Biophysical Journal, Volume 99

Supporting Material

Exploring the paths of (virus) assembly

Paul Moisant, Henry Neeman, and Adam Zlotnick

ENUMERATOR

Paul Moisant
Oklahoma City, OK
405-549-1286
archer_ghandi@yahoo.com

Enumerator

Enumerator is a suite of programs designed to describe the assembly of a geometric solid from its components. The immediate application is a description of viral assembly. It takes a description of the components and enumerates all the subunits. Then it assembles every possible intermediate from these subunits, assuming each one is added one at a time. These intermediates and the details of assembly are used to create a database. This database can be stored in a number of formats. The database can be used to create a system of differential equations describing assembly in terms of a series of concurrent chemical reactions. The data can be filtered either at, or after, creation time. This allows the creation of a sparse data set, which in turn, allows the generation of a more tractable system of equations.

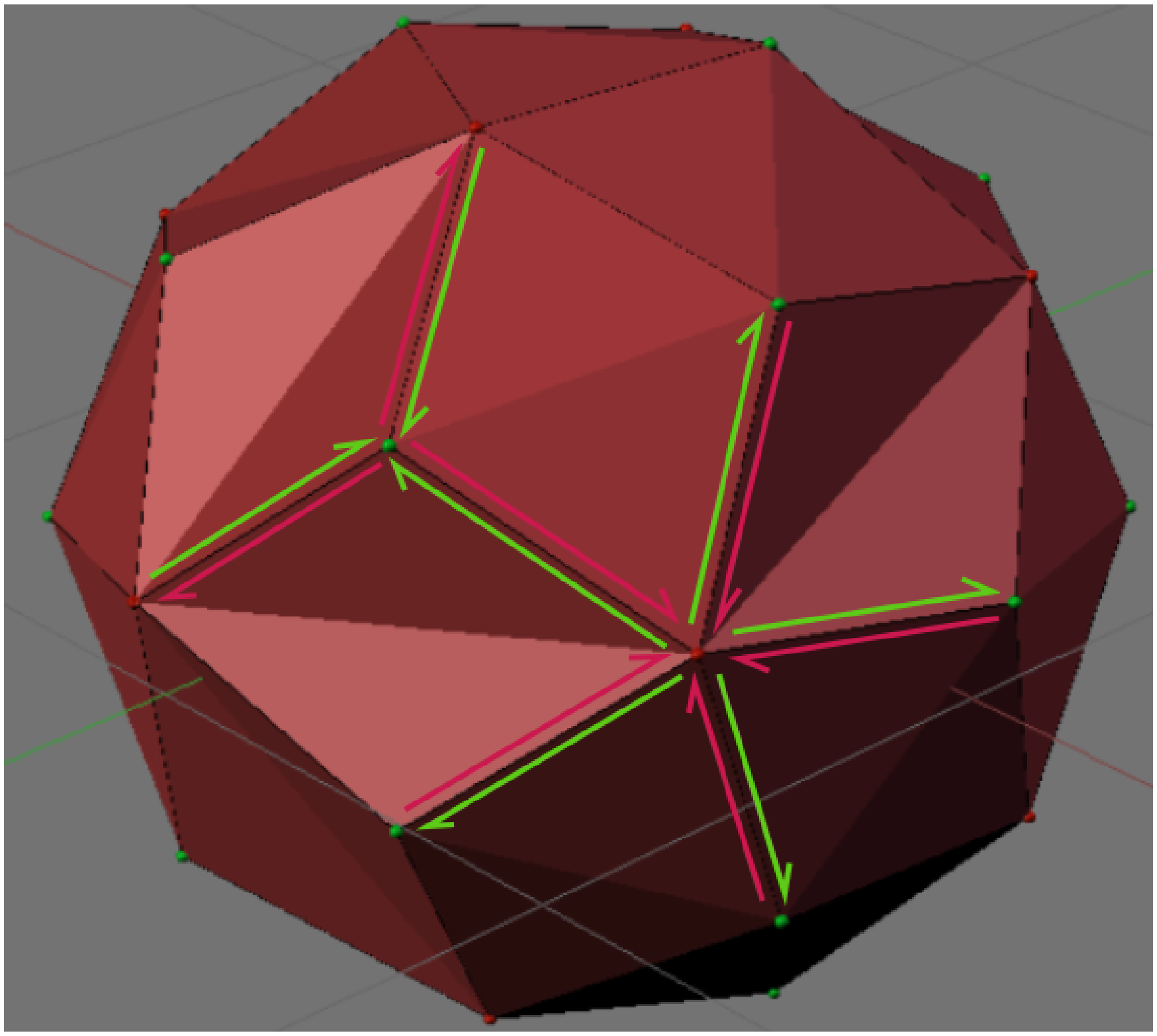
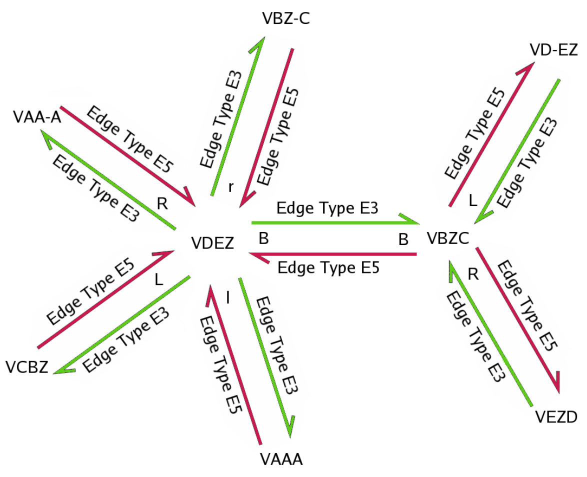


