CS Thesis Notebook

Andrew Hilton

November 7, 2016

Week of October 2nd, 2016

Goals for the Week

Readings

- Read John's paper on the Face-encoding Grammar [?]
- Read the paper Pete sent me on math and stuff
- Look at citations of Toussaint's paper [?] to see if there has been any work adapting his style of grammar to a similar problem space
- add citations to those reading things, just so everything is nice

Organization

- Figure out a weekly time to meet with John
- Come up with a note taking template to include in the notebook
- Add new sources to centralbib
- Retroactively digitize the contents of my physical notebook

Monday, October 3rd

General Notes

- Finally started this notebook. Going to have to figure out how to do stuff when I don't have internet access. Probably going to look into implementing some type of include or input command, and just keep weekly documents and compile them into this central document.
- Finally got internet access on my laptop up campus.

Tuesday, October 4th

Objectives for Today

Today I would like to read and take notes on the paper outlining John's tetrahedral face-encoding grammar. This means that I need to **A)** Actually add that paper to the bibliography, and **B)** work on making my note taking template, and append that to the notebook.

Wednesday, October 5th

Did a quick read-through of John's Face-Encoding Grammar paper [?]. Need to do a more in-depth note-taking read through, but the main focus of the paper seemed to be concerned with simulating the soft robots, not the uses of the grammar itself. I will have to talk with John in person a little bit more about the grammar

Thursday, October 6th

Things I Have Done Today

- Signed up for GitHub
- Added John's Face-Grammar paper to centralbib
- Moved all of my documents into Git Repository

Things I Should Do Today

I should setup a weekly meeting time with John, and try to talk with him about the next steps in my project. Now that I have a GitHub account, I might move this notebook and the work to a repository there so that I can do more stuff from command line, and not have to deal with Overleaf. If I have time today I am going to try and find some more papers talking about systems for producing Generative Encodings, and add some sections to the notebook about work I did before starting the tex notebook. Tomorrow I would like to do some more note-taking on papers.

Lab Meeting

Frankie talked about noise in simulation. According to John, Jakobi has a good paper for anyone doing stuff on Genetic Algorithms (Frankie is sending that to me over Slack)

Talk with John

After lab meeting today I forced John to talk with me about my project. He is suggesting that I use the tetrahedral face-grammar [**?**] going forward, but that I should port the existing code into python for ease of use. He is going to send me the C code that he has somewhere, afterwards I am going to rewrite the system in Python. The basic idea is that the subdivide operation is going to be removed from the grammar for ease of use (it made certain parts to complicated). After that the basic idea of the system is that there is a queue of the the faces (including labels) and every iteration you dequeue the first face, apply the appropriate rewrite rule and queue that into the next iteration's queue. The system itself really only needs to concern itself with each face, and the points it is associated with. This type of system would work well with STL format files, but Blender has some interfacing with Python so it would be good to look at both of those for the visual representation of the grammar. Next steps are to work on those pieces, and start to come up with questions myself.

0.1 Where did that algorithm type-up go?

Week of October 23rd, 2016

I have a lot to fill in for stuff but I will get to that eventually.

Tuesday, October 25th

Starting to finally get myself organized on doing the actual setup of the grammar system.

Math-stuff

- figure out the vector from one corner of a face to the center of the triangle
- figure out how to find the height of the tetrahedron (gives me the scalar to multiply cross product by)

What I got

The overview of my solution of how to calculate the new point after a grow operation has been used. Pick an edge of the face that is being operated on, and zero it by subtracting the vector from the origin to the corner ($\overrightarrow{p}$). Next take the cross product of the two vectors that define the edges of the corner. Then multiply that vector by the scalar representing the height of the tetrahedron (h) [1]. Then add the vector to the center of the face ($\overrightarrow{c}$). This should produce the extended point from the face, after it has been zeroed. Then re-add the positional vector ($\overrightarrow{p}$) to get the extended vector centered on the original face.

Turns out that this approach of taking the cross product, scaling and adding the vector to the center is **not the right solution**. The process of finding the vector to the center is exactly the same process as solving for the system of equations where the dot products are $\frac{1}{2}$. I will explain moren below.

Wednesday, October26th

I Did It?: Finally solving the Problem of the Grow Operation

So it turns out that the worst possible thing that could have happened happened, John was right... After consulting many math professors [2] and several STEM faculty and students [3], and getting many different potential solutions, through many variations, I was able to arrive at the solution proposed by John (although very indirectly and as vaguely as possible). I will now attempt to be as exact in defining the solution to the math behind the Grow operation. I will try and talk through the process it took to arrive at this solution in greater detail at a later date.

