VascuSynth: Vascular Tree Synthesis Software

Release 1.00

Preet Jassi and Ghassan Hamarneh

April 14, 2011

Medical Image Analysis Lab, Simon Fraser University, Canada
E-mail: {preet | hamarneh}@cs.sfu.ca.

Abstract

In [1], we presented an algorithm to synthesize volumetric images of vascular trees and generate the corresponding ground truth segmentations, bifurcation locations, branch properties, and tree hierarchy. In this work, we provide the software needed to simulate these volumes. Our software expects a number of physical parameters and oxygen demand maps to produce 3D volumetric images of vasculature, as well as information about the bifurcation locations, tree hierarchy and branch radii in a GXL file. We foresee our software useful for large scale evaluation studies of medical image segmentation and analysis software.

Contents

1 Introduction **2**

2 Overview of the Simulation Method **2**

3 Downloading and Building the Software Libraries **3**

4 Specifying the Input Parameters and Running VascuSynth **4**

4.1 Input Parameter File 5
4.2 Output Image Name File 5
4.3 Noise Parameter File 6
4.4 Vascular Parameter File 6
Oxygenation Map 7
Supply Map 7
4.5 Run VascuSynth 9

5 Interpreting the Output of VascuSynth **9**
5.1 The Volumetric Image 10
5.2 GXL File of the Tree Topology and Geometry 10

6 Sample Command and Parameter Files **10**

7 Conclusions and Future Work **11**

8 Acknowledgements **11**

1 Introduction

Automated segmentation and analysis of tree-like structures from 3D medical images are important for many medical applications, such as those dealing with blood vasculature, lung airways, or neuronal structures. However, there is an absence of large databases of expert segmentations and analyses of such 3D medical images, which impedes the validation and training of proposed algorithms. In [1], we presented an algorithm to synthesize volumetric images of vascular trees and generate the corresponding ground truth segmentations, bifurcation locations, branch properties, and tree hierarchy. The tree generation is performed by iteratively growing a vascular structure based on a user-defined, possibly spatially varying, oxygen (or nutrient) demand map. In this work, we provide the software and documentation needed to simulate these volumes. Our software expects a number of physical parameters and oxygen demand maps and produces 3D volumetric images as well as a Graph eXchange Language (GXL) [1] file containing information about the bifurcation locations, branch radii and tree hierarchy. With these volumes and corresponding ground truth data available, large scale validation and evaluation studies can be performed on methods designed to analyze branching tubular structures such as vessel segmentation and bifurcation detection methods.

2 Overview of the Simulation Method

The tree generation is performed by iteratively growing a vascular structure based on a user-defined (possibly spatially-varying) oxygen demand map, while enforcing physical constraints. The flow chart in Figure 1 gives an overview of the algorithm. The algorithm, at a high level, follows the steps below (details are given in subsequent sections as well as in [1]):

1. First the user must **supply the input to the algorithm** (Section 4). In particular, the user supplies the coordinates of a perforation point in 3D, which will act as the root of the tree, and the number of the terminal nodes desired (i.e. the number of leaves in the tree). In addition, the user supplies a number of physical parameters, such as perforation flow and fluid viscosity. Finally, the user must also supply an oxygen demand and supply map, which will guide the iterative growth of the tree.

2. With all input parameters specified, **the algorithm iteratively constructs the tree**. During each iteration, a candidate terminal node is first chosen according to the oxygen demand map. Locations in the volume with high oxygen demand are more likely to be picked as candidate terminal node

[1] http://www.gupro.de/GXL

locations. Then an existing branch is bifurcated to supply this terminal node. The choice of the branch to bifurcate and the location of the bifurcation point along the branch is chosen optimally according to a fitness function associated with the tree (as detailed in [1]). The oxygen supply map is then updated accordingly, and a new terminal node is selected, and so on. After the tree has completed its growth (i.e. the desired number of terminal nodes is reached), the final radii of the tree branches are calculated according to physical flow and radii constraints (described in [1]).

3. Now that the tree is constructed, **output files representing the vascular tree are generated** (Section 5). In particular, a 3D volumetric image of the synthesized vasculature and a GXL file describing the tree topology and geometry.