Figure 1 : The 30mer polyhedron with 3-fold vertex in red and 5-fold vertex in green. The directional edges are colored by the destination vertex.

The Components of a Geometric Solid

The geometric solid is described in terms of a number of components that are inscribed in a unit sphere. The most basic of the components, the vertex or a point on the sphere, is used to build all the other components. Based on a pattern that smoothly tiles the sphere , vertex are paired together. A straight line or chord, drawn between the two vertex, form an edge. All edges are directional. They have an inverse edge where the same two vertex have the opposite order. Figure 1 shows the 30 sided polyhedron, hereafter referred to as simply 30mer. Its edges are highlighted. Edges that share the same vertex are combined into turns. They too, have an inverse. The inverse turn is built from the inverse edges and the same common vertex. A set of edges that forms a closed loop, which does not overlap itself and does not enclose any stray vertex, are combined to implicitly define a face. Further more, any set of turns associated with any closed loop constitutes a path.



Left: Visual representation of the prototype edges, turns, and seeds from the 30mer geometry information file '30mer.ini'. 'rhomb' is a seed. E3 and E5 are edge types. 'l', 'L', 'R', 'r' and 'B' are the turn types.

Below: The abstract face.

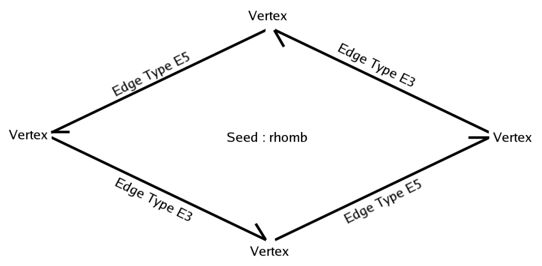


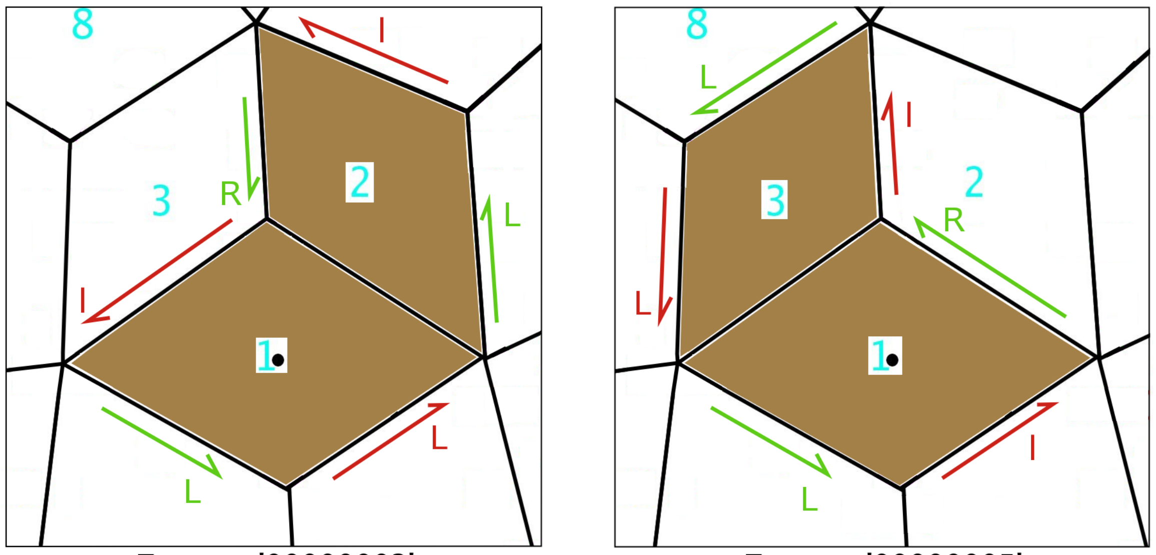
Figure 2

The pattern that controls this building process is described by a set of prototype edges, turns and faces. These are referred to as edge types, turn types, and seeds respectively. This is a minimal description that utilizes only a small subset of the vertex. All the vertex are predefined and help the process to stay on track. The pattern focuses on what types of turns can be made starting from each edge type. Figure 2 shows the turn types (left,) and the abstract face or seed (bottom.)

