I AM A MAN OF QUOTES.

BOB QUOTEY

MATTHEW J L MILLS

RHORIX USER MANUAL

PUBLISHED ONLINE ONLY

Documentation for RhoRix is created with the Tufte-LaTeXpackage:

TUFTE-LATEX.GITHUB.IO/TUFTE-LATEX/

RhoRix 1.0, October 2017

Contents

RhoRix is dedicated to teachers, mine and yours.

Introduction

It is a fact that human understanding of chemistry can be furthered by the use of visual representations of chemical concepts. Many examples exist, perhaps the most famous of which is the 'ball and stick' model of chemical structures, which depends on an empirical concept of bonding. The theories of Quantum Chemical Topology provide a route to generation of informative 3D pictorial descriptions from quantum mechanics. The use of such images is common in the literature and their improvement is warranted. However, these images have traditionally not been easy to make nor at the forefront of technology, often requiring software in which the QCT and drawing concepts are tangled together. Currently available software is mostly dependent on OpenGL technology, which is designed more for software and game development than drawing. Modern 3D drawing software tends to be expensive, pricing out scientists who may only have occasional need to produce a high-quality render. It is additionally atypical for 3D artists to be involved in production of scientific images.

The program described in this document, RhoRix, is provided to bridge the worlds of QCT calculations and state of the art imaging software, and is built upon the free and open source 3D drawing suite Blender. Blender supports modeling, animation and rendering of 3D scenes, and provides an API for Python scripting allowing the writing of extensions to its functionality (termed Add-Ons). The centerpiece of RhoRix is an Add-On, which allows the user to read a file containing a set of objects constituting the topology of a 3D scalar function of the wavefunction (the electron density $\rho(\vec{r})$ being the canonical example) and creates corresponding 3D objects for manipulation in Blender.

Through this functionality the program allows the full power of Blender to be applied to rendering topological images, allowing creation of state-of-the-art visualizations of the topology of quantum mechanical 3D scalar functions. The QCT and drawing stages are completely uncoupled, and artists are free to work on topologies computed and loaded by theoretical chemists. Simultaneously,

enthusiastic chemists are provided with excellent tools to apply to rendering their figures themselves.

This manual describes the theoretical background, installation and use of the included programs. Users who would prefer to get straight to producing images can consult the provided Quick-Start guide for the absolutely necessary information.

Things RhoRix Does Not Do

RhoRixis not able to perform topological analysis of scalar functions, and adding such functionality is not planned. A number of mature programs already exist and do not currently warrant replacement.

Installation

This section describes the 2 ways of installing RhoRix. The obvious prerequisite is Blender itself. Blender installation is covered in detail elsewhere. In the following text it is assumed you have the appropriate working version of Blender installed.

The more permanent solution is installation, i.e. putting the script into your Blender user preferences directory, which will cause it to appear in the Add-Ons list in the User Preferences window. Doing this directly is OS-dependent. This dependence can be avoided by installing from inside Blender. Navigate to User Preferences and choose the 'Add-Ons' tab. Click "Install Add-On" and navigate to the location of the script on your machine. This will copy the script to you personal Add-Ons directory, and it will now appear in the list of available Add-Ons. To activate it you must tick the checkbox for that entry. You can then save the Blender configuration for all future documents, or alternatively tick the Add-On in each document you use for QCT drawing.

The non-permanent option is to store the script in a text block within your Blender document. This has to be added to each document you make so is less desirable unless you intend to make changes to the script, as it allows for quick editing and reloading of the program. To do this, bring up a Text Editor in a convenient window and click the 'New' button in the window header. Paste the script into the resulting Editor. Press Alt+P to execute the script. After you make changes to the script, pressing Alt+Pagain will re-execute the script and apply your changes. The location of any Python error messages is OS and execution-environment dependent. Check documentation if you cannot find them.

Irrespective of the method, the script will add an operator named 'Import Topology' to the built-in list that can be accessed by pressing the spacebar with the 3D view active. Additionally, the operator is added as a menu item under File -> Import -> Quantum Chemical Topology (.top). No keyboard shortcuts are defined.

Using Blender

It then remains to place appropriate lights and position the camera. The minimal demand on the user is typically to move the camera to get the desired part of the system in the final render. QCT4B positions the camera such that the camera is outside the system and points at the origin.

There are 2 important 3D viewports for the quickstart user. The program will show the 3D view window once the system is read in. The purpose of this view is to allow the manipulation of objects and materials, and provide a pre-rendered image of the scene. The second viewport is the Camera View. The camera view shows you which objects will be rendered to the final image and the orientation they will have. Thus you should check that your system appears correctly drawn in the 3D view first, and then enter the camera view mode to determine the viewpoint of your final 3D render.