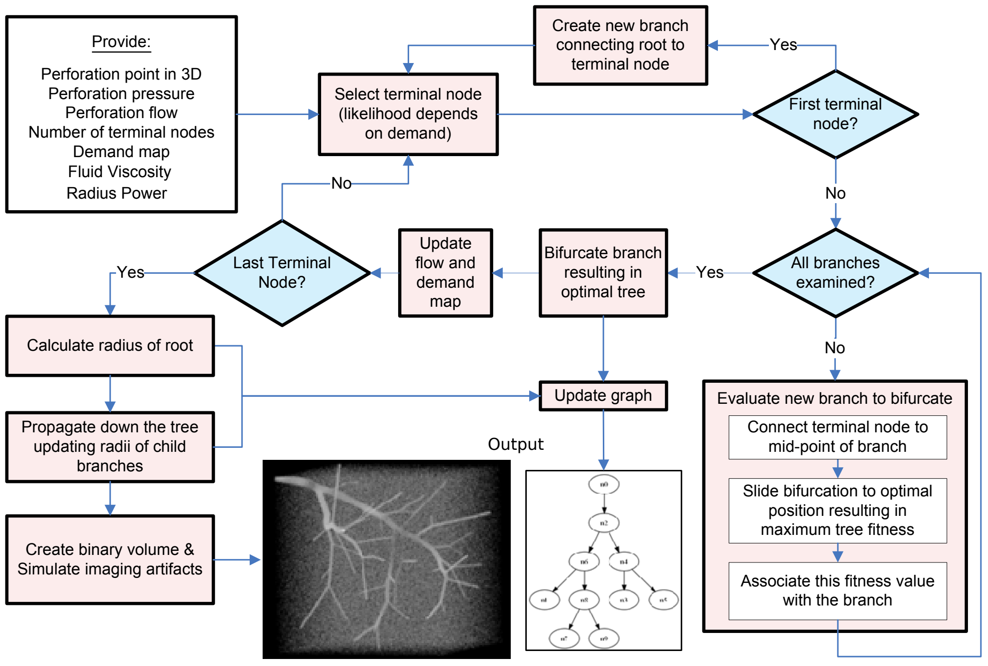
Provide:
Perforation point in 3D
Perforation pressure
Perforation flow
Number of terminal nodes
Demand map
Fluid Viscosity
Radius Power
Select terminal node (likelihood depends on demand)
Create new branch connecting root to terminal node
Yes
First terminal node?
No
All branches examined?
Yes
Bifurcate branch resulting in optimal tree
Update flow and demand map
Last Terminal Node?
No
Yes
Calculate radius of root
Propagate down the tree updating radii of child branches
Create binary volume & Simulate imaging artifacts
Update graph
Output
No
Evaluate new branch to bifurcate
Connect terminal node to mid-point of branch
Slide bifurcation to optimal position resulting in maximum tree fitness
Associate this fitness value with the branch

Figure 1: Flow chart depicting the overall progress of the algorithm.

We will describe the steps needed to download and build the needed software libraries and tools: CMake, ITK, and VascuSynth (Section 3). Then, we provide a detailed description of how to supply the input to the algorithm (Section 4) and interpret the output results Section 5.

3 Downloading and Building the Software Libraries

The first step is to download the software libraries as follows.

1. If you do not have ITK installed on your machine then you must follow this step, otherwise skip to step 2. Download and compile ITK following the directions posted on the ITK website [2]. This

[2]http://www.itk.org/ITK/project/welcome.html

includes downloading and working with CMake [3]. Our tests have been performed with ITK 3.18.0 and CMake 2.8.1.

2. Download the files accompanying this document, i.e. the VascuSynth source code.
3. Generate the Makefile for the VascuSynth.
    (a) Open the CMake GUI.
    (b) In CMake, click "Browse Source..." and select the directory where the VascuSynth source code resides.
    (c) Click "Browse Build..." and select the directory where you want the to build the binaries. This can be the same folder as above but preferably in a separate folder.
    (d) Click "Configure".
    (e) If the area below the "Browse Source..." and "Browse Build..." buttons is red, the "Generate" button is disabled and the value of `ITK_DIR` is `ITK_DIR-NOTFOUND`, click the value field of `ITK_DIR`.
    (f) Click the "..." button and select the directory where you downloaded and compiled ITK above. This folder is one level above the location of the ITK binaries (the "bin" folder).
    (g) Click "Configure" again.
    (h) The "Generate" button should now be enabled, click "Generate".
    (i) Exit CMake.
4. Compile VascuSynth
    - OSX and Linux
        (a) Open a new Terminal.
        (b) Navigate to the VascuSynth source directory.
        (c) Run the following command: `make`
    - Visual Studio in Windows
        (a) Open Visual Studio.
        (b) From the File menu, select open a project/solution.
        (c) Navigate to the directory where the VascuSynth source code resides and select the VascuSynth project file.
        (d) From the Build menu, select Build Solution.

