Planetarium Visualization System
SOFTWARE ENGINEERING • PROFESSOR DALE PARSON

Nicole Cresse • Michael Pandel • Brandon Stack
Table of Contents

1. Introduction: Summary of Design Spec Document Page 3

2. System Overview Page 4

3. Design Considerations Page 5
   3.1 Hardware considerations
   3.2 Software considerations
   3.3 End-user Environment

4. Architectural Strategies Page 8
   4.1 Processing
      4.1.1 Minim Library
      4.1.2 Other Libraries and Resources
   4.2 Plans for Extension or Enhancement
   4.3 Reuse of Existing Software
      4.3.1 SoundToWaveforms

5. Visualization Design Page 12
   5.1 Introduction
   5.2 First Steps
   5.3 Current Progress
   5.4 Direction

Illustrations and Images

Figure 1. Deployment Diagram for Planetarium Visualization System Page 3
Figure 2. Dataflow Diagram for Planetarium Visualization System Page 3
Figure 3. Activity Diagram for Planetarium Visualization System Page 7
Figure 4. State Machine for Visualization Page 8
Figure 5. Dataflow Diagram with Four Instrument Inputs Page 10
Figure 6. Class Diagram Page 11
Figure 7. Bars2 visualization Page 12
Figure 8. Blob2 visualization Page 13
Figure 9. Possible Animation States Created by Nate Renninger Page 15
1. Introduction: Summary of Design Specification Document (Nicole Cresse)

We have begun working on building visualization designs with Processing, and have them working with either audio signal or an audio file. These designs can be found in section 5, with detailed descriptions and information about each stage. So far we have worked with polar geometries, plotting frequency measures in a “bar” style around a center point (see Bars2 figure in section 5). We are using Nate Renninger’s design ideas to navigate our visualization designs in our Processing program (see Possible Animation States Created by Nate Renninger in section 5).

In this document, we will provide a system overview, design considerations, architectural strategies, and a visualization demo design for the Planetarium Visualization System. We will provide a general description of our current system, including its functionality and overall design. There are also design considerations to take into account, such as what hardware and software we will be working with, and what restrictions or advantages specific hardware or software will provide for our project specifically. We will also explain how the end user will be interacting with the system.

We will also be considering architectural strategies. This includes programming languages we will be using (in this case, Processing and associated libraries that we have implemented), plans for extension or enhancement of the current system, and the reuse of any existing software or code.

Lastly, we will discuss our current status of the system, in the Visualization Demo section. In this section, we will go into detail about our first steps in the project, and how we planned on moving forward. We will then move into our current progress with the system. This will include visuals from our current builds in Processing. We will also discuss the direction we plan to move with the system, and what decisions need to be made in the near future.

2. System Overview (Nicole Cresse)
Currently our visualization code is in development, but does function as expected (in testing) with an audio file or audio signal. Our program displays visual features according to the sound input it is receiving, using assorted libraries in Processing. We have decided to base our design off of polar geometries, with the center of the dome being the center point of our visuals.

**Figure 1.** (See left, Nicole Cresse) UML Deployment Diagram for Planetarium Visualization System

At the moment we have one feature plotted, and that is frequency over a period of time. As discussed in the Requirements specification document, we plan on creating a visualization that depicts unique avatars for each audio signal. For now, we are using two inputs because that is all Processing will allow us. We could extend Processing to three or more inputs if we wish to in the future, but we feel it may not be necessary until we have something working. Our goal is to create a “layered” effect with our instrument avatars, having them all based on the polar geometries discussed previously. They would change in size, shape, and color based on the inputs they are receiving from our live instruments.

**Figure 2.** (Michael Pandel) Dataflow Diagram for Planetarium Visualization System

The overall goal for our system is to have this process integrate real instruments (guitar and violin, for example), as the audio signal going into our program. Once we have achieved the visualization design desired (see section 5.3 and 5.4), we will be progressing to using live instrument input. Currently, we are focusing on our Processing code,
and setting small goals for what we want it to look and act like, getting closer and closer to a finished product. Once we have a few different avatar designs working, we will be able to layer these designs to attain the overall effect we are looking for. We may stick to using frequency and time domain for each individual instrument, but we could also pull different features from the audio (using certain Java/Processing libraries) to use in our visualizations if we so desired. This way we would see more variation in each instrument, and different analysis patterns.