[1] make sure that the orthogonal vector is going in the correct direction from the plane
[2] Prof. Jeff Jauregui, Prof. Brenda Johnson
[3] Shelia Kang, Aaron Cass, Pete Johnorson

The Thorn in My Side

Performing the Grow operation on Face F_0, defined by the points V_1, V_2, V_3, a new point V_4 is produced, orthogonal to the plane described by F_0. This produces three new faces F_1, F_2, F_3, defined by points $V_1 - V_4$. F_0 is functionally removed from the system. In order to implement this operation, a way must be found to determine the location of V_4, given only the information intrinsic to F_0.

The approach to this problem relies on finding the vector orthogonal to the plane by taking the cross product of two vectors that describe the originating face. Then the vector must be scaled by the height of the tetrahedron that will be produced by the new faces, and repositioned to the center of the originating face, producing a point that is perpendicular to the center of the base, and a unit distance away from the three describing points. An additional positional shift will have to be performed before and after these calculations in order to compensate for the coordinates of the originating face.

Below is a more in depth description of the process.[4]

Given that F_0 is offset from its closest point V to the origin by the vector $\vec{p}$, subtract $\vec{V} - \vec{p}$ to center the point on the origin. Then define two vectors $\vec{u}, \vec{v}$ from the origin to the other points on the face. The cross product of these vectors $\vec{u} \times \vec{v}$ produces the vector orthogonal to the plane described by the face. This is then scaled to be the height of the tetrahedron $h = \sqrt{\frac{2}{3}}$. This is then added to the vector $\vec{c}$, which is defined as the point at the center of the triangular face [5]. This should then produce the vector $\vec{w}$ with length h, perpendicular to the face, at the point above the center. The original positional vector $\vec{p}$ is then readded to this value in order to reposition back on the originating face. Using this newly recentered value for $\vec{w}$ a new vertex can be created, along with three new faces.

Moving Forward

Having finished determining the relationships in order to implement the Grow operation I can now start moving forward with the implemention of the system itself. I will need to find some python packages that will allow me to calculate the cross product (otherwise I must implement that myself) and find some classes for the points and vectors with the necessary operations already implemented. I can then begin implementing my own additions to the system (i.e. creating the face class, and the actual computational aspect of the system). I will then work on integrating these structures into an existing modeling software and then begin on implementing the evolutionary component of the project. I would like to be done with the system implementation and some rudimentary integration with modeling software by the end of the week, so that I can sooner rather than later begin collecting some results.

[4] *Disclaimer: This relies on the assumption that the center of an equilateral triangle can be found by taking the average of the 3 points, which I have not verified*

[5] found by taking the average of the 3 points defining the triangle

Beginnning to Implement

I started to implement some of the classes. I am most likely going to be relying heavily on NumPy, as it is readily available, and it seems to cover most of my needs. I am going to be using NumPy arrays as the underlying representation of the Vertices, but I wanted to wrap it in something so that I could make changes and tweak the interface. Each face has an list of 3 vertices that it, and its label. Need to figure out some of the exact details of the implementation (such as how the vertices are stored). One of the decisions I have to make for the system at large is that I need to decide whether or not to create a new class for the grammar rules themselves, or to keep that part of the system simplified as a library. I will just use the default list as a queue because the functionality is already built in to those as a data structure.

Thursday, October 27th

Bio-Seminar

John told me that the Bio seminar for this week might be related to my thesis in some way. It is one of Professor Theodosiou's colleagues, who is doing work on developmental biology, specifically how organs develop. *Molecular control of physical forces during morphogenesis of the vertebrate gut*[6] by Nandan Nerurkar.

Developmental Biology is the study of how organisms go from embryonic cells to fully formed organisms (ontogeny). We care because when things go wrong we get birth defects. There is the idea that any change in shape requires a physical force. Originally we thought that human development was just getting bigge (sperm cells were thought to contain just small people who got bigger). However we have since learned that there is actually a transformation occurring. D'arcy Thompson: On Growth and Form (1917), D'arcy outlined the importance of mathematics and physics in the study of biology.

