NGVV Spec

**Startup:**  
 NGVV is associated with the .VolumeXML extension which allows the user to launch the viewer by selecting the file with that extension. Alternatively NGVV will prompt the user for a path to the .VolumeXML at startup. Either a local path or an HTTP address will be acceptable.

At startup NGVV loads all sections and slice-to-slice transforms (stos)in the volume. It then calculates the slice-to-volume and section-to-volume transforms. This requires ten minutes on my dual-core laptop for 90 sections. A progress bar updates as the volume loads.

The .VolumeXML file will specify a connection string to the database. A connection is attempted. This will involve prompting the user for credentials. I’d like to allow the user to browse if database connection fails, but I won’t do that if it takes time or introduces bugs.

**Setup & First time use:**

The current database schema and NGVV both only support viewing a single volume at a time. Setting up a new volume for annotation involves copying the schema into a new database. Once the new database is created the .VolumeXML file has to be generated for the volume using a python script which specifies the address and name of the volume database.

Once this is done NGVV is launched. The first session begins by defining a number of structure classes. This involves specifying what geometric primitives are used to delineate them on the EM data. A set of suggested tags for each structure is also defined at the class level, i.e. a pre-synapse structure class may suggest a “Ribbon” tag. Synapse classes will be displayed in a tree view.

**What do we annotate?**

Annotation is implemented using the following constructs:

**Structures:** A structure is a real biological object that we have identified in the volume. It could be a cell or a synapse or any other identifiable structure. A structure can span multiple sections or exist only on one section. Structures can be linked together to record relationships. Structures which are part of a larger structure may have a parent. For example a Rough ER structure would have a parent cell it was contained in.

**Structure Type:** A structure type is a tree-based data structure that is populated before annotation begins. It should contain all the broad categories of structures the user wants to track. When the user creates a structure it must belong to one of the predefined structure types. Structure type also describes how those structures are annotated. Currently only a single point is used, but a circle with radius, local mask of member pixels, or a polyline are easily implemented with the current database layout.

**Location:** A location entry marks the position of a structure on a specific section. Each structure must have one location marked, but they can have many locations even on a single section. A location may be linked to another location. This can be a location on the same structure, such as tracking a process through sections, or between different structures, where you would link a pre-synapse to a post-synapse.

**Location Link:** A location link is used within a structure to indicate how the structure spans sections. As the user tracks a process across sections the location links record the approximate morphology of the process. Location links are only are used within the same structure.

**Structure Link:** A structure link indicates that there is a relationship between two different structures. A pre-synapse and post-synapse could be linked for instance.

**TODO:**

**Structure Attribute:** A structure attribute is an additional piece of information attached to a structure. It is a key, value, value-type triplet where the key is a string and the value can be any tidbit of information. One attribute that each structure should have is a default color.

**Template:** A template is a structure type and a set of structure attributes which will be added to each structure created with this template.

**User Interface:**

The majority of the display is devoted to showing the EM section. I’ve been thinking about how to enable mapping between sections. My first approach will be to display the sections above and below the current section in an alternate color. I’ll use some exponential alpha function so most of the adjacent sections are transparent except for dark borders. Hotkeys can hide either adjacent section since I think people will only want to see one at a time. The program will know how many nm/pixel and spacing between sections. I’ll adjust the adjacent sections z coordinate to be either closer or further from the camera the appropriate amount. This should create a parallax effect that will help people get a sense of where processes and borders are relative to each other.

*NOTE: This entire overlay two sections approach doesn’t work because of the 15% overlap of tiles in a section breaks the alpha math and creates a grid.*

The only data visible on the EM image are locations and links between locations Locations are displayed with some text showing the name of the structure they mark. By default the name is the structure type code followed by a ID number. “Cell 335”. Locations on the adjacent sections are visible and selectable, but have a symbol indicating if they are above or below the current section.

On the right side I’ll have a docking toolbar that lists all of the commands available to the user. This is the series of commands I envision:

**Default Command:** The default command is fairly straightforward. Left clicking an existing item selects it. Hitting return brings up the property page of the currently selected item. Right clicking an existing structure brings up a context menu which includes an option to view a property page. Hitting the delete key on a selected location deletes it. The scroll wheel changes the zoom level. Dragging an existing structure with the left mouse button changes its location. Dragging with the middle mouse button/scroll wheel rotates the view. Right clicking an empty region provides a context menu which allows the user to create a new structure in that location. Most commands inherit the behavior of the default command, but the default behavior can be overridden.

**Mark New Structure:** Selecting this option from the toolbar expands a menu which requires the user to select any top level structure class. Clicking the EM image creates a new location linked to a new instance of the specified structure. New items are selected so hitting enter after clicking will bring up the property page of the new item. Clicking another location creates another structure of the same type. This lets the user quickly mark all structure of a given type on a section.

**Extend Command:** This command is used to quickly track structures and create new substructures in the volume. The behavior varies depending on the context of the input:

If you left click a location on an adjacent section and drag the mouse it changes the cursor to a copy of the location type you are dragging. If you release the mouse button it creates a new location on the current section and a link to the old location. Pressing escape or right-clicking cancels the command.