4 Specifying the Input Parameters and Running VascuSynth

Eventually, the user will issue a command that looks like:

```
VascuSynth paramFiles.txt ImageNameList.txt voxelWidth noise.txt
```

where

1. `VascuSynth` is the name of the executable generated after compiling and building the VascuSynth code.

[3] http://www.cmake.org

2. `paramFiles.txt` is the name of a text file containing the names of parameter files that will be used to generate the vascular tree datasets (Section 4.1).
3. `ImageNameList.txt` is the name of a text file containing the names of the folders that will contain the generated data sets (Section 4.2).
4. `voxelWidth` is a real number describing the width of a cubed voxel in the generated volumetric image.
5. `noise.txt` is the name of a text file that specifies the type of noise to degrade the image with (Section 4.3). This parameter is optional.

4.1 Input Parameter File

The first command line argument in `paramFiles.txt` is the name of a text file. Every line in this text file points to a 'vascular parameters' filename that is used to generate a single vascular tree dataset. In its simplest form, `paramFiles.txt` will contain a single line and hence running `VascuSynth` will result in a single tree data set. For example, the single line of `paramFiles.txt` can look like this:

```
p1.txt
```

This means that the vascular parameters text file `p1.txt` contains all the parameters needed to generate the synthetic dataset. `paramFiles.txt` can include more than one vascular parameters filename, one filename per line, as in the following example:

```
p1.txt
p2.txt
```

Each filename will result in a separate dataset, i.e. the above example will generate two datasets, one governed by the vascular parameters file `p1.txt` and the other governed by `p2.txt`.

The composition of the the vascular parameters files, e.g. `p1.txt` or `p2.txt`, is described in Section 4.4.

4.2 Output Image Name File

The second command line argument in `VascuSynth paramFiles.txt imageNameList.txt`, is the name of another text file; `imageNameList.txt` in this example. This text file contains the names of the folders that are generated for each dataset to contain the 3D volumetric images and the GXL file. There must be the same number of entries in the output image name file as there are entries in the input parameter file. For example, if the input parameter file contains two lines, `p1.txt` and `p2.txt`, then the following output image name file would be valid.

```
image1
image2
```

4.3 Noise Parameter File

The last command line argument `noise.txt` is the name of another text file. This text file describes the noise that will be added to the produced image. Shadow, Gaussian, Uniform and Salt-and-Pepper (impulse) noise are supported. Multiple types of noise can be added to the image. Each line in the file contains information about the type of noise that you wish to be added to the image. The format of a line is the name of the noise type in capitals, then a colon and a space followed by the parameters for that noise type. The entries of the noise file must adhere to the following format:

```
GAUSSIAN: <mean> <sigma>
SHADOW: <numberOfShadows>
UNIFORM: <lb> <ub>
SALTPEPPER: <valSalt> <probSalt> <valPepper> <probPepper>
```

Where `<mean>` and `<sigma>` are the mean and standard deviation of the probability distribution of the Gaussian noise that is added to the image. Shadow noise adds dark spots in the image to introduce gaps in the vessel tree. `<numberOfShadows>` is the number of shadow spots to add to the image. `<lb>` and `<ub>` are the lower bound and upper bound respectively for the probability distribution of the uniform noise that is added to the image. Both the lower and upper bound can have values in the range of `[-255,255]`. `<valSalt>` is the integer value of the light noise that will be added to the image in the range of `[0,255]` and `<probSalt>` is the probability, in the range `[0,1]`, that a pixel takes this `<valSalt>` value. Similarly, `<valPepper>` is the intensity value of a pepper noise pixel in the range of `[0,255]` and `<valPepper>` is the probability that a pixel will take that value.

The noise is applied cumulatively to the image from the top of the noise file to the bottom. If multiple image names are specified then multiple vascular images will be produced, and the noise configuration specified by the noise file specifications will be applied to all images.

