1. Simodont model builder

The Simodont system (MOOG) is an educational virtual reality dentistry simulator. It has a 3D display and high resolution haptics, allowing the user to use tools to interact with virtual tooth models in the virtual space. The haptic technology of the system allows for models to have varying 'density' – which changes the feeling of the interaction between the tool and the tooth. Parts of a model with low density feel soft and are more easily removed (less force required) using the drill – high density materials are harder and require more force from the user to remove. If one were to try to create custom models for the Simodont system for use in research or education, it involves a monotonous and technical processes.

The models are *.zip* files containing several files which contain the data and meta-data for the model. Fortunately, the files used to define the model are non-proprietary and so can be edited or created with commonly available tools. The Simodonts model builder is a piece of software that aims to streamline the process of creating models for use in the Simodont systems.

Model data structure

Simodont models contain several multidimensional arrays which define the 'voxels' of the model, which is of a cuboid shape. Voxels are cubes arranged in a uniform 3D grid, which are the 'unit' of the model. When using the Simodont to remove areas of a tooth model, the user is removing these voxels. There are four arrays or channels defining the voxels – these are stored in *.nrrd* files, a common format for storage of multidimensional arrays.

Table 1 - Descriptions of the various multidimensional arrays or 'channels' that are contained within the model

.nrrd name	Description
"color"	4D array of 8-bit unsigned integers. Data for the colour of each voxel. First dimension is always of length 3 (R, G, B)
"density"	3D array of 8-bit unsigned integers. Data for the density at each voxel (0 = minimum density, soft; 255 = maximum density, hard)
"iso"	3D array of 8-bit unsigned integers. Data for the 'Isosurface' used in the marching cubes algorithm for construct the polygonal mesh of the 3D model. Currently this data is set to be the same as the density data, as more testing is needed to see its effects.
"segment"	3D array of 32-bit unsigned integers. Data for the segment assigned to that voxel. This allows for naming areas of the model, e.g. 'target' (area to remove). The number represented in binary contains '1' in the column (counting from the right) of an assigned segment. E.g. a value of 00000000000000000000000000000101 means that the voxel is assigned to segment 1 and 3. Segments are assigned names in a separate file within the model.

In addition to the channels of model data, there are also several meta-data files containing things such as the model name, copyright information, segment names, etc. These are contained in *.xml* and *.json* files.

The Simodont Model Builder

The model builder is a piece of software written in Python. It has been designed to be cross platform, fast, and scalable. It uses a graphical user interface (GUI) to allow researchers to create models while viewing the results in real-time. The nature of the models, being 3D, with multiple channels, makes creating an intuitive user-friendly interface for interacting with the data a significant challenge.

