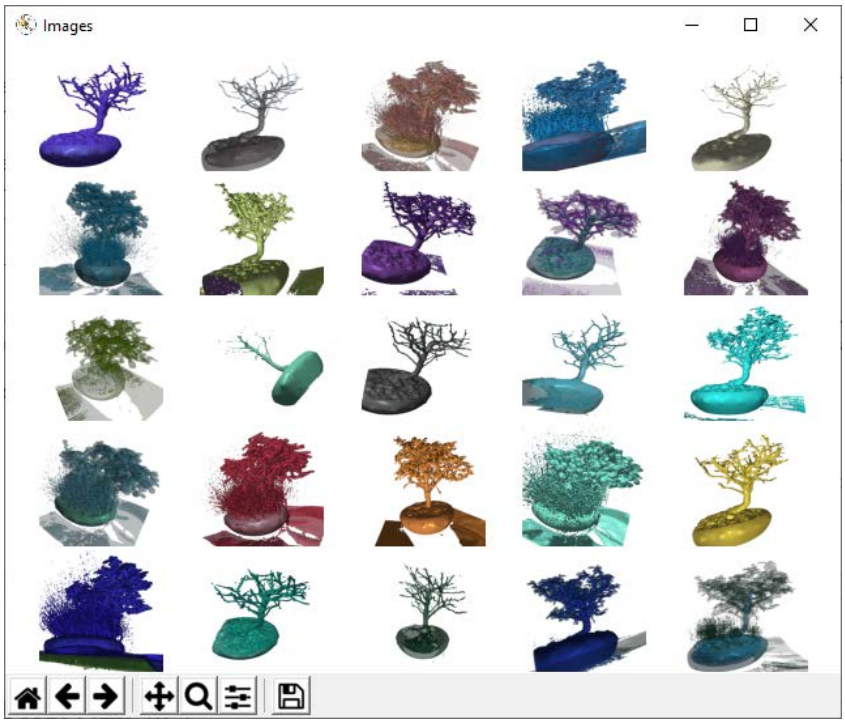

Figure 2. Example of the output of the VolEvol application in a matplotlib window.

Any parameters listed in Table 2 may be modified through command-line arguments, in any combination. Examples:

- use a different dataset:

```
python volevol.py --dataset mydataset.vdf
```

- specify an output folder for the generated images and disable displaying them in a window:

```
python volevol.py --outputDir myresults/myimages --showGeneratedImages False
```

- specify the resolution of the output images:

```
python volevol.py --imageWidth 800 --imageHeight 600
```

- specify the number of iterations of the MAP-Elites-based evolutionary component and use a Centroidal Voronoi Tesselation (CVT) archive instead of the default grid archive:

```
python volevol.py --noIterations 50 --useCVTArchive True
```

- read parameters from a configuration file:

```
python volevol.py --config myconfig.cfg
```

Important: if a configuration file is specified, the values from the file will override the default ones. Any parameter values explicitly specified in the command line will override both the default ones, and any values from configuration files.

5 Creating a Standalone Application

The most direct way to generate a standalone app from our Python implementation is to ”freeze” the script. Freezing is a process where a platform-dependent executable application is generated from a Python project, so that the application can be run without requiring a Python distribution. Freezing works by packaging the python source code, an interpreter and all required modules / binaries in a single folder, or possibly integrating them into a single executable file.

We successfully “froze” our source code using *pyinstaller*. For this purpose, we provide a .spec file that imports the necessary binaries and has the required settings for our application. In order to create a standalone version, simply run:

```
pyinstaller volevol.spec --distpath mydir
```

Note that this does not automatically transfer any input data (such as volume data sets) or shader files (.vert, .frag) to the standalone app.

Important: by default, pyinstaller packages **ALL** available modules with the target script, not just the necessary ones. This can easily result in huge standalone applications where most of the packaged binaries are unused. We recommend running pyinstaller in a virtual environment where only the modules from Table 3 are installed. This should drastically reduce the size of the resulting application.

6 Conclusions

The software prototype presented in this document was developed within VolEvol and its primary purpose is to fulfill our objectives for this project. The app is intended to automatically generate images from volume data sets without explicit user intervention. We do, however, provide multiple parameters that allow users to customize many aspects of our application, from specifying input data and output dimensions, to tweaking the functionality of the renderer and evolutionary components. For every data set tested, the application automatically generates images of the data, the majority of which we consider to be meaningful representations of the underlying objects. Furthermore, we reduced the dependencies and the external modules required by the application to a minimum and we streamlined the code so that there are no redundancies and as few bugs as we could find. Our application does not assume prior knowledge of the contents of the processed volume data set and it is not domain specific. This means it can handle data from any domain, albeit at the expense of being a “jack of all trades, master of none”. In future work, beyond the VolEvol project, we intend to continue improving upon the work carried out so far, possibly by defining a more focused objective and tailoring our optimizer and renderer to be more problem-specific instead of the much more general use-case intended for the purposes of AI4Media.

AI4media

ARTIFICIAL INTELLIGENCE FOR THE MEDIA AND SOCIETY

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951911

info@ai4media.eu www.ai4media.eu