The building process, starting from a prototype edge, creates edges and turns in a cascading manner. One edge's prototype is used to find the next vertex for all feasible turns. These vertex are used to create the next set of edges. The new edges in conjunction with the starting edge, are used to create the turns. The preferred path of creation navigates around the seeds. When a seed is enclosed, a face is created. After creating a face, the process picks up from an edge that has not been visited. This starts defining the next face. The whole process stops when all edges have been visited. In this manner, all edges, turns, and faces are enumerated.

Assembly

The assembly of the geometric solid is described as beginning with a face. Each face represents a viral subunit. This cascading process starts with a single face, or monomer, and ends with the complete geometric solid, or capsid. Intermediates are simply a set of connected faces that form a thin shell. New intermediates are created by filling in an adjacent open face, thus extending the shell. Starting from a given group of intermediates, each intermediate is processed one by one. The intermediate in consideration is expanded to create all possible new intermediates. These are all collected to make the next group of intermediates. This expansion is repeated, resulting in a cascade of sets. The assembly process stops at the complete geometric solid.



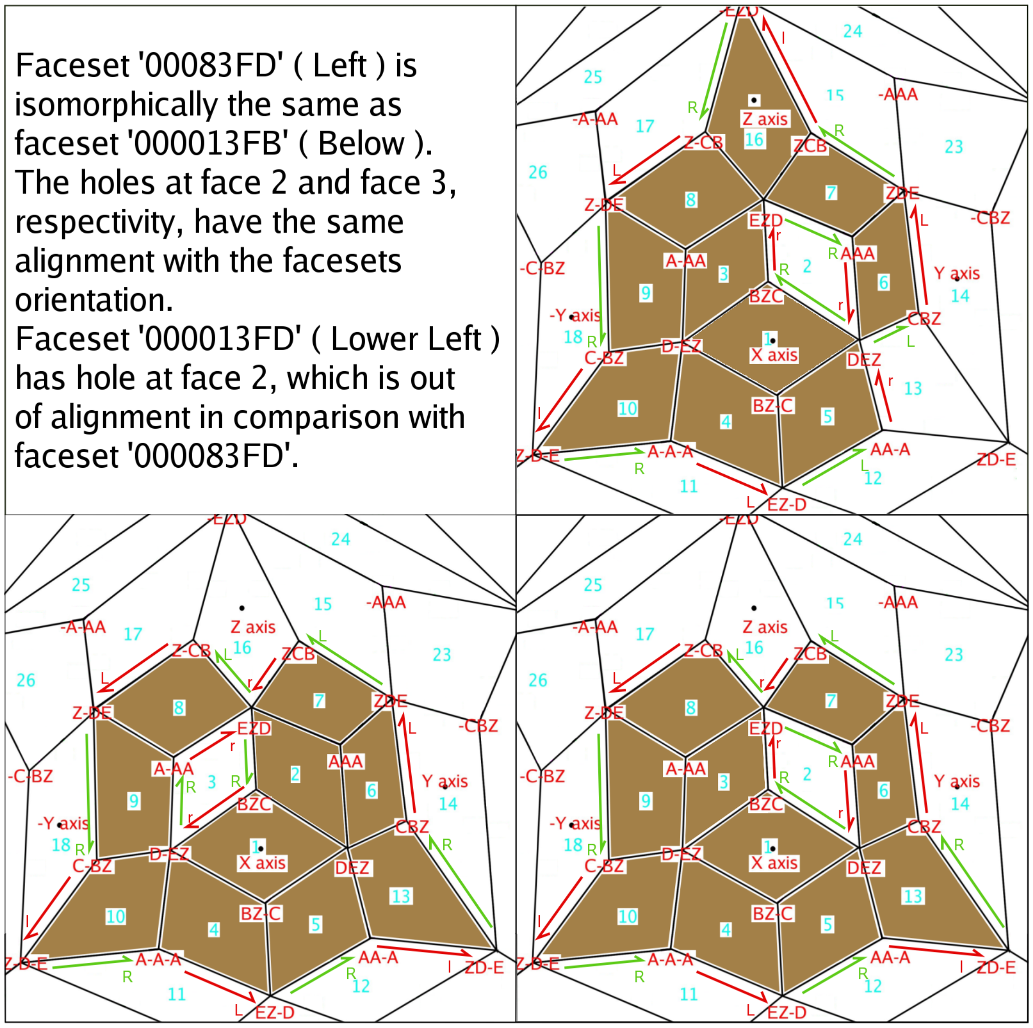
Faceset '00000003' Faceset '00000005'

Both intermediates are dimers. One is a rotation of the other. They both have the same sequence of turns: 'IRILLL'.

