**Beat Slicer**

Andrew Zhao (ahzhao)

A block slicing rhythm game, where players use motion tracking controls to swing their arms and slash through cubes that fly towards them in lanes, all in rhythm with music.

**Competitive Analysis**

Notable similar projects include Beat Saber and Fruit Ninja. Beat Saber is a 2017 hit VR game made in Unity, tasks the player with slicing through 3D cubes to the beat of music with a pair of lightsabers in each hand. My project is also a cube-slicing rhythm game, but on PC running via cmu\_112\_graphics. Instead of using a pair of IMU/accelerometer and IR tracking enabled controllers, I use a computer for tracking in my project, which finds the bright light of a mobile phone, held by the player, with its flashlight on. Graphics are planned to be drawn in 2.5D using cmu\_112\_graphics.

Fruit ninja is a mobile fruit-slicing game where the player uses their finger to cut through fruits. My game brings in music and rhythmic elements, while also using full body scale movement instead of finger swipes.

**Structural Plan**

An overarching “game” class, a subclass of App from cmu\_112\_graphics, will run the whole project. Cubes and polygons, the blade, the camera driver, the audio driver, and a 3-d spatial grid are classes that will be imported and contain specific functions for the game.

* Game Class – unites all classes/elements, is run, App subclass
* Grid3d Class – handles 3d grid-space to 2d screen-space conversions
* Poly3d Class – Movement, positioning, drawing for arbitrary 3d polygons
  + Cube subclass – known to have 6 perfect sides
* ConvexHull Class - Implementation of convex hull and face merging algorithms
* Slice3d Class - contains slicing algorithm for polyhedra and its helper functions
* Blade Class – takes points from camera driver, makes lines for slicing
* camDriver Class - OpenCV camera functions, reads + masks webcam image
* audioDriver Class – reads, stores, plays audio data
* songMaps - contain bpm, music/game sync data, music filename, block pattern

**Algorithmic Plan**

3d slicing of cubes consists of several steps:

* **Generating a plane from a 2d slice line.** I will represent planes in the form ax+by+cz=d, in code as a tuple of (a,b,c,d) constants that will be unpackaged for use. The two slice lines form a pair of points for the plane, while the third necessary point for plane generation will be the focal point of the 3d grid, resulting in planes that radiate out and are roughly consistent with the player’s 2D view even when slicing at multiple angles.
* **Splitting points of a cube according to such a plane.** Using the plane coefficient method each point can be solved as to be above or below the plane, and put into one of two resulting arrays of 3d points.
* **Generating new points for such polyhedra** by calculating intersections between cube edges and the slice plane. Since only sliceable objects will be cubes, their edge pairs can be predetermined and stored as a list of indices of the cube’s points. Thus, pairs of points defining edges can be easily checked for intersection with the plane. Finding the specific intersection point can be done by using a parametric equation for the line between the two points, and solving with the plane coefficients to get the 3d space point of intersection.
* **Correctly ordering inserted points** and new groups of points. This is not relevant if the face generation algorithm adapts for the disordered list of points.
* **Generating faces** from sets of points. I borrow a QHull implementation called convexHull.
* **Merging created faces.** The convex hull consists only of triangles, so square/pentagonal/hexagonal faces require merging triangles. I can use the previous plane function to calculate orthogonal vectors to each face, and merge faces with 2 same points (indicating a shared edge) and parallel vectors.
* **Ensuring convexity** of merged faces. Reordering would require generation of a new plane, establishment of axes and projection of points. Instead I opt to insert the second shape’s points between the 2 common points of the first, which guarantees convexity.
* **Creating new polyhedra** from the new sets of points with correct location and velocity properties. I have cube’s slicing method create new polyhedra with the new sets of points and faces, and using the known slice plane, impart 3d velocity vectors orthogonal to it.

**Music synchronization** depends on the specific pyAudio implementation I’m using. For slashing sounds, I will create threads for each sound so they can run in parallel, and use a callback function for this. For the main soundtrack which must be in sync with the game’s timer, I planned on using the game’s timerFired to call audio frames. This will not work because timing inaccuracies cannot be accommodated by stretching the song, so instead, I now have the audio driver updating cube positions. Semi-MVC violation as I’m updating data in the “view” (sound), but the implementation ended up working well.