To move the camera, it is recommended that you enter the camera view mode (keypad 0), and then use the fly mode (Shift-F) to position the camera using your mouse. The enter key freezes the camera when you find the viewpoint that you want. For larger systems, the z-clipping (which hides objects from the view deemed too far from the camera) will eliminate parts of your system. If this happens, select the camera, and then its object Data panel. Increase the value of 'End' in the 'Clipping' section until the part of your scene you want returns to the camera view. Pressing the F12 key will invoke the Blender Render engine and produce a rendered image of your scene from the chosen viewpoint. The Esc key leaves the render mode if you want to make further changes.

The camera can be moved in other ways once in camera view mode.

Quantum Chemical Topology

A detailed introduction to the theory of QCT (an umbrella term encompassing all methods involving topological analysis of scalar functions in chemistry) is outside the scope of this document and is better introduced elsewhere[1]. However, a description of the relevant concepts is warranted, being a prerequisite for understanding the implementation in RhoRix.

[1] Bader, Popelier

Components of a Topology

A topology consists of various objects, each with different 3D representation. Critical points are points in space described by a position vector $\vec{r}_{cp}$ and a ran and signature (ω,s) which depend on the behaviour of the function around that point). Atomic Interaction Lines (AIL) are paths through the scalar field with particular properties. Interatomic Surfaces (IAS) are boundaries between basins.

Describing Topology: the .top File

Introduction

Given the variety of programs available for topological analysis of scalar functions computed from chemical wavefunctions, it seems apt to provide a generic file definition into which the output files of each program can be converted. This filetype can then be read by QCT4B and no dependence on the underlying analysis programs exists in the code, meaning quirks of the topological analysis algorithms are not dealt with in QCT4B. A further benefit is to avoid tying a user faced with importing a foreign filetype to a single technology. That is, any language can be used to write a converter to the .top format, not just python. The extension '.top' will be used for these files, and they are based on the xml filetype.

Representing Components

Central to producing a 3D image of a topology is a mapping from its constituent abstract objects to geometric objects that can be rendered. A simple concrete example of this is the ball and stick representation of molecules, wherein a nucleus is represented by a sphere, whose radius and color depend on the particular element, and any empirically identified bonds are represented by equal radius cylinders usually all of the same color. There is no correct mapping for a topology, only sensible choices that can be defended by argument. The following section details the mapping implemented in RhoRix.

Critical Points

Critical points are typically represented with a sphere centered on their coordinates. The color and radius of these spheres can be determined by their rank and signature. In particular it is usual to set the non-nuclear CPs to have a small radius, and to use the van der Waals radii to set the relative sizes of nuclear attractor CPs. The set of van der Waals radii used in the Add-On can be found in the ap-

pendix. These are defined in a file and manipulation of defaults can be achieved by editing directly. For CPs corresponding to nuclei, it is typical to color them by element. Non-nuclear CPs are instead coloured by rank and signature. Due to this mismatch between a pair of integers and an element name, a single text label is used in the .top file. Thus a label must be defined for each CP type, and they are listed in the following table.

ω	s	Label
0	0	bcp
0	0	rcp
0	0	ccp

The complete set of element and CP colors is given in the appendix of this document. These are also defined in an external file which can be edited.

Blender allows spheres to be created directly.

Atomic Interaction Lines

Atomic Interaction Lines are rendered as curves. This sits in opposition to the standard method of representing bonds as cylinders connecting nuclei. Given the generally curved nature of AILs between what would normally be termed 'non-bonded' atoms, and the desire to treat all objects of one kind uniformly, all AILs are drawn as curves.

At the lowest level, an AIL can be rendered as a set of disconnected points. The algorithms used to determine AILs are stepwise, so the underlying data is always a vector of discrete values. Whilst this representation is rigorous (all points drawn are directly computed from $\rho(\vec{r})$, so are exact within the tolerance of the integrator) and is typically sufficient for analysis, it is not particularly attractive, and our goal in RhoRix is beautiful images. To improve beyond this, each pair of points can be connected by a straight line, which will appear smooth in the limit that infinite path points are included. Alternatively a smooth curve can be interpolated through the data. As the latter is prettiest, this is the route taken by RhoRix. Blender offers the ability to create an interpolated line from a set of points.

$$V_m = (1 - \lambda_m)V_1 + \lambda_m V_2 \tag{1}$$

The .top File

In abstract terms, a critical point is an object with a position, rank and signature. To represent a critical point then requires a 3-vector of floating point numbers and two signed integers. In order to allow storage of associated CPs with other topological objects, a labelling system is also needed. To achieve this, it is necessary to provide a mapping between the rank and signature of a CP and a String variable. Functions for conversion in both directions then need to be defined.