3. Design Considerations (Nicole Cresse)

There are a few aspects to consider when creating our visualization software, including the hardware and software that we have access to. It is important to note what restrictions or advantages it may provide for us, in terms of current progress and our finished product. Also, we will explain how the end-user environment will be interacting with both the hardware and software that are a part of this system.

3.1 Hardware considerations (Nicole Cresse)

For our project, we have some hardware to consider while programming. First of all (and possibly most obviously) we have the planetarium projector. This limits us to a 1200 pixel diameter, which we must take into account in our Processing program. The display uses a VGA connection that our program will use as its output for visualization. Secondly, we will be using a UA-101 High Speed Audio Capture Box (or a similar model) to route our audio signals. For the sake of simplicity, say we have one electric instrument that will be connected to this box, and from there will be connected to the planetarium display.

For testing purposes, however, we will simply use the Macbook Pro for our output display. This way, we can test our software with our electric instrument as an input, and not have to bother connecting to the planetarium display because it will not always be available to us for testing purposes. We will set our max pixel width to 1200 in our program to achieve the correct size that we will eventually use for the planetarium display, when we decide to do a test-run. The Macbook will allow us to use software such as Ableton Live to process our audio signals coming from the Audio Capture Box.

3.2 Software considerations (Nicole Cresse)
In terms of software, there are a few components that are necessary to the system. These include Processing and Ableton Live, primarily. Processing has many advantages, as well as a disadvantage. Ableton Live will allow us to process our audio, and even record audio if we wish to do so for visualization purposes.

3.2.1 Processing (Nicole Cresse)

One software that we will be using heavily is Processing. This is the programming language we are utilizing for our visualizations and audio processing. Processing does however, come with limitations. We are only able to use two audio inputs by default. However, if we wish to extend Processing ourselves, we could add support for more inputs. Currently, we would be able to use two different instruments for input, unless we build in support for more. Processing does have its own benefits though. It allows us to use certain libraries that work well with audio, such as minim and beads (see 4.1.1).

We have also found www.openprocessing.org to be extremely helpful with our processing code. It is a site where users may upload open source processing projects to share with others. We have used this site to test some visualization examples, and to get ideas for our own visualizations. It has helped us build our code, and is a great reference.

3.2.2 Ableton Live (Nicole Cresse)

Once we have created our desired visualization in Processing, we may run our audio through Ableton Live. This would allow us to easily hook up an instrument to the UA-101 Audio Capture Box (or similar model) and send the audio directly into Ableton Live. This would allow us to process our audio signal before the visualization is sent to the display, and apply effects or loops to the audio to attain different effects out of the Processing program.

3.3 End-user Environment (Nicole Cresse)
To put it simply, the end user will be sitting in front of the MacBook pro with his or her instrument. From here, they can plug their instrument into the Audio Capture Box, that is connected to the MacBook Pro via USB port. Then, they will use the box as the audio input and audio output in Ableton Live, allowing the signal to come directly into Ableton and then be sent out to the planetarium. Within Ableton Live, the user may process the audio in any way that they would like before it is processed by the visualizer.

![UML Activity Diagram for Planetarium Visualization System](image)

Figure 3. (Nicole Cresse) UML Activity Diagram for Planetarium Visualization System
4. Architectural Strategies (Michael Pandel)

4.1 Processing (Michael Pandel)

Throughout the course of this project we have been using Processing to build our visualization effects. The visuals were coded to react to the audio signals of instruments. We used the minim library to help us with manipulating and distorting visuals. Other libraries have been considered for our project, such as beads, but minim seems to do exactly what we wanted.

The first visual is a circle shape object, which reacts when an audio signal is passed through. If the object detects a signal it will then shoot bar like objects outward towards the edge of the screen. We use a ‘for’ loop to build this circle object, every time the loop repeats it then plots the coordinates of the next bar. Processing will stay in this loop until the circle object has been completely placed.

The second visual is a blob shape object. This visual, like the first visual is manipulated and distorted by audio signals. To create the blob we used a polygon, integer arrays for x and y values, and another for loop to plot out the coordinates. At each coordinate a vertice on the polygon is placed. This process is almost identical to how we created the first circle object, using a ‘for’ loop to plot pixel at the proper coordinates.

The scale variable allows us to determine how many points are in the blob. We then fill this object with color. To create the over all layered effect of the blob, we ended up repeating this process three times. Each time we changed the color, to give it the layered look.