The user can specify many noise files as the last set of arguments in the command line, if they wish to degrade a generated image with many different noise configurations. For example, with two noise files `noise1.txt` and `noise2.txt` where `noise1.txt` specifies Gaussian noise and `noise2.txt` specifies shadow noise, for example, the command to execute VascuSynth would be as follows.

```
VascuSynth paramFiles.txt ImageNameList.txt voxelWidth noise1.txt noise2.txt
```

If two data sets are provided as input to VascuSynth (two imageNames and two paramFiles) then for each image created for each dataset, two degraded versions of the image would be produced with the different noise configurations. For example:

```
VascuSynth paramFiles.txt ImageNameList.txt voxelWidth noise1.txt noise2.txt
```

4.4 Vascular Parameter File

As discussed in Section 4.1, for every vascular tree dataset we need to prepare a vascular parameters file, e.g. `p1.txt` or `p2.txt`. Each vascular parameters file is a text file that specifies the parameters for each vascular structure that is generated. One parameter is specified per line, with a colon and a space separating the parameter name from its value. An example of the content of a vascular parameters file (e.g. the content of `p1.txt`) is:

```
SUPPLY_MAP: testS.txt
OXYGENATION_MAP: testOx.txt
```

```
RANDOM_SEED: 1
PERF_POINT: 0 50 50
PERF_PRESSURE: 133000
TERM_PRESSURE: 83000
PERF_FLOW: 8.33
RHO: 0.036
GAMMA: 3
LAMBDA: 2
MU: 1
MIN_DISTANCE: 1
NUM_NODES: 200
VOXEL_WIDTH: 0.04
CLOSEST_NEIGHBOURS: 5
```

The `SUPPLY_MAP` and `OXYGENATION_MAP` values must be strings and the `PERF_POINT` must be a triplet of integers corresponding to the location of the perforation point in the volume. All other parameter values must be numbers. For more information about the parameters, please refer to [1]. For convenience, Table 1 briefly describes each parameter in the vascular parameters file as well as the range of values. The `SUPPLY_MAP` and `OXYGENATION_MAP` are explained in the following two subsections.

Oxygenation Map

The oxygenation map is a text file that allows the user to specify the demand for oxygen in the volume. The first line of the oxygenation map is a triplet of integers corresponding to the size of the volume. The rest of the file proceeds in couples of lines. The first line is composed of six integers, where the first three integers correspond to the location in the volume of the top left corner of a box and the last three integers correspond to the location of the bottom right corner of the box in the volume. The next line corresponds to the amount of oxygen demand in the specified box, where 1 is high oxygenation demand and 0 is low oxygenation demand. Oxygenation demand values that appear later in the file will override values that precede it, if there is any overlap in the specified regions. An example oxygenation map is:

```
100 100 100
0 0 0 100 100 100
1
0 35 35 100 45 45
0
```

The first line in the above example indicates a volume of size $100 \times 100 \times 100$. The next two lines indicates a demand covering the whole volume, from corner (0,0,0) to corner (100,100,100), with a value 1 (i.e. highest demand possible). The next two lines indicate a zero demand (overriding the first) in the box specified by its two corners: (0,35,35) and (100,45,45).

Supply Map

The supply map is used by the oxygenation map to determine the effective values of the oxygenation demand. Each point in the supply map is a vector v of length n which describes the function f that is multiplied

Table 1: Parameter Specification

Parameter	**Type**	**Description**
RANDOM_SEED	Integer	The seed into the sequence of random numbers generated by the Mersenne Twister algorithm. This parameter is optional. Excluding RANDOM_SEED or setting it to 0 will ensure that no two vascular structures will be similar even if their input parameters are identical.
PERF_POINT	Integer Triplet	The location of the first branch in the vascular structure in the volume.
PERF_PRESSURE	Integer	The initial pressure at the perforation point.
TERM_PRESSURE	Integer	The pressure at the terminal nodes. The TERM_PRESSURE must always be less than the PERF_PRESSURE.
PERF_FLOW	Real	The initial flow at the perforation point.
RHO	Real	The viscosity of the fluid in the vascular structure.
GAMMA	Real	Impacts ratio of the radii of left and right branches at a bifurcation location.
LAMBDA	Real	Exponent of the fitness function where increasing lambda reduces the size of the radii in the vascular structure.
MU	Real	Exponent of the fitness function where increasing mu reduces the length of the branches in the vascular structure.
MIN_DISTANCE	Integer	The minimum distance between a bifurcation point and a candidate node.
NUM_NODES	Integer	The number of leaf (terminal) nodes to generate.
VOXEL_WIDTH	Real	The width of a cubed voxel for the supply and oxygenation map volume. This value can be different from the width of a cubed voxel for the volumetric image generated that is specified as an argument in the command line.
CLOSEST_NEIGHBOURS	Integer	The number of closest segments considered in candidate node selection.