A gradient vector is a set of 3-vectors of floating point numbers, each of which gives the position of a point on its line. Special classes of gradient vector, such as AILs or those which are members of an IAS, are distinguished by their start and end points. All gradient vectors originate at either infinity, or a critical point, and all terminate at a critical point. In an object-oriented world, the start and end points of a given GP can be saved in a GP object as references. When storing on disk, it is necessary to use a reliable labelling system.

An IAS is rigorously a set of gradient paths originating at a given BCP and extending to infinity. However, when drawing an IAS without gaps, it is necessary to triangulate the surface resulting in a set of points and their connections, i.e. a graph. This graph is the abstract form to be stored. Since the IAS is specifically associated with a BCP, the label of this BCP should also be stored.

The top file therefore has the following structure:

A root tag <topology> and closing tag <

topology>, inside which any number of the following items may be defined in any order:

<CP> <LINE> <SURFACE>

Within these objects, positions are described by <vector> tags with <x>, <y> and <z> components. The <CP> tag includes single <vector>, <rank> and <signature> elements and a <label>. The <LINE> tag includes 2 or more <vector> elements, along with labels for the start and end points. The <SURFACE> tag includes a set of <vector> elements and a set of <edge> elements composed of <A> and <B>, the indices of the points they connect.

Manipulating the Appearance

Whilst the default appearance is fine for rendering images (the MORPHY GUI has been used to render images for a large number of publications), the main point of writing this program is to allow you to go further. Thus, some of the more common appearance changes are described here. The user is urged to think creatively and come up with things not described in this document. A friendly user will let us know what they come up with (and how) so that others can build upon their ideas. A perfect user would write a full description of how they did it for direct inclusion below. Rather than ask permission to reproduce a large number of published images, the program website holds a long list of links to papers that contain QCT images that you can refer to for inspiration, as well as annotations for some papers regarding the graphics.

Program Notes

The QCT4B Add-On can be summarised in the following scheme

1. Read the .top file and convert into python objects.
2. Create the Blender materials required to render the topology.
3. Create the Blender representations (using the materials) of each topological object.
4. Setup the Blender world such that the renderer produces the desired default result.

The first step requires python only. A class is defined for each type of topological object. Due to the use of XML in the .top file, the python XML library can be used to majorly simplify reading. An object (i.e. an instantiation of one of the defined QCT classes) is created for each located topological element. The readTopology function has the required behaviour. It takes a single argument which is the full path to the .top file. An element tree is created directly from the .top file. The root of this tree is the <topology> tag, and the root is scanned branch-by-branch for topological objects.

The second step determines how the rendered scene will look. The default materials are intended to replicate the GUI of the program MORPHY and use the standard render engine. In order to maintain simplicity, a single material is created for each element rather than each atom. This allows the user to change the appearance of all atoms of a particular element at once. Where particular control over a single atom is needed (e.g. for emphasis), the user can create (via Blender's interface) a unique material for that atom. Surfaces are dealt with in the same way, although a separate material for surfaces and nuclei of a given element is defined. Similarly, all AILs share a single material with a default black colour.

Colours for the other materials are discussed above. The default material uses the Lambert diffuse shader (intensity 1), the Cook-Torrance specular shader (color 1,1,1 and intensity 0.5) and has alpha and ambient set to 1. Changing these defaults currently has to be done in code, although they can be edited easily inside Blender.

Creating the blender representations (step 3) simply involves iterating over the QCT objects and calling the appropriate Blender functions.

Finally setting up the world really only requires addition of a light source and camera. It is assumed the user will want to reposition this camera, so its position is essentially irrelevant.

Blender Add-On Code

Certain code is required by Blender to define an Add-On. This section discusses this code in QCT4B. First the program has to import bpy, the blender python API. The Add-On itself is defined as a class that takes an Operator argument. The operator has to be given an ID and a label (Import Topology). The operator class requires definition of the classes 'execute' (called when the user runs the script) and 'invoke'. Invoke opens a file select window with the filter set to only include files with the .top extension. Execute carries out the four steps discussed above. It is also necessary for the Add-On to define register and unregister functions which are used to include the Add-On in Blender itself. As suggested, these classes just register and deregister the operator. The register function is called when the script is added. This is achieved by the only line of executable code in the main program. Finally Blender required the definition of a dictionary. This contains basic information about the Add-On for display inside Blender, including the author, version number etc.