Figure 3

Keeping track of intermediates is done by noting which faces are present. This is done by using a face set, denoted by 'faceset'. This is a set of on/off switches, or bits, where each face is represented by a bit. When the bit is on, the associated face is present.

A critical feature of creating new intermediates, is that duplication is removed. Two face sets may have different faces present but are still isometrically the same. This is simply referred to as equal. Figure 3 shows two dimers. One has the faces 1 and 2. The other has faces 1 and 3. These are simply a rotation away from each other.



Faceset '00083FD' (Left) is isomorphically the same as faceset '000013FB' (Below). The holes at face 2 and face 3, respectivity, have the same alignment with the facesets orientation.
Faceset '000013FD' (Lower Left) has hole at face 2, which is out of alignment in comparison with faceset '000083FD'.

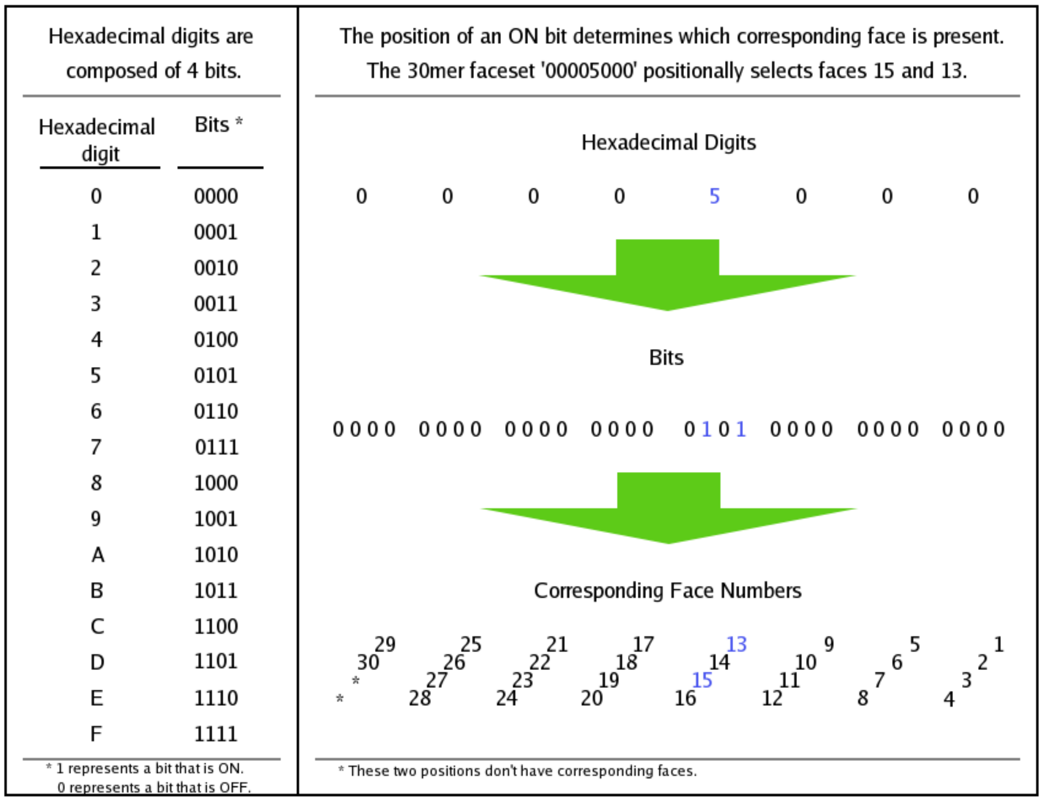
Figure 4 : Comparison of intermediates with a hole.

The equality of the two face sets is determined by comparing the outer boundary path. The outer boundary is the set of all shell edges whose inverse edge belongs to an open face. The outer boundary path is the sequence of turn types that would navigate around this outer boundary. This path is used to name the boundary. Multiple paths are necessary when holes are present in the structure. Equality of two paths requires that the same sequence of turn types be present in both. In the case where multiple paths are needed to describe an intermediate, the relative position and orientation of the holes play a critical part in determining equality. Figure 4 illustrates this.

Definitions

Overall, the geometric solid is defined in terms of the faces present. Each face is implicitly defined by the path that navigates around its edges. The face's starting edge pins the face to a specific place in the geometric solid. The path in conjunction with the starting edge, specifies a specific series of turns. Each of these turns are defined by two connected edges. The edges are defined by two vertex joined by a chord. The vertex are the X,Y, Z coordinates of a point on the unit sphere.

Species, or intermediates, are represented by sets of faces. A face set keeps track of what faces are present. Each face is represented by an index number. This number identifies an entry in an internal table of faces. These index numbers align up with an array of bits starting from the right and going to the left. Figure 5 illustrates the hexadecimal format used by the face set.