![State Machine for Visualization](image)

Figure 4. (Michael Pandel) State Machine for Visualization
4.1.1 Minim Library (Michael Pandel)

For this project we are using the minim library to do most of our audio analysis. At this point in the project we are using the audio context object to play audio. This audio goes through our processing code which allows us to test the visuals without the need of live instruments. Our team also uses the FFT object to get frequency domain information. This FFT object allows us to grab information from the audio signal. The information that is captured is then used to drive the movement of our visualizations.

4.1.2 Additional Libraries and Resources (Michael Pandel)

At this stage in the project we are only using the minim library, but we have also looked into incorporating other libraries such as Beads. Beads is a library which is written in the java language and is used to process real time audio. There are a few reasons why we would start to incorporate the beads library. One of these reasons would be to create custom pshapes, these pshapes can then be altered by various analysis variables to create manipulation and distortion.

4.2 Plans for Extension or Enhancement (Michael Pandel)

As this project continues on we are starting to see the finish project in pieces. We plan to put all the current visualizer together to build one visualizer, which will represent the overall blob design. As the first stages of this project come to an end we are starting to think about extensions and enhancements.

The first extension/enhancement that will be coded is the additional design effects. We will be taking the code, which we have already put together and start to add additional effects, along with re-constructing some of the code as needed. We will be doing this to make the visuals look slimmer and cleaner.

Once the visuals are completed we then will start to try and incorporate the idea of a “tweet” being able to control the color and or emotion of the overall design. The reasoning behind why this is an extension/enhancement and not the main focus is simply because it is a project in itself.
A big extension/enhancement opportunity that would not only help our project but future planetarium shows/projects would be to write the code which would allow more than two inputs. As of right now most libraries that we have tried working with (minim, beads) does not support more than two audio inputs. If we could find a way to code an enhancement to allow for four inputs that would be a huge accomplishment.

![Dataflow Diagram with Four Instrument Inputs](image)

**Figure 5. (Brandon Stack) Dataflow Diagram with Four Instrument Inputs**

4.3 Reuse of Existing Software (Michael Pandel)

Before this project began we all had limited to no knowledge of how to work in processing. So we did a lot of code analysis to make sure that we understood the new language. We studied code, which was giving to us by Dr. Parson from previous semesters and found ourselves looking at multiple code examples online. We wanted to get a good background on the code before we just dove into this project. Listed below are projects that we have used to help us produce our own project.

4.3.1 SoundToWaveforms (Michael Pandel)

SoundToWaveForms was our first starting point, this code was given to us by Dr. Parson so that we had something to initially look at for reference. After analysing the code and meeting up with Dr. Parson for a code session as a group. We then decided to incorporate the code which allowed us to plot our shapes on the dome. Using Dr. Parson polarToCartesian() and cartesianToPhysical() this made this process much easier.

4.3.2 MSA Fluid back js

This is a program that we found on openingprocessing.org (designed by username qdiibp), we thought it was a well designed visualizer type processing program. One of the main reasonings for
looking at this program was to analysis and understand how the visuals were working, and how we could apply certain feature into our own personal project.

4.3.3 Kuplanetarium

Kuplanetarium is processing program which was written by Professor Miller. This is a program that is using the twitter API to help grab a tweet which is hashtagged. We have added sonification based on chromatic equal temperament so that the text from the tweets is played based on its ASCII value. To implement the sound we use an example from the minim library entitled “Creating an Instrument”. This example used an extension of minims instrument type called “SineInstrument” which we amended to include a setFreq() function (Fig X). This allowed us to change the note played for a single instrument rather than creating a unique instrument for each desired note. It currently isn’t being used in our project, but is kept aside for once we reach the tweet stage.

Figure 6. (Brandon Stack) Class Diagram
5. Visualization Demo Design (Brandon Stack)

5.1 Introduction (Brandon Stack)
Currently we are focused on creating avatars for different instruments (violin, guitar, guitar/piano, sonified tweets). The avatars will be displayed on the Planetarium dome and will be reactive to sound input and possibly interactive via a twitter interface exemplified by Dr. Miller. We are building circular visual effects based on Nate Renninger’s Design Specifications (Fig. 7). In addition, we will be expanding to allow for up to 3 or 4 inputs for the visualization of each instrument as its own avatar.