In order to make the Blender extension as accessible as possible, it is necessary to provide means to convert output files from a variety of in-use topological analysis programs to the .top format. As Python is the most prevalent scripting language, it was chosen for this purpose. The conversion is completed via an object-oriented representation of a quantum chemical topology in terms of 3D display. That is, 3 objects must be defined to separately represent critical points, gradient paths and interatomic surfaces.

A critical point object has a position in R^3, a rank, a signature and a label. A gradient path object has a set of points, each of which is a position in R^3 and a pair of labels for its start and end points. A surface object has an associated graph, composed of a set of vertices, each of which is a position in R^3 and a set of edges, which are unordered pairs of vertices. It also has a label for its associated BCP (for an IAS) or nucleus (for an isosurface).

A topology is a set of 3 sets, the sphere, line and surface sets. Note that sets are chosen to preserve the non-ordered nature of these objects. This also decouples the identity of the topological objects from any array indices. Their uniqueness is therefore confined to

the labeling scheme of the topological analysis program. A general labeling scheme for critical points is defined as follows:

$(\omega, \sigma)_E i$

where ω and σ are the rank and signature, E is a String and i is a unique integer within the set of CPs with a given (ω, σ) pair. The presence of E is required due to the connection between NACPs and their corresponding nuclei, for which we need to keep track of the elements. For BCP, RCP, CCP and NNACPs this value is simply set to the appropriate label. The rank and signature are kept explicit in order to most easily accommodate degenerate CPs. A CP therefore must carry a label along with its position. Gradient paths carry two labels, their start and end points, although a special label is required for those lines that originate at infinity. It is simple to have this not conform to the above rules, so the value INF is used. An IAS object carries a single label, its associated CP. In its purest definition, the IAS is a bundle of GPs and so carrying a label is unnecessary since each GP will have its endpoints labeled. However, the 3D representation as a graph with points in R^3 as vertices does not carry this information and so the label is needed.

Functions for input from files and output of top files are required. In order to find wider use, input functions are provided for all QCT analysis programs to which the authors had access. In addition, the output functions are generic such that if the topological analysis program is not supported and cannot be modified to write .top files, only an input function is required. Detailed discussion of such a function is provided below, and the existing input functions in the library can be used for reference.

Writing an Input Function

All input functions share the same form. A file of specified type is opened for reading. It is then parsed line-by-line, searching for topological objects. Python objects are created when such objects are located, and added to an appropriate list. Finally a topology is created from the populated lists and returned ready to be written to a .top file. With the goal of generality, all output files of a given program should be supported, although many do not provide details of the full topology. The result of this is that the library functions must be able to deal with incomplete topologies.

MORPHY

MORPHY is a topological analysis program by Popelier. Specific versions of MORPHY produce different output files. The 3D drawing version produces .mif files, while the standard version produces a .mout log file. The latter of these contains the CP and connectivity information but does not contain coordinates for AIL or IAS objects. The .mif file may contain any topological object.

AIMAll

AIMAll also provides a large number of output file formats, whose contents depend on the calculation settings used. For example, the data written to stdout contains CPs and connectivity, while the .*viz files (e.g. mgpviz, iasviz) files can contain other topological objects.

Functions:

The read functions have the general naming scheme readExtension (e.g. readMif, readMgpviz), and must return a Topology object. Their argument should be the contents of a file of the appropriate type.

The write functions have the general naming scheme writeExtension. The major write function provided is writeTop, which requires a Topology argument and a file to write to. Writing between other formats may be useful for debugging calculations from different sources, and so where possible the relevant write functions are provided. Bear in mind that complete files are not necessarily written (especially in the case of reading from log files); only the topological objects are output in the correct format.

Putting objects in folders.

A typical topology contains a large number of objects. When using Blender for manipulation it can be difficult to locate specific objects, and so a directory system is needed. There are several options:

Separate spheres from lines from surfaces. Spheres can then be grouped by rank and signature, lines by the CPs they connect and surfaces by the CP they belong to.Collecting related objects together. A CP folder contains any associated lines and surfaces.

In order to accommodate procedural programs, the .top file contains the information necessary to allocate CPU memory so that the file may be read sequentially. The required information is:

A fortran module is provided for reading, writing and representation of topologies. Alone this is not very useful, and is intended for incorporation in larger programs that perform further analysis on the topologies themselves.