Hexadecimal digits are composed of 4 bits.

Hexadecimal digit	Bits *
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

* 1 represents a bit that is ON.
0 represents a bit that is OFF.

The position of an ON bit determines which corresponding face is present. The 30mer faceset '00005000' positionally selects faces 15 and 13.

* These two positions don't have corresponding faces.

Figure 5: The faceset utilizes a hexadecimal coding to store its data.

Generating larger species is accomplished by adding faces. The data associated with this transition, and the two species in question are stored in two internal tables. The table Border, stores the species and assigns an identification number or index number. The table Graph stores the data associated with the transition from one species to the next. Two types of records are present. An up link stores the information associated with the addition a face. A down link stores the data associated with the removal of that face. Each link has its own identification number.

Duplication

As the intermediate building process works, it keeps track of duplication using face sets. Any duplicates are eliminated, while tracking the amount of duplication. This becomes the ways up. It is recorded in the field called 'ways' in the appropriate up link. It is used by the differential equation generator.

Detection of duplication is dependent on the role, and the mapping function. A role is a path that starts from one of its associated edges. However, unlike the path, it is not treated as a closed loop. The mapping function, generated from two edges of equal length, simply rotates a vertex around the sphere. A mapping preserves the relative positioning of the intermediate's paths due to the fact that the position of the hole is determined by any one of its shell edges. The role allows appropriate edges to be selected for use in generating the mapping function. The path and its edges is referred to here as simply a boundary.

In the simplest case where both intermediates have only one boundary, it is sufficient to compare the roles. A table containing all the roles from one of the boundaries is generated. If a role from the second intermediate is found in this table, then the two boundaries are considered equal.

In the more complicated case where two intermediates have more than one boundary each, comparison requires mapping. Given two intermediates with the same number of boundaries, a source and a target, the process is as follows. All possible edge / role pairs from the target intermediate are extracted and stored in a table. For each boundary in the source intermediate, one starting edge and its role are extracted. These pairs are sorted by the number of turn types within their role. One of the largest roles from the source is designated as primary. All the other source pairs are designated as secondary. The primary role is used in an edge/role table lookup. The edge found from this lookup, and the primary edge are used to generate a mapping. Each secondary role is used to look up a target edge. All secondary edges are mapped to the target system. These must equal the looked up target edge, in order for the intermediates to be considered equal. When any two roles in a given intermediate are of equal length, multiple table matches can occur. In which case, all feasible combinations are explored.

The Recorded Data

The data collected during the assembly process is recorded in two tables, the border table and the graph table. The border table stores the data associated with the intermediate. The graph table stores information associated with the transition from one intermediate to another. Both tables have an 'id' field which uniquely identifies a record or entry. This is automatically assigned when an entry is inserted.

Besides the 'id' field, the border table stores a number of fields. The 'nfaces', short for number of faces, is the number of faces an intermediate possesses. The 'size' field is the number of boundaries an intermediate has to describe it. Only the complete capsid has zero boundaries. All others have a minimum value of one. The 'edges' field is the number of open edges in an intermediate. The 'chance' field is the probability that the intermediate will occur given its number of faces. The 'faceset' field is its faceset. The 'complete' field is the number of vertex that is completely encapsulated by its shell. The final field, 'symetry', contains the symmetry of the whole intermediate.

Similarly, the graph table has additional fields beyond its 'id' field. Since a graph is a transition from one intermediate to another, its 'start' and 'end' fields hold the border ids associated with the start and end of its transition. The 'dir' field, short for direction, holds what kind of transition it is. An upward transition, where one face is added, has an upward direction. This is denoted by a 'U'. A downward transition, where one face is removed, has a downward direction. This is denoted by a 'D'. The associated ways up or ways down value is stored in the field 'ways'. The opposing downward transition or upward transition, respectively, is stored in the 'inverse' field. The change in the number of closed contacts is held in the 'new_closed_contacts' field.

The Program Structure

Enumerator's suite of programs encapsulate the flow of information. The main input file, 'geometry.ini', contains the prototype definitions and additional control input. The core programs, load and preload, takes this file as input. The program preload creates the prototypes from their definitions. From these prototypes, it creates the geometric solid components. The prototypes and the components are used to create the model data files. The program load takes these model files as an additional input. It enumerates all the intermediates and builds the database. The filter section of 'geometry.ini' can modulate load output.

The output from the core programs is used as input for the remainder of the programs. The program dump, will dump the intermediates face sets into a file. The program verify validates a list of face sets against the database. These two programs can be used together to self validate the content of the database. The program load_postgre allows the database to be converted into a format suitable for the PostGRE database manager. The program select applies a filter to the database and outputs a list of intermediates by their index numbers. The program diffeq is a differential equation generator that can accept the output from the program select. Finally, the program vrml produces a 3d image file.

