BROWN WESTRICK: Good morning. My name is Brown Westrick, and I'm going to be talking to you about the speech synthesis project. Our main goal for the speech synthesis project was to create simulated speech using a model of the vocal tract in which we would model the flow of air over time. There's existing software called new speech that already does this. And we want to deport it to cell and then improve the speech quality that it would afford us by using additional computational cycles.

So again, new speech was originally developed for linguistics research but now it's available for free under the new public license. It already models airflow in the vocal tract in real time. What this means is that there are no pre-recorded sounds. Many speech synthesizers nowadays have very large dictionaries of sounds that they just piece together and then try to smooth the transition between them. However, this model attempts to actually do what the vocal tract is doing instead of just trying to imitate the end result.

The quality of the speech, of this synthesizer, the way that it exists is not as high as the current ones that use the recorded libraries. But it has potential to be much better because you have so much finer control over all the different parameters. Our goal was that we have this software that would make an acceptable speech in real time. We are hoping it would be able to take advantage of the additional computational power of cell to be able to get an increase in speech quality. And I will it over to Drew who will tell you about the new speech system.

DREW ALTSCHUL: So new speech is made up of three different major parts. The first of which is just called the new speech engine. The second of which is, which is probably the largest part of it, which is called Monet. And then the tube resonance model, which is the

final part that actually outputs the sound. And as [INAUDIBLE], the basic processes is you take a text input standard string and the new speech engine will take care of and transform it into basic phonetic information, which they will then deal with and will take that string and eventually convert it into these what we call vocal tract parameters, which are basically parameters that can be sent to the tube resonance model.

And those parameters will define exactly how this tube, which represents the throat and the nasal tract, changes over time to represent speech sounds. And with those parameters, you can send a signal through it and create a voice. So the first part of the example will take a perfectly normal string, like "all your base are belong to us," and transform it into what we call the phonetic format of it. And you can see it highlighted.

The actual sounds are highlighted, whereas various parameters are also included in the output string, like /w, and you can determine where the words are. And [INAUDIBLE] which determine where sentences and various phrases end. And basically, Gnuspeech makes uses of dictionary files as well as some basic linguistic models in order to create this phonetic output from the basic input string.

So having created that phonetic model, you can then send it to Monet, which is by far the largest part of the program, which in turn will take the phonetic information, and as I said, use what a basic diphone file, which takes a very large range of sounds and characters and will then transform these phonetics into direct parameters, which can represent the changing of a throat the entire nasal tract as you voice your own speech.

So Monet has to go through a long process of calculating these phrases given the whole-- the rhythm, the intonation of the phrase that's being given to Monet. And also, a very important part of the Monet process is by taking each phrase and the postures-- is what we call the output. Looking at the phrase and piece by piece examining the sounds and realizing that as the postures of the throat change, there are important changes being made between there.

And basically having some sort of-- basically a slow change between them, not a gradual conversion as opposed to a sudden change in the actual shape. So then having outputted the basic postures, you finally send it to the Tube Resonance Model, or TRM, which will take the vocal tract which is divided into eight sections in this model. And send a signal off of a sine wave through the modified tube resonance model. And therefore, all the changes that occur as time goes on and the postures which change the width of the tube at various stages will then cause different speech patterns to come out and basically create an actual speech pattern, which is usually recognized.

So basically you have these three parts from a basic string to phonetics to throat postures until finally you get the actual speech out. So now I'm handing it over to Joyce to talk a little bit about the resources and algorithms.

JOYCE CHEN: Well, before I talk about the resources and algorithms, I'll talk a little bit about the TRM, which is the tube resonance model. So we already talked about how Monet outputs like two parameters based on transitions between different words and postures and so on. And the tube resonance model actually simulates the physics of the vocal tract. First you have a glottal source. If you have done any linguistics, you might have heard the little clicking sound the glottal source makes.

There are different ways to simulate the glottal source. Now, ideally, the way you have a good, natural glottal source is you have a simulation of the physics between two oscillating masses as you air passes through them. Now, back in the days when, say, people were doing speech research on gnu speech, actually simulating the physics of glottis was not possible.

So what they did instead was, you know, they would try a half sine wave or they would research the most natural sounding glottal pulse shape, initialize a wave table, and do table lookup on it, updating it with the amplitude and so on to change it a little by little. So one of our goals was possibly to-- because we harnessed the additional computational power and make more natural sounding speech.

And now I'll talk about allocating the resources. For example, in new speech Monet,

there is not as much computation. Monet has a lot of rules. For example, between postures and postures, like different shapes of vocal tracts, you can't just do an linear interpolation to smoothly change. There are different rules beach in order to change between the postures which greatly affects the speech. This was much harder to improve on, like to parallelize.

Then the tube resonance model. It had a lot more computation. In fact, the thing that took probably most computation was after we got our signal data from the mouth end of the simulation, we would have to up sample or down sample it. And that was something that had a lot of potential to be parallelized. However, when you were simulating the tube presence model, you could only update the signal inside the vocal tract incrementally.

