QCT for Blender

...

A Blender Extension for Importing Quantum Chemical Topologies

Matthew J L Mills

May 8, 2015

Chapter 1

Introduction

Blender is a free and open source 3D animation suite, supporting (amongst other things) modelling, animation and trenderint of 3D scenes. Blender also provides an API for Python scripting, allowing the writing of extensions to the program's functionality. This document describes one such extension (termed an Add-On), which allows the user to read a file containing a set of objects consituting the topology of a scalar function of the chemical wavefunction of a system (the electron density $\rho(r)$ being the canonical example) and creates corresponding 3D objects for manipulation in Blender. This functionality allows the full power of Blender to be applied in creating images and animations of such topologies.

Generation of pictorial representations of topologies is an important and typically overlooked problem. Blender, representing the state of the art for 3D drawing, has the potential to put QCT imaging at the edge of modern graphical capabilities. Past and current QCT drawing programs have at best provided access to the OpenGL libraries. Adding access to Blender allows the full use of the Blender Render and Cycles engines, massively increasing the potential for final images. Drawing on the same 3D canvas as used for the topology is also better facilitated by Blender. Arbitrary shapes are simple to draw/place/manipulate in blender, allowing for easy 3D annotation of rendered topologies.

The basic goal of QCT4B is to allow the user to read a topology into Blender with a default acceptable representation, and then allow the user to apply the full set of Blender tools to making better renders specific to their goals.

1.1 Things QCT for Blender Does Not Do

QCT for Blender is not able to perform topological analysis of scalar functions and never will. Python is not the place for such things, and a number of mature programs already exist. One such external program is required, and a list of existing programs is given below.

Chapter 2

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. However, a description of the relevant concepts is warranted, being a prerequisite for understanding the .top filetype.

2.1 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 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.

Chapter 3

Describing Topology: the .top File

3.1 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 codes 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.

3.2 Representing Components

3.2.1 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 of nuclear CPs. the set of van der Waals radii used can be found in the appendix. However, for CPs corresponding to nuclei, it is more typical to color them by element. 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.

Blender allows spheres to be created directly.

3.2.2 Atomic Interaction Lines

Atomic Interaction Lines are rendered as curves. At the lowest level, an AIL can be rendered as a set of disconnected points. This can be sufficient for analysis, but is not particularly attractive. Beyond this, the point can be connected by straight lines, or a smooth curve can be interpolated between them. As the latter is prettiest, whis is the route taken by QCT4B. Blender offers the ability to create an interpolated line from a set of points.

$$V_m = (1 - \lambda_m)V_1 + \lambda_m V_2 \tag{3.1}$$

Chapter 4

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 must be 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 future, it would be of benefit to be able to define and share 'profiles' that can be selected prior to reading. 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 black colour.

Colors 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.

4.1 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.