All the programs in this suite of programs have command line parameters in common. These are the '-h', '-i', and '-d' parameters. The most important command line parameter, '-h', brings up a description of the program and its valid command line parameters. The '-i' parameter specifies the geometry information file to be used. This defaults to 'geometry.ini'. (The geometry information file is covered in more detail below.) The '-d' parameter overrides the directory used for the database.

The Geometry Information File

The geometry information file has the formal definitions of the prototypes and has additional control information. The prototype sections are described in this order: the vertex, the seed, the edge type, and the turn type. Additionally, the control sections are described in the following order: the database section, the filter section, and the reaction section.

The format of each section starts with the section name encapsulated in square brackets '[]'. Each line in the section is either a comment, an attribute, or an attribute/value pair. The comment line starts with a '#'. An attribute line is simply the attribute name followed directly by a semicolon ';'. This will turn on the appropriate attribute. The attribute/value pair takes the format of 'ATTRIBUTE=VALUE;'. This sets the attribute 'ATTRIBUTE' to the value 'VALUE'. For both the attribute and the attribute/value pair, the semicolons are important. They signal the end of the line. Any additional characters after the semicolon on that line are treated as comment.

The [VERTEX] section contains the vertex prototype definitions. The X, Y, and Z attributes specify the x, y, and z Cartesian coordinates of the point on the unit sphere. The NAME attribute specifies the vertex's name for use in the other sections. The 30mer.ini uses a naming system. The name starts with a 'V'. The next three parts are the labels associated with the x, y, and z values. For example, 'VDEZ' correlates with the vertex at < .67597346921 ... , .41777457946 ... , 0.0 >. A negative sign may by inserted. 'V-DEZ' corresponds to the position of < -.67597346921 ... , .41777457946 ... , 0.0 >. See the file '30mer.ini' for the full list of value substitutions and the formulas used to derive their values.

The [SEED] section contains a seed prototype definition. The attribute 'SYM', short for symmetry, contains the intended rotational symmetry of the corresponding face, (how many ways a face can be rotated and align up with itself.) The NAME attribute specifies the name of this seed for use in the other sections. Multiple seed definitions require additional [SEED] sections.

The [EDGE] section contains an edge type prototype definition. It pairs two vertex and two seeds together, and assigns it a name. The NAME attribute specifies the [EDGE]'s name. The vertex pair is named in the START and END attributes. Its inverse edge is named in the INVERSE attribute. The INSIDE and OUTSIDE attributes specify [SEED] names. The edge type is closely associated with the INSIDE seed when defining a face. Likewise, the OUTSIDE seed is closely associated with the INVERSE edge. Other geometries may have a different arrangement as more edge types and seeds are possible. One [EDGE] section is required for each edge type.

The [TURN] section contains a turn type prototype definition. It pairs two adjacent edge types together and labels it with the type of turn. The type of turn is given in the first character of the attribute NAME, which is case sensitive. The pair of edge types is given in the START and END attributes. The sharpness of the turn is given in the SHARP attribute. Those that hug the inside face are sharp (L)eft turns. Those that follow the outside face are sharp (R)ight turns. A turn that directly ends up where it started from, is a (B)ack turn. All other turn types have the designation of (N)one. (The letters in the parenthesis are the characters used in the geometrical information file.) The attribute DESTINATION holds the vertex name that the turn will end up at. One [TURN] section is required for each turn type.

The [DATABASE] section contains the database name. This is the subdirectory that the data files and database is stored in. To change the subdirectory when invoking any of the programs, specify the new subdirectory after the command line argument '-d'.

The [FILTER] section contains all the filtering specifications for the program load. It specifies the kind of filter to place on the intermediate building phase. The attributes that select filters are TOP_SIZE, MC_SIZE, and CUTOFF. One or more of these must be specified. The NAME attribute can be anything, it is currently ignored. The TOP_SIZE attribute, if given, selects a top filter. Its value sets the number of intermediates that are kept per number of faces. It must have a value of 1 or greater. The MC_SIZE attribute selects the (M)ost (C)ontacts filter. This filter categorizes the intermediates by its number of faces and then by its number of closed contacts (sorted in descending order.) The value given to the attribute, determines now many categories to keep. This too, must have a value of 1 or greater. The CUTOFF attribute selects the probability cutoff filter. It specifies the minimum probability that an intermediate must have in order for it to be retained. Any intermediate with a probability lower than this value is cut off. The WEIGHT attribute specifies the weighting applied to the probability, when the a newly added face only has one new closed contact. Figure xxx shows an example section where all three filters are specified.