by every point in the oxygenation map in determining the effective values. The supply map file is a text file that specifies the vectors of length n through the volume. The first line of the supply map file must be four integers where the first three specify the dimensions of the supply map (and hence the volume) and the last integer specifies the length of the vectors. The rest of the supply map proceeds in line pairs, similar to the oxygenation map. The first line in the pair specifies the region in the volume and the second line in the pair specifies the vector v of length n for that region. The first line in the pair is composed of six integers where the first three are the top left corner and the last three are the bottom right corner of a cube within the volume. The next line in the pair is composed of n numbers. This line specifies the components of the vector v for the region specified by the box in the above line. An example of a supply map file is:

```
100 100 100 4
0 0 0 100 100 100
0.65 0.34 0.01 7
```

The first line in the above example indicates a volume size of $100 \times 100 \times 100$ with a supply map vector length of 4. The next two lines indicates the values for the supply map covering the whole volume, from corner (0, 0, 0) to corner (100, 100, 100), where the components of the supply map vector are <0.65, 0.34, 0.01, 7>.

4.5 Run VascuSynth

After preparing all of the input text files as described in the previous sections, VascuSynth is ready to be executed. From the command line, navigate to the directory where the VascuSynth executable has been compiled. To run VascuSynth on a unix based system, issue the command

```
./VascuSynth paramFiles.txt imageNameList.txt 0.04
```

For windows, execute the command

```
VascuSynth.exe paramFiles.txt imageNameList.txt 0.04
```

The first parameter `paramFiles.txt` is the name input parameter file. The second parameter `imageNameLists.txt` is the name of the output image name list file. The third parameter `0.04` is the voxel width.

Executing VascuSynth will generate the output described in the next sections.

5 Interpreting the Output of VascuSynth

For each dataset, a folder is created with the name of the specified image name. Within this folder there is another folder named `original_image` and a GXL file `tree_structure.xml`. The GXL file is an XML file adhering to the GXL doctype. It contains information about the bifurcation locations and branch radii. The GXL file can be read by visualization software such as GraphViz to produce a graph that depicts the tree structure. The `original_image` folder contains the volumetric 3D image. The volumetric 3D image is saved as a series of 2D slices saved as `JPEG` images, where the image name is prefixed by `image` and suffixed with the slice number that the image corresponds to, for example `image001.jpg,image002.jpg,...,image100.jpg`.

5.1 The Volumetric Image

The volumetric image is saved as a series of 2D JPEG slices. The image can be viewed by using software such as VolView [4] by simply selecting a single slice of the image in VolView. Figure 2 Shows an example maximum intensity projection of a generated vascular structure in VolView.

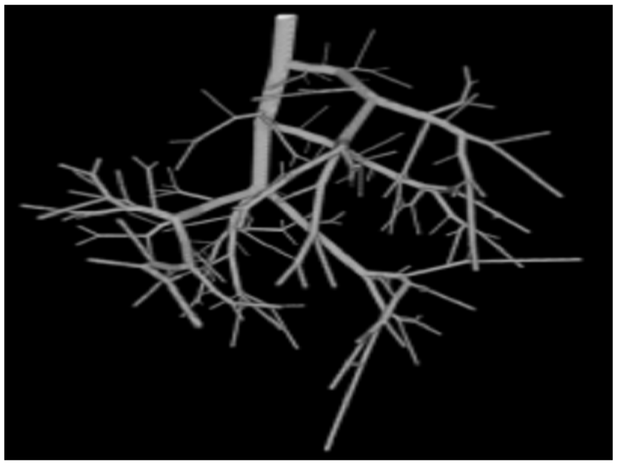

Figure 2: Sample maximum intensity projection of a generated vascular structure when viewed in VolView.

