**Introduction**

AudioVisual is an OpenGL/Audio project that has the intention of providing an interface between graphical models and audio streams. The objectives of this implementation are:

* **Audio** *(Meta: These are goals not features)*
  + Smooth, error-free, realtime, threaded access to an audio buffer.
  + Accessible interface providing audio analysis metrics; including pitch tracking, energy analysis..
* **User Interface**
  + Modular implementation of the immediate-mode GUI
* **Visualisation Engine**
  + Simple graphics engine with: *lighting, AA, normal mapping, texture & model loading.*
  + Model-view based rendering
  + Action-queue framework to build behaviour.
  + A High-dimensional model, with a dimension-free polytope winding and interpolation algorithm.
* **Visualisations**
  + A set of visualisations, built within the bounds of the engine, with their own setting systems
    - Tetrahedra in a field
    - Transformations on a Tesseract
* **Render Configuration**
  + The ability to run the application in two modes:
    - Realtime Mode: *View and configure audiovis from realtime buffers.*
    - Scripted Render Mode: *Configure audiovis in advance, and save output to a file*
* **Development Process**
  + Clean versioning: requirements, installer, changelogs, testing process, release process.
  + Information: description and how-to, installation instructions,

**Meta Notes**

* You should focus heavily on requirements and define them well.

**Versions**

* **On every version:**
  + Changes are logged in a changelog (mostly user-oriented)
  + An installer is made.
* **Choosing features for a version:**
  + Every version has at least one visible use-case modification, even if it is simple
  + Tick-Tock Approach:
    - Even version revisions (0.2,0.4..): are more focussed on end-user changes
    - Odd version revisions (0.1,0.3..) are more focussed on infrastructure changes

**Process**

**Scheduled Features (***Next Version:***0.3)**

* 1.a.ii.1 – Multibuffer
* 5.a.i - Render-to-file Part I
* 6.a. Unit Testing

**Urgent Features**

* 4.a. – Tetrahedra in a Field
* 1.b.i – Audio Stream Settings

**Candidate Features**

1. **Audio:**
   1. **Buffer:**
      1. Audio Analysis Metrics: *Either: Import GIST correctly or Implement DFT, energy metrics etc yourself.*
      2. Long & Interpolated buffer:
         1. Multibuffer: *The user should be able to choose not only the sample size of the recorded buffer, but a suitable multiple (1-224), or duration of these samples to be stored in a longer term buffer. (Note: you should look up deques/queues for this). The multibuffer should be appendable. Bonus points: Later, when synchronisation is implemented, the multibuffer size, buffer size, interpolation rate, should be automatically optimised.*
         2. Buffer Interpolation: *When using audio for visualisation logic, a buffer should be obtainable from the multibuffer, with a certain offset from the end in ms or samples.*
      3. Timestamp Management: (Current problem is that
   2. **Stream**
      1. **Audio Stream Settings**: *Build an interface to the Recorder that allows you to (a) see all stream properties via a public interface, (b) Modify specific stream properties, re-initialising PortAudio safely with different settings: Channel No, Left/Right, Sample Size, HostAPI, Device*.
      2. **Audio Files:**
         1. Part I: Implementation: *Build an interface that can load an audio file and play it into a buffer of the same type. (.wav only for simplicity)*
         2. Part II: Abstraction: *The audio file stream should be an implementation of an abstract stream, sharing type information with the PortAudio recorder class.*
         3. Part III: User Interface: *Allow the Audio Mode to be swapped, and let the user choose an audio file from the interface.*
2. **User Interface**
   1. **IMGUI Renderer**: *A class which holds all window instances, and holds a render function with all the main IMGUI setup/shutdown calls.*
   2. **Main Window**: *A window from which you can show/hide all other windows, itself toggleable with a hotkey.*
   3. **Audio Window**:
3. **Visualisation Engine**
   1. **View-Model System:** *(*
   2. **Audio Attribute Mapper:** *(Choose visual attributes to be mapped to audio attributes with an interface. Interface should have the ability to cycle through permutations of mappings in a gradual fashion.)*
   3. **High-Dimensional Model**:
   4. **Visualisation Transition System**: *(Needs more definition but: Use commonality in vertex/shader info to smoothly move from one visualisation to another at a suitable time)*
   5. **Camera**
      1. Part I: Initial Implementation
      2. Part II:
   6. **Visualisation Base**
      1. Abstract Configuration Settings:
4. **Visualisations**
   1. **Tetrahedra in a Field:** *(Tetrahedra are generated with some thrust in a direction, from a rotating point in the middle. They orbit the centre, and are pulled in and pushed out from the centre. They shrink over time, and at a certain size threshold they disappear.)*
      1. **Map Spatial Properties:** (Space: *Generation Rate, Field Strength, Field Deformation*, Shapes: Velocity, Colour, Shrink Rate)
      2. **Governing Audio Properties:**
   2. **Exploding Cubes**
   3. **Julia Fractal:** (A 3D Julia fractal, whose properties change with parameters of the audio.)
   4. **Parametric Geometry**
5. **Render Configuration**
   1. **Render-to-file:** *The ability to write the framebuffer to a file*.
      1. Part I : Initial Implementation: *Load example from (*[*https://stackoverflow.com/questions/3191978/how-to-use-glut-opengl-to-render-to-a-file*](https://stackoverflow.com/questions/3191978/how-to-use-glut-opengl-to-render-to-a-file)*) and build a vcxproject for it. Get libraries and produce includes & policy for it. Modularise it so that it is easy to incorporate to the main project.*
      2. Part II : Incorporate: *Incorporate the render-to-file code into the main project. Build an interface so that you can start & stop a render.*
      3. Part III : Flexibility: *Allow the user to choose the quality, and*
   2. Command-Line interface:
   3. Configuration XML files:
6. **Development Process**
   1. Implement Unit Testing for the main modules.
   2. Create installation instructions for github, with a list of dependencies.
   3. Make sure that all build configurations (x64 & x86 Debug/Release) work correctly.
   4. Create a publishable installer, with a policy to update the installer.

**Completed Features**

**Current Problems**

1. **Audio**
2. **Synchronisation**
   1. You need a way of synchronising audio and visual processing. For visualisation logic, an iteration through the logic must include: the time that has passed, the audio buffer to use for the iteration, the time travelled through the audio buffer
      1. **When Processing Visualisation Logic:** All calculations are referenced by the amount of samples that have passed.
      2. **When Getting Audio Buffer:**
      3. **When Rendering to Screen:**

**Unorganised Features**

1. You’ll want to add skyboxes. You might want to *animate* the skyboxes, and that would require them to exist as their own visualisation.
2. If you are doing audio processing in advance, and also moving mapped attributes along a ring of parameters for a visualisation, then:
   1. You’ll need to know the range of all of these properties in advance, to be able to normalise their contributions to each parameter. *Normalisation should be an option in* 
      1. For instance: if the pitch takes a value from 0.3-0.5 but the amplitude takes a value from 0.01-0.93, if they are both contributing to a property (say field strength), they need to be normalised so that the move in contribution is gradual.
3. Configuration system:
   1. You should have two options by default: Scripted & Realtime. Scripted is enabled if the program is ran with a “.xml” file.
   2. You should be able to generate a configuration from within the program.

**Example Configuration File**