5.2 First Steps (Brandon Stack)

The initial approach we chose was to develop Nate’s twitter project idea and each individual instrument avatar as a separate project. We are currently using a sound file as input for the avatar projects. Nate’s concept of possible animation states specifies three distinct avatars, each for a unique audio stream. The violin is designated as a solid line, drawn in a circle. The guitar is a solid blob object, also drawn in a circle. Both the guitar and the violin are time domain.

Figure 7. (left) Bars2 visualization

The non-specific “instrument 4”, is shown as a histogram drawn in a circle, from the frequency domain of the audio stream, along with dots at the top of the bars to show time lapse.

The “instrument 4” avatar was started first. The initial problem encountered was getting the right coordinates for a circle, which was solved by taking code from the soundToWaveforms example which had several functions built for working with polar coordinates. This example also allowed us to skip prototyping the violin avatar since that is basically done for us. The second task was determining what to use to draw the bars. Luckily, Processing’s basic shape functions included line() which draws a line. This made the task nice and easy. We simply calculated the start and stop of the line and drew it outward from a set radius at a distinct angle. these distinct angles are approximately
mapped from 0 to 2π radians based on how many FFT frequency ranges we wish to use. If you notice from the figure above, the negative space inside the figure is a perfect circle.

The next problem was figuring out how to draw the dots at the tops of the bars. Fortunately Nate’s specs did not specify how they were supposed to move. A simple solution was to just take the point for each bar that moves and use a slower refresh rate. This means that the bars can be longer than the point of the dots, but was very easy to implement. We also discovered at this stage that anything with a refresh rate slower than the draw methods looping, done by flow control inside the draw method, would need to be kept in an array of coordinates. This was to prevent a choppy display due to the object only being drawn every ‘n’ loops. Another problem we encountered was scaling the FFT output to look evenly distributed. This was solved by playing with different ways to scale what frequency ranges we used and the output values they returned based on an iterative counter.

For the guitar or blob avatar, it was suggested to use a polygon which turned out to be very practical and surprisingly capable of being smooth along the edges. This was much easier to put together because we had the bars avatar as a reference point. We used an audio context directly to get samples from the audio stream. These are, of course, already in the time domain. We used a similar technique as we did for the lines but only used the outer radius for the blobs edges.

![Figure 8. Blob visualization](Image)

5.3 Current Progress (Brandon Stack)
So far working examples for the Guitar and the “instrument 4” avatars have been developed. We started with basic functionality for each and then changed based on aesthetic critiques from Nate. Many changes were necessary and many more will follow but we are getting very close to being able to combine everything together into one program.

Since the initial stages, we have decreased the number of and also thickened the lines on the “instrument 4” avatar and slowed the refresh rates for every visual component in both the “instrument 4” and guitar avatars. The line thickness was just more closely match what Nate drew. The slowed refresh rate however was necessary to keep people from having seizures. The guitar avatar’s polygon was also given twice as many vertices which greatly smoothed its procession. With these steps already implemented, we are currently very close to Nate’s designations.

5.4 Direction (Brandon Stack)

What’s left now is to combine everything into one file, add code for multiple inputs, add the face, write code for the violin avatar, and to implement the twitter functionality for grabbing tweets marked with the “#kupplanetarium” hashtag.

The code for the face, the violin avatar, and combining everything into one file should be very easy to create. the only concern would be lagging when all of the visuals are together. however, we have already begun to use a flow control technique for different refresh rates for different objects. this could be used to pipeline and slow down the resampling of objects as much as necessary.

The real concerns are with the twitter code and using multiple inputs. Multiple inputs should be possible but the concern is, once again, lagging. We need to make sure we don’t ask too much of the processor. Some possible options for this are using two inputs that become the left and right stereo channels (Fig X). This would allow us to get two channels, and may prove to be faster than four separate mono channels. However this would mean only two unique avatars could be created without dividing input streams somehow. Another option would be to use only one mono channel but to only use frequency domain information. The frequency domain info for each visual could be jerry mandered into segments based on a corresponding instrument. These optional paths are however undesired. we will first explore the idea of 3-4
unique channels.

The Twitter functionality concerns us because we have yet to run an experiment to test if it interrupts the Processing draw method. One possible alternative is to use a socket server and client program. We could then use one laptop to periodically grab input from the Twitter database and another to run the visuals. A tertiary concept would be to ditch the idea of twitter controlled visuals and to simply use the sonification of tweets. this would be run on a separate laptop directly into the audio interface we are using.

![Possible animation states created by Nate Renninger](image-url)