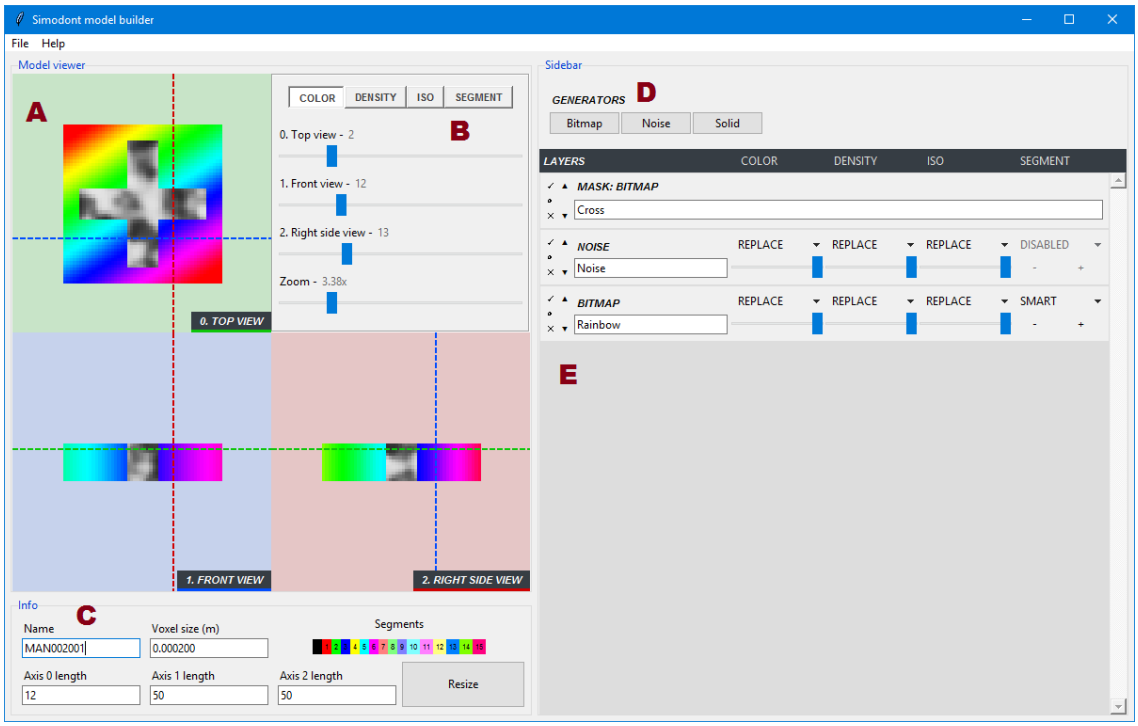

Figure 1 - Screenshot of the Simodont model builder GUI. Here we see a noise layer, masked with a cross (from a bitmap), overlaid on a rainbow bitmap applied to the top view.

Figure 1 is annotated to highlight the main features of the application:

A. *Model viewer* – allows the user to view the models they are creating from the top, front or right side. This is done in such a way to replicate third angle orthographic projection, a commonly used standard for representing 3D objects in 2D.
B. *Model viewer controls* – allows for manipulation of the way the data is displayed in the three views. The channel selector at the top changes which channel of data (Table 1) is currently displayed in the model viewer. The sliders are linked to the colour-coded dashed lines, which indicate the 2D 'slice' of voxels currently displayed. A slider for the zoom level can magnify of the views.
C. *Model settings and information* – Allows the user to change the model name, model size and voxel size. It also displays the segment legend, which shows how the segments are represented using a colour when the model viewer channel (B) is set to segment.

D. *Generators* – Allows the user to generate data for use in the model. When clicked, the generator launches its own mini-application with its own settings and interface. It returns to the main program the data for each channel (color, density, etc). See Generators section.
E. *Layers* – When data is created using a generator, it creates a layer. Layers are composited on top of each other, from bottom to top, a common feature in creative software suites such as photoshop. Here we can delete and reorder layers, and can change the opacity, and the blend mode of the layers – which determines how the layer data interacts with the data in the layers below. See Layering system section.

After the user creates a model, they can output the model into a .zip file containing all the necessary files for the Simodont systems, no files need to be manually edited. The generated zip file can then be uploaded to the Simodonts for use in research or education.

Generators

Generators are the way in which we create models in the model builder. They are mini-applications which create data in whichever way they wish. This data is then converted into a layer.

Currently, three generators are implemented, which allow for building basic models:

1. *Bitmap* – Generates data from a 2D image. Images will be resized and stacked in a selected direction to make them into a cuboid. The ‘color’ channel is taken directly from the image, and the ‘density’ and ‘iso’ channels are created by converting the image to greyscale. The ‘segment’ data is assigned to a specified value where the pixel brightness is greater than 0.5 (0 – 1 scale).
2. *Noise* – Generates 3D fractal noise using a perlin noise algorithm. Many parameters can be changed including the feature size and min/max value. Returned data is all greyscale, segment is a solid colour specified by the user.
3. *Solid* – Generates data based on a user selectable solid colour that is used for all voxels in the ‘color’ channel. Density, iso and segment are assigned manually.

The Generator system is designed to be fundamentally flexible. In the future, if a researcher cannot create a model that suits their needs using the above tools alone, creating new generator is a relatively simple task. Creating and implementing a generator does not require any modification of the model builder source code. A developer must create a python script that generates the data via a custom GUI. The Simodont model builder automatically checks all files in the /generators sub directory in the application. If the newly created generator passes validation checks a new button will appear (Figure 1 – D). The application will launch the generator when the button is pressed and wait for it to return the newly generated data.

An example of where this functionality may be if in the future there is a need to generate a realistic looking 3D tooth object. An algorithm that generates the necessary structure and converts it into voxels could be packaged in a generator, taking advantage of the layering system and exporting functions of the model builder.