If you were to break it up, there was a possibility that there would be a lot of pops in between when you try to space them together. We thought about trying to resolve that with interpolation between forms. There were nested loops. The main synthesized thing had nested loops. You had a posture. And then you simulate on the posture and between the postures. And that took the most computation, as well as updating the glottal wave table. All right. Now I will hand it off to Omari to explain the challenges.

SPEAKER 5: So the TRM algorithm, which is-- we're most focused on trying to-- so we most tried to focus on parallelizing the TRM algorithm. Because both Gnuspeech and Monet are almost entirely just dictionary lookups involving large amounts of memory with not that much computation. So there wasn't really that much potential for parallelizing those. So we looked at the tasks that were being done on TRM and profiled them.

And you can see what took most of the time was the noise generator part, like the attempt at the glottal source that was being put into the tubes and the actual updates where the tubes are supposed to be as they were shifting. And so unfortunately, the main loop as this was updating was very, very fast. It was about 15 microseconds. So it would be pretty difficult to update, for example, several

SPUs as fast as we needed them to, considering how communication costs affect them.

So parallelism was not very simple to use. Our main, original idea for this exploiting parallelism was to make a pipeline of the various parts of the TRM model, maybe using three or four SPUs, like one for each part of the throat as a sound would go from one to the next. So that all of them would be engaged simultaneously, like going from one posture to the next in a linear fashion. But unfortunately, the timing for this was very, very fast, in the order of about 70 kilohertz, which is many times a second for SPUs to be transferring data back and forth to each other with mailboxes and memory.

So that was somewhat difficult.

OMARI: Unfortunately, with this project, we faced a number of challenges, the first and foremost being that Gnuspeech is written in a programming language most of us weren't familiar with. And it's huge. Monet, for example, is 30,000 lines. It's hardly documented. And it took a fair amount of time just reading through and figuring out what was going on.

Additionally, because it uses Gnustep, which is a GUI library, the calls are asynchronous and it makes it tremendously difficult to debug as well. I had tried to convert part of it to C++ to try to get the tube running on one of the SPEs, and that took three days in and of itself. And I ended up having to toss that attempt out. Another problem we had was dynamic pointer alignment. In Objective C, most of the objects are stored as dynamic pointers.

And basically in Objective C, there's also no malloc alignment or anything of that nature. So we couldn't really transfer any of the objects from Objective C memory area to the SPUs to work on the data and then send them back. So what is working now? We are able to do line buffered text in the Gnuspeech engine and translate that to utterances so the-- phonetic pronunciations. And get to the point where we would execute the tube model.

Unfortunately, one of the-- a bug in Gnuspeech potentially is preventing us from properly executing the tube model right now. So that's one thing we're having problems with. Additionally, the tube runs on the PPE. We've been trying to get the tube to run on the SPE, but it's not going well, partly because of the dynamic pointer alignment issue and partly because of some other things we've run into.

Currently not working. As Drew had mentioned, there are a lot of dictionary lookups in the preprocessing stage of the pipeline. And there's a bug in Gnustep where it won't parse the dictionary if it's above a certain size. The dictionary has I believe 70,000 entries, it takes up almost 3 megabytes. But if there are more than like 3,000 entries in the dictionary, it just doesn't parse. And we have no idea why.

So to conclude, this was a tremendously difficult problem. There are a bunch of data dependencies and the synchronization is very, very close. However, we feel that with more time and with more experience with the code base, we would have been able to parallelize it. And the parallelization can almost definitely help the vocal quality in terms of naturalness. Getting, for example, as Joyce mentioned, a higher quality glottal source. Speaker identification and vowel identification.

For example, when you pronounce different vowels, sometimes quality the glottal source changes. And lastly, it would be worth the time to re-write the whole thing from scratch, skipping Gnustep, skipping Objective C, and going with C++ or most likely C for the whole thing. Thank you. Any questions?

AUDIENCE: It sounds like you guys are taking a fine-grained approach in which you're splitting the application across different units. Since you're synthesizing completely independent words, let's say, could you just run the whole application on an SPU? There's engineering there, but from a parallelization standpoint, can you just take the whole application and run it, for example, on different words?

OMARI: So I believe you're suggesting we run the tube on different SPEs and then feed data to the separate instances of the tube from the PPE?

AUDIENCE: Well, including-- the whole processing backline. I mean, order the sentences from

one sentence to the next?

OMARI: So the big stumbling block for us was that there isn't currently an Objective C compiler for the SPEs, so we can't run the Objective C code on the SPEs at all.

AUDIENCE: But if you did it from scratch, if you were to write your own-- just throw away all your Objective C and start from scratch, would that be a better parallelization strategy than a fine-grained one?

OMARI: Possibly. So one of the disadvantages of splitting up words is that there is state that connects the different-- [INAUDIBLE] continuous state that connects the different postures vocal tract all the way through the utterance. And so we would have to do possibly some prediction, possibly some interpolation to figure out how to connect the different, separate utterances which should been produced consecutively.

AUDIENCE: Permitting one sentence at a time or something?

OMARI: Yeah. That might be an option, yes.