The [REACTION] section contains a predefined reaction. This is applied during the intermediate building phase. The attribute FACESET is the face set of the starting intermediate. The attribute FACE specifies what face is to be added during the reaction by its identification number. The attribute UP specifies the chance that this reaction will take place. Multiple reactions require multiple [REACTION] sections. All desired reactions, for a given number of faces or a lesser number of faces, must be given in a continuous manner. The face set of a given reaction must either be the result of another reaction or it must be the monomer. The NAME attribute is simply ignored at this time. Use of this section requires knowledge of the positional relationship of the faces and their numbers. The [REACATION] sections support nucleation.

The [SEED], [EDGE], and [TURN] sections specify the pattern used in building the turns, edges, and faces. Figure 1 illustrates the pattern. (Note the color coded arrows and vertex.) The sections are based on the vertex types. A 3-fold vertex has three [TURN] sections associated with it : a (sharp) left turn, a (sharp) right turn, and a back turn. A 5-fold vertex has 5 sections: a (sharp) left turn, a left turn, a right turn, a (sharp) right turn, and a back turn. Figure 2 (Left) illustrates this. The [EDGE] sections supply the edge types to support the turns. In the case of the 30mer, only two edge types are required, E3 and E5. E3 leads to a 3-fold vertex. E5 leads to a 5-fold vertex. The one [SEED] section, 'rhomb' (short for rhomboid), is sufficient for the 30 mer model. Other models may required more.

The [VERTEX] sections are critical for a successful build of the components of the geometric solid. The program 'preload' uses the vertex specified within the [EDGE] sections START and END attributes to generate the first edge. It also utilizes the [TURN] sections DESTINATION attribute to calculate the angle of the prototype turns. The process used by the program requires the vertex to be in the correct places to eliminate drift due to rounding error. The calculated destination of a turn is replaced by the closest predefined vertex. Finding the values for the predefined vertex is a matter of direct calculation. See the 30mer.ini file for the formulas used in finding the 30 mer model's vertex.

The Preload Program

The preload program takes a geometry information file as its input. It uses the geometry primitives specified by the [SEED], [EDGE], [TURN] and [VERTEX] sections to build the model using the method outlined previously. The [DB] section tells it where to place its internal database as files. The seed, edge type, turn type, and vertex databases are respectively stored in the flat text files: 'seed.dat', 'edgetype.dat', 'turntype.dat', and 'vertex.dat'. The edges, faces and turns are stored respectively in 'edge.dat', 'face.dat', and 'turn.dat'. All the '.dat' files concretely define the model. The other programs, except for dump and vrml, require this model to even operate.

The process follows a few rules. The Edges, faces and turns, when newly encountered, are created, given identification numbers, and stored in an internal database. The inside face is created, if necessary, when the edge is created. The outside face is handled when the inverse edge is processed. It creates edges using the appropriate vertex.

The process seeds its processing queue with the first edge whose vertex come directly from the abstract edge type. During the execution of the main loop, new edges get added to the queue. The processing is done when the queue is empty. For each edge in the queue, the process finds all the appropriate vertex for each type of turn that can be applied to it. From these, the turns, their inverses, and end edges are created. The next edge processed is usually the result of making a sharp left turn. This edge is removed from the queue. When the edges around a face are all processed, the next available edge from the queue is used instead. When all the edges are processed, the model databases are written out.

The Load Program

The program load, builds the intermediates and the links between them. The model generated by preload is used as input. It uses the previously explained assembly process to generate the internal databases. These are output to a set of flat text files. It creates two types of files. The file, 'border_xxx.txt', contains the intermediates. The file, 'graph_xxx.txt', contains the links between intermediates. The names of these files, have the number of faces in place of the 'xxx'. In the case of the graph file, where one is traversing from one number of faces to another, the lesser of the two is used to determine which file any given link is located.

In addition to the previously explained assembly process, load accepts an optional set of user defined reactions. These are defined in the geometry information file [REACTION] sections. These must be connected as previously mentioned. Additionally, multiple paths are allowed. However, all these paths must end up with the same number of faces. These reactions can be ignored by specifying the command line parameter '--do-not-apply-reactions'.

The program implements check points; points in the process where the data is saved. The check point parameters are -'r', '-l', and '--one-pass'. All of these are optional. Once all the intermediates with a common number of faces are processed, a check point is created. The '-r' parameter tells the program to resume from the last check point. The parameter '-l' will tell the program to start its process with the intermediates associated with the supplied number of faces. The parameter '--one-pass' tells the program to stop after creating a new check point. If none of these are specified, then the check points are ignored.

The load program can apply filters to the assembly process. These are defined in the [FILTER] section of the geometry information file or can be specified on the command line. The command line uses the same filter types found in the [FILTER] section. The most contacts is specified by the '--mc' parameter. The top filter is specified by the '--top' parameter. The probability cutoff filter is specified by the '--cutoff' parameter. The values associated with these parameters are the same as the values associated with their respective counterparts in the [FILTER] section. Likewise, all three can be specified. The '--weight' command line option accepts a floating point number and allows the weighting to be set.