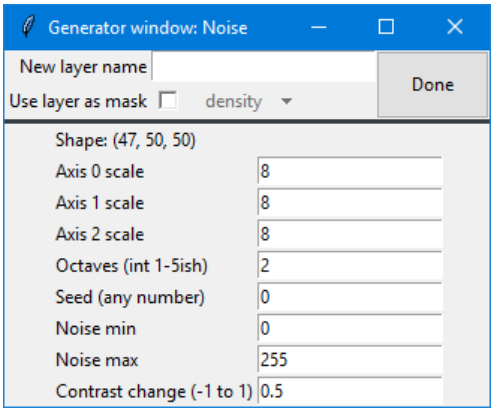

Figure 2 - Example of a generator with a custom GUI and parameters. On pressing “Done” a layer is created using the generated data.

Layering system

The layering system is a feature of the Simodont model builder which makes the application much more powerful. It allows users to combine data created by generators (‘layers’) to create a complex model. This also makes development of models less cumbersome, as individual layers can be deleted or tweaked without the need to start from scratch.

The layering system features support for *mask* layers. Masks are common in other media software suites and control which areas of the layer are used. A mask is greyscale, and where the voxels of the mask are white, data from the layer beneath is passed through. This allows for combining generators, e.g. if a model requires noise generated but only applied to a certain area, we can use a mask above the noise layer to control which areas of the layer are active. The layering system works in parallel for the four channels.

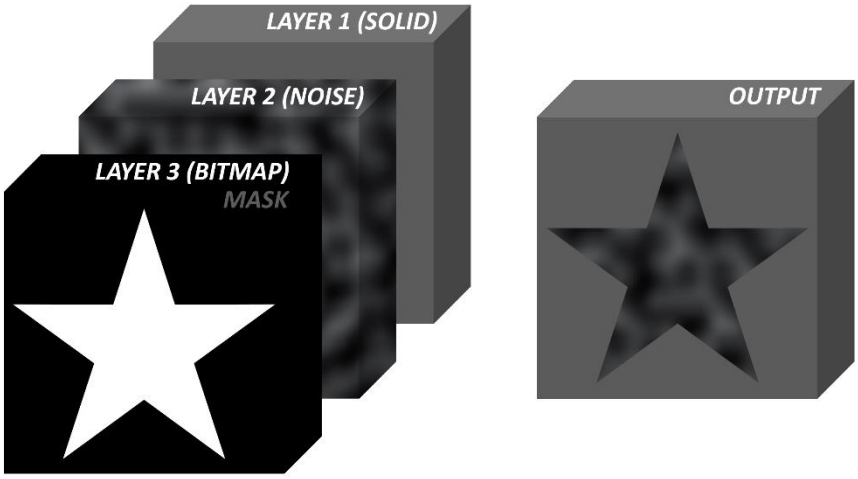

Figure 3 - Diagram showing an example of the layering system, viewing the density channel. Left shows three layers, (1) from the solid generator, (2) from the noise generator, and (3) a mask layer from the bitmap generator. Result is Layer 2, masked by Layer 3, overlaid on Layer 1.

Additionally, the system supports layer 'blend modes'. Blend modes affect the way a layer's data interact with the data underneath.

1. *REPLACE* – Replaces voxels beneath with voxels with layer data. (default)
2. *ADD* – Adds voxel values to voxels beneath. This has the effect of only increasing the values in the channel data (e.g. in the 'color' channel, it will only brighten, and in the 'density' channel it will only increase density).
3. *MULTIPLY* – Scales the value in the layer's data to 0 – 1, and multiplies it by the data beneath. This has the effect of only decreasing the values in the channel data, darkening or decreasing density.
4. *DISABLED* – Does not apply any operation on layers beneath, layer data for this channel is unused.
5. *SMART* (segment channel only) – Replaces the data only where there is a segment assigned (i.e. uses disabled blend mode where the voxel = 0, replace otherwise).

These features of the layering system interact in a rendering chain shown in pseudocode below.

```
# rendering the layers
create empty 'output data'
get layers

for each layer in layers:
  if the layer(s) above this layer are masks:
     get mask and apply it to current layer
  composite layer with 'output data', considering mask, opacity, blend mode

# creating the three 2D images (for display)
'channel data' = get current channel data from 'output data'

for view in views:
  index = get value of the view's associated slider
```

```
15.   take a 2D slice of 'channel data' at index along view axis
16.   convert to RGB image
17.   scale to zoom level
18.   display on screen
```

Output system

The program outputs a .zip folder that is compatible with the Simodont case manager. Upon upload, the “save/load snapshots” setting must be disabled.

Future work

- Ability to render the model in isometric 3D for easier visualisation
- Edit the names and colours of the segments
- Improve clarity and user experience of layer system.
- Save the model as a custom file format, including layer data, that can be saved and edited later.