Studies have shown that there is feedback between physical forces and genes. Genes are able to create forces in order to change the morphology of physical tissues, but physical forces (such as magnetic) can actually affect the way that genes get expressed. The main question of Nerurkar's research is to figure out how developmental signals modulate physical forces to shape developing embryo, specifically by looking at the small intestine, because it is important, and interestingly shaped. The intestines look like they are jumbled together randomly, but they are actually very specifically arranged. This is shown by the number of loops in the small intestine being conserved throughout members of a species, independent of size. Because the shape is important, when the process of looping goes wrong, it can lead to serious birth defects in infants.

The Morphogensis of the Gut: The digestive system starts off as a single tube that encompasses all of the necessary functions, although not well. As the gut continues to develop different regions of the tube begin to separate into different organs, and begin to take the shape necessary for their specific

[6]apparently gut is considered a biology term

function. They were able to observe that there is a physical interaction between two types of tissue in order to create the looping of the intestine. There is a membrane on the side of the tube, that grows at a slightly slower rate than the tube itself, which causes the looping.

I kind of stopped paying attention for a bit, but it seems like they were using gene inhibition and different viruses and bacteria to try and create physical changes in the system in order to affect the shape that the organs develop, or were able to observe the developmental signals. They then used the information on the developmental signals to study the physical forces at play in development, and how the two things interact.

Implementation Work

I think the next step is to finish implementing the face and vertex classes, test them, and begin working on the actual grammar production system. I still need to answer a couple of design questions, but I think that I can be pretty successful with a fairly basic system. I am probably going to just implement the rules and operations as a dictionary mapping labels onto a tuple containing the function and the list of arguments to the operation. A question I will have to answer once I get to the evolutionary component of the system (probably worth talking about sooner rather than later) is how the number of arguments to a given operation will be determined automatically. *Is it possible to ask a function (as a first class object) how many arguments it takes?* I then need to work on the modeling interface, which will require looking into the modeling software I have available to me. I know that there is a module that allows you to interact with STL fairly easily.

Questions to Answer:

- Where will the calculations for vertex location be located? It is mostly used for the grow operation, but it will be needed for the setup.
- How will the initial state of the system be setup? Will it be populated by multiple tetrahedra, or will it just start with a single tetrahedron with a random assortment of labels and proceed from there?
- Should the initial setup of a tetrahedron rely on the grow operation? That would basically bring the amount of information needed to initially setup the system would be initializing a single face, and picking a direction for the cross product.
- How will the positional vector $\vec{p}$ be determined? Initially I figured it should just be the one closest to the origin, but would that make the calculation more difficult if that cross product goes in the "wrong direction"? I think the best way of doing this would be to find the normalized cross product, and multiply it by h and $-h$, in order to find both directions. Intuitively it seems like the vertex extension calculations in this case would produce

2 vertices, one that already existed (opposite the face being operated on) and one that does not, which is the new one that gets produced by the operation. However there may be some problems if somehow two faces end up being parallel to each other. I don't actually know if it is possible to have that kind of collision in a tetrahedral mesh system, that might be why people like them.

Friday, October 28th

Work I've done today

Today I was mainly working on implementing the 3D grammar. I think that I have finished working on the Face class, and I am considering just getting rid of the Vertex class because it doesn't really add anything functionally, and it creates another level of abstraction that I just end up ignoring because I have to have access to the array objects themselves because I rely heavily on their operations. The way that I am currently doing the calculations for the Grow operation is by delegating the calculation of the points to the Faces themselves, returning both options (the negative and positive direction) and figuring out which one is already in existence in the system, and then just working with the other one. I want to be mostly, if not all done with implementing the back end of the grammar system by tonight, so that way I can work on interfacing with the 3D modeling software and hopefully have that finished by Monday.

I need to re-type the algorithm in this notebook just so that I can have it in writing (I also think that I at some point got rid of a few lines in the notebook) [7].

the next thing that I need to figure out for the next part of the system is how to implement the actual grammar production part of the system. I could either implement it as a script that runs for a given number of generations, using a given configuration, with a given set of production rules.