The Dump and Verify Programs

This pair of programs help self validate the database content. The dump program dumps the face sets of all or a selection of the intermediates. The verify program accepts the generated output file and checks some or all of the face sets in this file against the database. Any missing or duplicated intermediates are reported.

Both programs have the same optional command line parameters. The number of faces to process is set by the '-n' parameter. The number of face sets to process is limited by the '-l' parameter. Without either parameter, all face sets are processed.

The Load_Postgre Program

The load_postgre program converts the collected data (previously described in 'The Recorded Data' section) into a format that the PostGRE database manager can understand. The output is a number of SQL scripts suitable for PostGRE. The first type is an optional database creation script. This is generated when either '-C' (database creation only) or '-c' is specified on the command line. The remaining two types, 'loadpg_bXXX.txt' and 'loadpg_gXXX.txt', respectively hold the intermediate data and the transitional data . These are associated with a given number of faces. The 'XXX' in the file name is its number of faces. These are created if the option '-C' is not given. Internally, these two file types rely on the COPY command to load the respective table. If the program had been interrupted, the work up to the last stop point can be recovered by specifying a '-r' on the command line.

The data loaded into the PostGRE tables is the same as the collected data with a few notable exceptions. The field; edges, is renamed to 'open_edges'. A new field; 'closed_contacts', is added. The field; 'complete', is renamed to 'vertex_complete'. Finally, the field; 'end', is renamed to '_end' to avoid a PostGRE key word.

The Select Program

The program 'select' filters the established intermediates. The output file is feed into the differential equation generator. The output file consists of indexes to intermediates. Each line of this file contains only one index. The index format is a comma separated list that starts with the number of faces followed by the intermediates identification number.

The program 'select' has additional command line parameters. These are the '-t', '-r', '-l', '--one-pass', '--mc', '--top', '--cutoff', '--weight', '--apply-reactions', and the '-o' parameters. The required parameter '-o' specifies the output file. The optional parameter '--apply-reactions' tells the program to use the predefined reactions found within the geometry initialization file. The remainder are grouped into two categories: Those that control the filtering process and those that control the check point activity.

The filter control parameters are '--mc', '--top', '--cutoff', and '--weight'. These are the same as the ones from the load program. However; the program 'select' requires at least one of the '--mc', '--top', and '--cutoff' parameters to be specified.

Two kinds of check points are implemented. The first type is a time delay controlled check point. When the time is up, another check point is created. The second type of check point is based on data sets; intermediates that have the same number of faces.

The check point control parameters are '-t', '-r', '-l', and '--one-pass'. The '-t' parameter controls the delay time of the time delayed check point. Its value is an integer greater than or equal to zero and is measured in minutes. The default value is 60 minutes. Specifying a value of zero will disable this type of check point. The '-r' parameter tells the program to resume from where it was interrupted. It uses, as a starting point, the most recently occurring of either the last time delay check point or the last completed data set. The parameter '-l' will tell the program to start its processes with the data set associated with the supplied number of faces. The parameter '--one-pass' tells the program to stop after completing a data set.

The Differential Equation Generator

The differential equation generator, 'diffeq', takes the intermediates and the transistions as inputs, and produces a text file that can be used with the Berkley Madonna PDE solver. Optionally, the file generated by the program select, may be used to specify a selection of intermediates. This file is passed in via the '-s' parameter.

The program uses the graph data to formulate the equations. The 'ways up' field is used in calculating values associated with the reactions that builds towards the capsid. The 'ways down' field and the 'new_contacts' field are used in calculating values associated with the degeneration of intermediates. The number of faces and the border ids (identification numbers) are used to break the data into individual equations and interweave them together.

The Vrml Program

The vrml program utilizes a 3D graphic file format called VRML, which stands for Virtual Reality Modeling Language. It translates the data in 'vertex.dat', 'edges.dat', and 'faces.dat' into a base image. If given an optional face set, supplied by the '-f' parameter, it will color the corresponding faces in green.

Conclusion

This is a work in progress. Currently, it has a number of limitations. Use of the geometry information file [REACTION] section requires knowledge of the face numbers and how they are positioned relative to one another. The face numbers are generated by running the program preload. This means that the appropriate geometry information file must be edited after the program preload, is ran to provide the missing face numbers. Also the positional map of all the face numbers must be drawn by hand. This is not a trivial task, considering that one must walk through the model files to generate it. However, there are plans to modify the program vrml to display the face numbers, which will aid in generating the map, and selecting faces for the geometry information file [REACTION] sections. If you are using a model that I have provided, then this map (drawn as Schlegel diagrams) is already completed.

Additionally, the program load only supports one type of monomer. Multiple seed prototypes are treated as the same type of monomer. This behavior is applicable for some models, but not all. It will be addressed in a future version.