**Timeline Plan**

TP1 – Slicing, integrated with camera, hopefully basic sounds

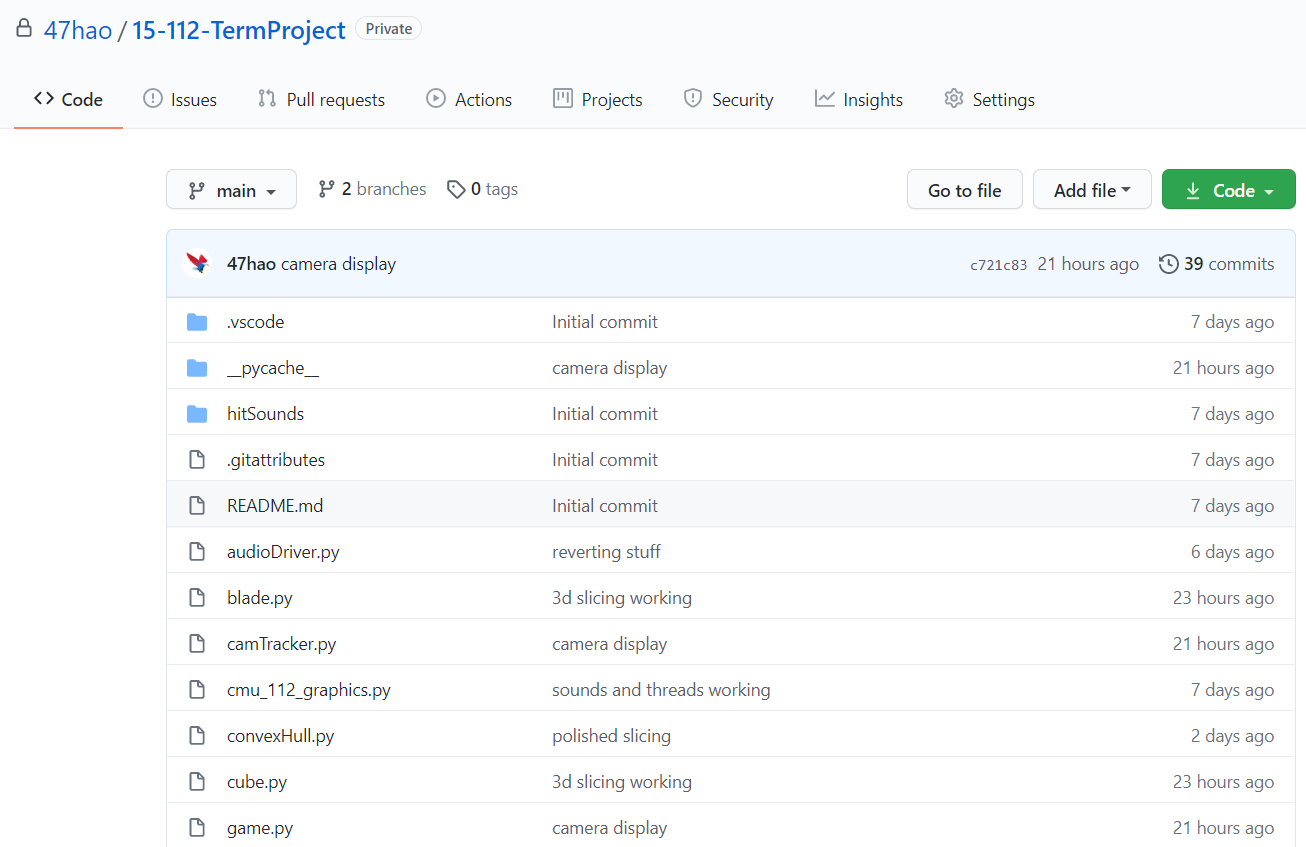
* Wednesday: music synchronization with game ticks

TP2 –scoring for slices, predetermined cube patterns coming

TP3 – Smooth menus/sounds, 2D lighting effects/better graphics, better blade, sparks flying, fragment cut mark, range of motion calibration, more song maps

**Version Control Plan**

* Github yay



**Module List**

* OpenCV
* PyAudio
* Scipy, numpy
* PIL ImageOps
* (phone) flashlight, hardware

**Sources Used**

* https://www.tutorialspoint.com/python/python\_multithreading.htm
* <https://people.csail.mit.edu/hubert/pyaudio/docs/#:~:text=To%20use%20PyAudio%2C%20first%20instantiate,PyAudio>
* https://www.youtube.com/watch?v=\_bjZ\_-VZVnU
* https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html
* https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_morphological\_ops/py\_morphological\_ops.html
* https://docs.opencv.org/master/dd/d49/tutorial\_py\_contour\_features.html
* pythonprogramming.net/color-filter-python-opencv-tutorial/
* <https://github.com/RamenBucket/112-Hackathon-20>
* Cmu\_112\_graphics <https://www.cs.cmu.edu/~112/>
* <https://www.gamasutra.com/blogs/YuChao/20170316/293814/Music_Syncing_in_Rhythm_Games.php>
* <https://bsmg.wiki/mapping/map-format.html>

**TP2 Design Update**

* Notes/beats will probably be manually mapped
* Cubes will not be drawn wireframe, but as pseudo 2D-3D polygon layers that still uses 2.5D algorithm
* Song maps will by .py files

**TP3 Updates**

* Calibration – added mode before playing the game, where the user moves their light around to set the min and max bounds in the camera view. This way the game is comfortably adapted to the player’s location and physique, instead of forcing the player to move to accommodate the computer webcam’s properties.
* Tasks for many processes are multithreaded to improve framerate, including visual animations and sliced fragment physics; can now handle many cubes with no lag
* Sparks fly outwards in random directions and sizes with each cut
* Light pulses – background pulses gray and changes color on beats, patterns prewritten into the song data files