¡¡¡¡¡¡¡ HEAD

Week of October 30th, 2016

Sunday, October 30th

Today I mostly worked on testing the math behind the Face cacluations, to see if it worked with the modules I was using. I discovered a slight rounding error in the calcExtension function of the Face class, but it was very miniscule[8]. I was able to circumvent this issue by switching from testing equality, to just checking to see if two points were in a threshold of each other. This was done by finding the length of $\vec{A} - \vec{B}$. After I was done with testing I started working on the modeling functionality. I started off by looking into STL formatting, and seeing if I could convert my data into meshes. This seemed kind of promising, except

[7] I think I forgot to push from my laptop at somepoint

[8] somewhere in the 10^{-17}th place

that the module I found to help with this was **A)** not well documented and hard to understand, **B)** it wasn't compatible with Python 3, which is what I plan on writing the rest of the system in anyways, and **C)** the program I would use to open and view the stl files would most likely be Blender, which has a lot of builtin Python compatability. This lead me to investigate the Python API for Blender, which looks promising. I will have to spend some time figuring out the best way to approach the problem from this side, and it may turn out that the STL formatting stuff is just the right level of complexity and extendibility that I am looking for. Though if I do end up moving forward with the blender approach, and I end up needing an stl down the line, I can just use the Blender tools to export as an STL.

Wednesday, November 2nd

I have a lot to catch up on as I have not been keeping the notebook updated with what I have been doing. Being sick has made work a little bit harder but I just made some real progress so things are starting to look up. The big benchmark that I just hit was completing the implementation of the backend of the grammar. This means that I have a system, that given a mapping of strings representing the production rules of the grammar (the "genotype" of the system) it returns an object that encapsulates the data regarding the faces being operated on, the vertices present in the system, and the number of iterations that have been done. I plan on making it so that you can alter the initial configuration of the system (at least the initial labels of the faces. I might keep it constrained to one tetrahedron at a fixed location to start). One thing I realized is that my system is a modified Model-View-Controller pattern. The model is the grammar, the view is the 3D models, and the controller is the evolutionary algorithm[9]. The next step is to work on implementing the interface between the backend grammar data, and the 3D modeling software. After that I can work on the evolutionary system.

Design Decisions

For the design of the grammar I made some initial decisions based on what I felt would make transferring the data from the grammar into the other parts of the system (specifically the 3D view). I wanted to be able to encapsulate the information about both the faces, and the vertices because I felt that it would make constructing the 3D model easier if there was a master list of all the vertices, rather than having to go through each face and construct it point by point[10]. Because of this I created the "GrammarRun" class, which encapsulates this information. Using this approach lead to other issues in where data needed

[9]The Model (Grammar) is altered by the genome passed into it by the Controller (the GA), the updated data from the Model is then passed on to the View (3D model), which is used to ascertain information about the Model (i.e. volume, surface area, etc.). This information is then passed back to the Controller, which uses it to inform its decisions about the next step in the GA

[10]not sure if this is actually going to be easier, but I feel like my method is more extendable

to be kept and what had access to it, specifically in where the production rules would be defined, and how to hide information about the implementation of the Grammar from other parts of the system[11]. This lead to me creating a two part system for the grammar. There is the GrammarRun class which contains all of the information of an single instance of a grammar defined by the tetrahedral language, and the runcontroller module which defines the implementation of the operations of the grammar, and interfaces with the setup of a GrammarRun, in order to hide the functional aspect of the operations from other parts of the system overall. This means that creating an instance of a GrammarRun involves making a call to the `startGrammarRun` function of the runcontroller module, passing in a dictionary mapping between the Left Hand Sides of the grammar (the labels being operated on) and a tuple containing a string representing the operation, and a string containing the arguments of the operation[12]

Example

Grammar:

$$A \to grow(BCD)$$
$$B \to relabel(A)$$
$$\vdots$$

Mapping in Genome:

$$``A'' : (``grow'', ``BCD''), ``B'' : (``relabel'', ``A''), \ldots$$

This call to startGrammarRun will return a GrammarRun object with the given production rules. The nextStep, and nRun methods of the GrammarRun are then used to perform the actual productions of the grammar.

Next Steps

Now that I have my internal representation of the grammar set up, I need to work on interfacing with 3D modeling software, to actually produce models of these systems. I am most likely going to be using Blender as it has an extensive interface with Python 3, and seems to be powerful enough to have some outside functionality (as opposed to something like STL which might not be able to tell me geometric information about the shapes produced)[13]. If I am able to set up a sufficient interface between my system and the Blender API, and later down the road I need to convert things to STL in order to make them more portable, it seems as though Blender is able to do that through Python, which is nice.

[11]not strictly speaking because Python...