5.2 GXL File of the Tree Topology and Geometry

The GXL file contains information about the branch radii and the bifurcation locations. The following steps explain how to visualize the tree hierarchy of the vascular structure using the graph visualization software GraphViz.

1. Download and install GraphViz [5].
2. Open a new terminal or command prompt and navigate to the image directory.
3. Run the following command to transform the GXL file into DOT format:
   ```
   gxl2dot -o tree_structure.dot tree_structure.xml
   ```
4. Run GraphViz and load the newly created DOT file, e.g. `tree structure.dot`.

6 Sample Command and Parameter Files

In the folder `Testing` accompanying this submission, you can find sample parameter files that can be used to create sample synthetic data. Please refer to the `README` in the folder.

[4] http://www.volview.org
[5] http://www.graphviz.org/

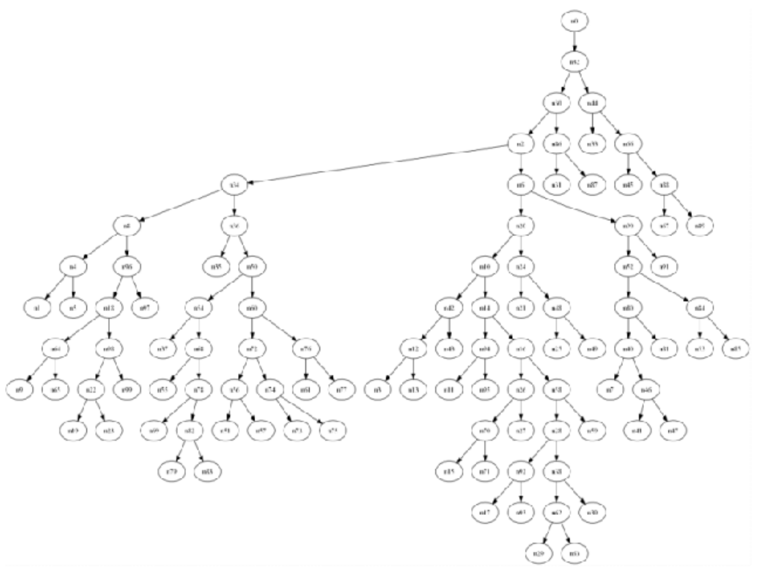

Figure 3: Sample GXL file visualized using GraphViz.

7 Conclusions and Future Work

We provided the software and documentation for the VascuSynth algorithm presented in [1]. VascuSynth generates volumetric images of vascular trees, the corresponding ground truth segmentations and produces a GXL file that contains information about the vascular structure. VascuSynth uses ITK to produce the generated vascular images as well as apply degradations to the volumetric image. Future work includes applying nonlinear warps to the volumetric image to produce a more realistic vascular structure. Additionally, the produced vascular images can be used as input to modality-specific image acquisition simulators such as POSSUM [2] or SimSET [6][3] to add further realism.

8 Acknowledgements

We thank James Styles for some of earlier code development. We thank Richard J. Wagner for provision of the Mersenne Twister random number. The Insight toolkit ITK (`www.itk.org`) is used to create the volumes, CMake (`www.cmake.org`) to build the project workspace, VolView (`www.volview.org`) to perform MIP visualization, GXL (`www.gupro.de/GXL`) to represent graph, and GraphViz (`www.graphviz.org`) to visualize the graphs.

References

[1] G. Hamarneh and P. Jassi, “VascuSynth: Simulating vascular trees for generating volumetric image data with ground truth segmentation and tree analysis,” *Computerized Medical Imaging and Graphics*, vol. 34, no. 8, pp. 605–616, 2010. (document), 1, 2, 2, 4.4, 7

[6] `http://depts.washington.edu/simset/html/simset_main.html`

[2] I. Drobnjak, D. Gavaghan, E. Suli, J. Pitt-Francis, and M. Jenkinson, “Development of a functional magnetic resonance imaging simulator for modeling realistic rigid-body motion artifacts.” *Magn Reson Med*, vol. 56, no. 2, pp. 364–80, 2006. 7

[3] R. Harrison, S. Gillispie, and T. Lewellen, “Simulation System for Emission Tomography (SimSET): Using simulation to research ideas in emission tomography (PET and SPECT),” *Journal of Nuclear Medicine*, vol. 49, no. Meeting Abstracts 1, pp. 157P–b–, 2008. [Online]. Available: http://jnumedmtg.snmjournals.org 7