<u>**The *CrystalGrower* Visualisation Package**</u>

<u>**Development history:**</u> The *CrystalGrower* visualiser has been adapted over several years by a number of contributors. This section offers a brief summary of the history and rationale behind the development of this software.

To study the crystals grown through the *CrystalGrower* method in detail, it was essential to develop an approach to visualise the results from these simulations. Early versions of the software employed Wolfram Mathematica with hard-wired programs for each structure type being simulated. The results output through this process were limited to the format of simple XY graphs.[1,2] Upon moving to 3D structures, simulation results were processed using Surface Evolver.[3–5] This approach was rather slow and cumbersome, requiring user post-processing on the images produced by this software. The renderer file size also became an issue for more complex zeolite cages, with even a relatively simple β cage increasing the file size dramatically compared to drawing cubes.[5] This large increase in file size prohibited the rendering of large crystals grown through longer simulations.

Due to the limitations imposed by the use of general programs to visualise the results from *CrystalGrower*, the decision was made to construct a companion visualisation program from scratch. This program would be designed from the outset to efficiently visualise results produced by *CrystalGrower*. The first version of this program was produced by Gebbie, and was capable of visualising crystals of a number of zeolite structures including zeolite A, zeolite Y, offretite, sodalite and erionite.[5] The code was written using <u>**Microsoft Visual Studio**</u> in the <u>**C++**</u> programming language, with calls to the <u>**OpenGL and Windows APIs**</u>. A database was present in the program containing the coordinates to build each cage from these select zeolite frameworks. To construct an entire crystal, these cages were stored as a memory object known as a <u>display list</u> then translated and repeated at coordinates listed in the output of the *CrystalGrower* simulation package. Upon manipulating the visualised crystal, these objects would be rotated or translated by an amount input by the user and redrawn. Constructing the program in this manner offered a solution to the issues of file size encountered with Surface Evolver, as the new input for the visualiser consisted only of X Y Z coordinates and a numerical value representing the cage structure to draw at the listed coordinates. Several performance improvements were implemented in the program, including the aforementioned use of display lists. Fully coordinated cages were also eliminated from the input for the visualiser to improve performance, as drawing objects completely obscured by the outer layer of crystal offered no benefit to the user.

Although the standalone visualiser program offered noticeable improvements over the use of third-party software and was perfectly suited to visualising results from the *CrystalGrower* simulation package at the time, issues appeared as *CrystalGrower* was developed further. Due to the use of a hardcoded database, the visualiser was limited to rendering cage structures already stored in the database, meaning it could not be ported to other crystal types without extensive recoding. With the shift of *CrystalGrower* towards a general crystal growth tool, rather than hardcoded for particular zeolite structures, the need arose to adapt the visualisation package to also contain a general algorithm for cage / tile construction.

Extensive recoding was conducted to adapt the *CrystalGrower* visualiser (referred to as simply “the visualiser” throughout this section) to be capable of constructing tiles automatically when required for framework structures (e.g. zeolites). Support was also added to construct spheres at the centres of molecules and ions to visualise ionic and molecular crystals in addition to cage type structures. This feature was then extended to draw entire molecular structures, including atoms and various bond types as spheres and cylinders, respectively.

The major change to the visualiser involved entirely removing the hardcoded database of display lists and creating another input file in addition to the current XYZ coordinate input file. Within this new input file, cage coordinates are listed along with the number of faces present in the cage and which of the cage vertices are a part of each face. For entirely flat faces, these are drawn as simple polygons dependent on the number of vertices present in each face. More complex (non-flat) faces required subdivision into triangles (an object known as a triangle fan in OpenGL), smoothly shaded into each other to give the illusion of a curved face. Combining these faces together yields a full cage which can be saved as a display list and called when required, similar to the original visualiser set up. Once constructed, each cage type is assigned a numerical value corresponding to the same ordering found in the *CrystalGrower* simulation package. The XYZ coordinate data is then fed into the program and the cage display lists are called at the correct coordinates output by *CrystalGrower*. An example of each type of input file used in the visualiser is included in the “Output Files” section of this manual, where all outputs from the *CrystalGrower* simulation package are discussed in detail.