[12]this argument list can be given as a flat string, or a list containing the individual arguments

[13]also Blender is much more free than solidworks

Friday, November 4th

Modeling Work

After doing some research on the Blender API, and testing some of the features out myself I have figured out how to

- run a python script in blender through the command line (also how to do it in the background)
- how to add faces (as meshes) to a blender file by creating and manipulating bpy Mesh objects and Object objects, and linking them to scenes to draw them (I will create a method in the Face class that returns the list of arrays as a list of tuples)
- while I was unable to figure out how to create new files programmatically through a script, I found that I can create new scenes through bpy commands, and edit them independently. This might mean that I make a file for every run, and the different results will get written to different scenes in the file

If I want I can probably make it so that the grammar is animated by drawing each iteration to a different frame in the same scene [14]

As of right now I really just need to focus on being able to generate images programmatically and make sure they look as intended. After that I will be able to start implementing the interface between the grammar system and the model.

Saturday, November 5th

So today I figured out how to do a save as operation in blender through the Python API, which means I can automate basically the entire system. I also figured out how to actually make a package work, so I can begin to start interfacing the grammar with the modeling script. I just had to add some import statements to the init file of the package. My next step is to figure out how to actually generate the actual models themselves. I think I have been able to create faces in blender, so it is now just a matter of doing it iteratively by taking the information from the GrammarRun, translating it into a datatype that is usable by Blender, and then passing that information onto a function that actually interfaces with the API.

Week of November 6th, 2016

Sunday, November 6th

I would like to have something that resembles being a complete link between the modeling system and the grammar system at the minimum by tomorrow,

[14] I would have to do a bit of reading on working with animation tools

so that I way I can have some images generated from my own work to be able to place on my poster[15]

Ideas for modeling

I am going to be building some scripts to directly interface between the grammar package and the modeling package in order to get a better idea of what actually needs to be implemented in order to make the system functional. I want to make sure that I have a good idea of what kind of information can be kept extendable

- Whether or not it makes sense to make my own serialization file format, what is the minimum information needed to be transferred between the two systems?
- should I look into builtin python serialization (xml/pickling)?
- I want to figure out what the best datatype is to pass to the part of the modeling system in order to actually produce the faces of the thing. As of right now I think the most sensible way of doing it is to just pass it lists of tuples, with each tuple being one of the vertices of the face. Don't need to worry about passing in the list of edges because blender does that based on the list of vertices, so since I already know that each list of vertices is going to be of len(3) then I can just hard code in the list of edges based on that.

I was just looking at another site explaining using the Python API and the order that they did for data creation seems to make more sense for my needs than what I was previously doing. The order of data creation I was doing before was based on another site that had me doing it

1. Create the lists of vertices and edges
2. Create the mesh (give it it's ID)
3. Create an object from the mesh
4. (*Optional*) Set the location of the object to that of the cursor [16]
5. Link the object to the current scene
6. Change the location of the mesh to the python data
7. Update the mesh

The new site on the other hand has me doing it this way

1. setup the lists of edge and vertex data

[15] which is due **TUESDAY**

[16] this apparently just sets the orientation of the object/mesh around the cursor, rather than the origin

2. create a new mesh
3. change the mesh data to match the pydata
4. update the mesh
5. create a new object from the mesh
6. (*Optional*) overwrite the object's .data using the mesh data
7. link the object to the scene
8. (*Optional*) select the new object

Trying out grammar-blender interface

I am going to use a grammar with the following production rules to test out the interface

$A \rightarrow relabel(E)B \rightarrow grow(CDA)C \rightarrow relabel(B)D \rightarrow grow(ABF)E \rightarrow grow(FCD)F \rightarrow relabel(E)$

I will use this grammar and model the system at various break points (for starters I am planning on doing 0, 1, 5, 20, 50).

PROGRESS!

I was able to generate renders of the grammar for up to 5 iterations (and more) however I ran into a few problems. For starters blender is not rendering the shapes as regular tetrahedra, they are sort of irregularly shaped, which makes the productions of the grammar look a little bit funky. The other problem I am running into is while it only takes a couple of seconds to produce and render the first 5 or so iterations of the grammar, it was upwards of 10 minutes to get to 20 iterations. I am going to set it up to run overnight and run a clock on it to see how long it takes to run the full thing (running it in the background with the time tool and gonna save the output of that to a log somewhere). Overall pretty successful as I am only a couple of tweaks away from having a working grammar system and modeling system. Next steps will be to fix the render so it looks right, *potentially* optimize the system to not run for forever, and to make my poster.