If you left click and drag to an unoccupied location on the same section upon releasing the mouse button you are presented with a menu indicating all structure classes which match or are children of the structure location you just dragged. For instance if you dragged a cell location on your section over the nucleus and selected nucleus from the context menu you would then create a nucleus structure whose parent was the structure location you dragged from. This might then involve more steps such as defining a radius. If you have a process that dips out of the section but doesn’t appear in the adjacent section you can drag the cell location marker to the separated process to indicate the second process links to the first. Hotkeys will be supported so you don’t have to select the structure class from a dropdown menu, you just drag to the correct location and click the hotkey.

If you left click a location on the current section and drag it over to an existing location in the same section you create a link between these locations. This would be useful if a cell had a presynapse structure defined but no post synapse.

If you click a hotkey it will create the structure associated with that hotkey.

**Synapse Marking command:** This is to facilitate marking of synapse/gap junctions quickly. In this mode you select a presynaptic cell, click the presynaptic location, click the post-synaptic location, and click the post synaptic cell marker.

**Annotation Storage:**

Annotation is stored on a SQL Database on a remote server. This is done to allow easy support of multiple users and enforce data integrity. Communication with the server works through a middle tier of WCF objects exposed by the IIS web-server. Web methods are invoked on IIS and the results are returned in a text-based format for cross-platform compatibility. I may expose other bindings to provide a more network efficient response format in the future.

**Queries:**

I've debated a lot about how the client should store the information on the server. Caching is a nightmare and should be avoided at all costs, but querying every time the client needs information is not feasible. So the general philosophy is to minimize client side caching while maintaining enough data to ensure performance.

The server stores a 'version' timestamp column on each row. This is standard for this type of setup, but I'll briefly explain. Assume two clients A & B. Client A & B query a row. Client B modifies and submits the change to the database. If the client A submits a change to the row the timestamp will be different and an error should be thrown and client A is responsible for requerying the row and having the user repeat the change. This prevents people from stomping each other's updates.

The greatest volume for querying information is querying locations. Structures change slowly, but dozens of changes can occur to the locations on a section between visits. The original strategy for locations was to query all locations on the section and the reference sections whenever the user switched visible sections. This worked when there were few annotations, but with over 45,000 locations marked this strategy is making switching sections too slow. So I changed the locationstore code to only query all the locations on a section during the first visit. After that I only query the changes made after the datetime of the original query (actually now its the datetime of the last change, but I plan to fix that). Since clients may not be in synch with the server it is important to use server time in UTC. Because deletes would located with this strategy I added a second table containing IDs and datetimes of deleted rows from the location table that is updated via delete triggers on the location table.

**TODO:**

1. Asynchronous requests: The call to the web service is currently on the main thread, which blocks the UI until the call returns. The location store needs to be made thread safe and the call should occur on a separate thread.

**Volume, Sections, and Transforms**

The volume is loaded from the .xml file passed to the program on startup. There is a single volume object referenced by a global static variable. If you want to load a different volume restart the program.

A volume contains a number of sections, which are linked based on transforms available to the software. A volume also contains a Transform collection which holds all of the ways to transform a section.

A section is built from a large number of smaller tiles. These tiles may be transformed in different ways. So the section stores tiles, which are the images and the current set of verticies used to render them. It also stores a TransfomCollection for each set of transforms which are applied to those tiles.

GridTransforms define how to place the tiles into section space, and from section space into volume space.

When the user selects a section he can choose which transform is used to position the tiles. This transform may have been directly read in from a .mosaic file. It may also have been generated at startup by adding a volume transform onto the section transform.

**Volume, Sections, and Transforms Rewrite**

The original system worked acceptably well on a local network. However at full resolution each tile was 16MB in size, which made off-site use impossible. As a result Brad Grimm implemented a tiling feature for ir-assemble that wrote 256x256 tiles that covered each section with no overlap. I then wrote a python script that combined four of these tiles and downsampled by two to create an image pyramid. I call the old system "Tile Stacks" and the new system "Tile Grids" the challenge is that in the new tile grid scheme the number of tiles on a level is no longer equal, and the coverage area varies. I need to implement a way to use these new tile grids while maintaining compatibility with the old system.

Additionally we need to support placing overlays on each section. Robert wants to see the CMP data in Viking, and I eventually want masks to work. I am preparing this document to outline a new system for rendering the volume.

Challenges:

Tiles Stacks have an easy mechanism to improve resolution as we zoom. The same verticies are used and larger textures are loaded into the same tile object. As we zoom the tile grid we have to load more textures which means we are using different tile objects.

A mapping object was originally a collection of transforms for individual tiles. However tiles in the Tile Grids are already transformed and do not require these objects. There is a single transform required to warp from section to volume space.

There is a call on every mapping object called “VisibleTiles” which is provided the coordinates of the corners of the display. The challenge of the new tile grids is determining which tiles are visible without warping all the tiles beforehand, which I tried and was too slow.

The algorithm I’m using is as follows:

1. Look at the volume-to-section transform and determine all of the mapped points (volume space) which fall inside the visible area or are connected to points in the visible area.
2. Using the resulting triangles, determine which grid points fall inside and map those points to volume space.