PROCEDURAL MICROSTRUCTURE GENERATOR & EXPLORER

Mohammad Samir Elassaad, Khalid Samir Elassaad, Soroush Arjomandi

Objective

This project aims to simulate the microstructure of a relatively simple binary system. Specifically, the continuously distributed nature of phases and grain boundaries are modeled and generated in Python 3, a rather universal coding language. Three targets this project hits individually are:

- **Model distinct phase size and boundary distributions**
- **Simulate an infinite structure by generating data chunks**
- **Enable a mobile player interface to imitate microscopy**

Former Cal Poly Materials Engineering lecturer John Nelson once described microstructural phases as *environments of atomic uniformity that one could travel through.* This idea inspired the foundation of this project as it is a profound and enlightening way to think of microstructures.

Perlin Noise

Perlin noise, the first procedural noise function, has seen widespread use in both research and industry for a diverse range of purposes in procedural texturing. Noise can be described as the random number generator of computer graphics, and Perlin and Hoffert defined noise as an approximation to white noise band limited (nonzero only within a specific range of frequencies) to a single octave [1]. Procedural is the adjective used to distinguish entities that are described by program code rather than by data structures [1]. Perlin noise generally determines noise at a point in space by computing a pseudo-random gradient at each of the eight nearest vertices on the integer cubic lattice and then doing a splined interpolation.

In the case of this project, this means that the 0s will be found closer to other 0s rather than 1s, resulting in grain segregation allowing for grain boundaries to become apparent. Python's noise module including Perlin noise was used for this application.

The 3D periodic Perlin noise function has been widely used in computer graphics to generate procedural textures, because of its ability to produce a noise component that can simulate natural materials such as marble in a very realistic way [2].

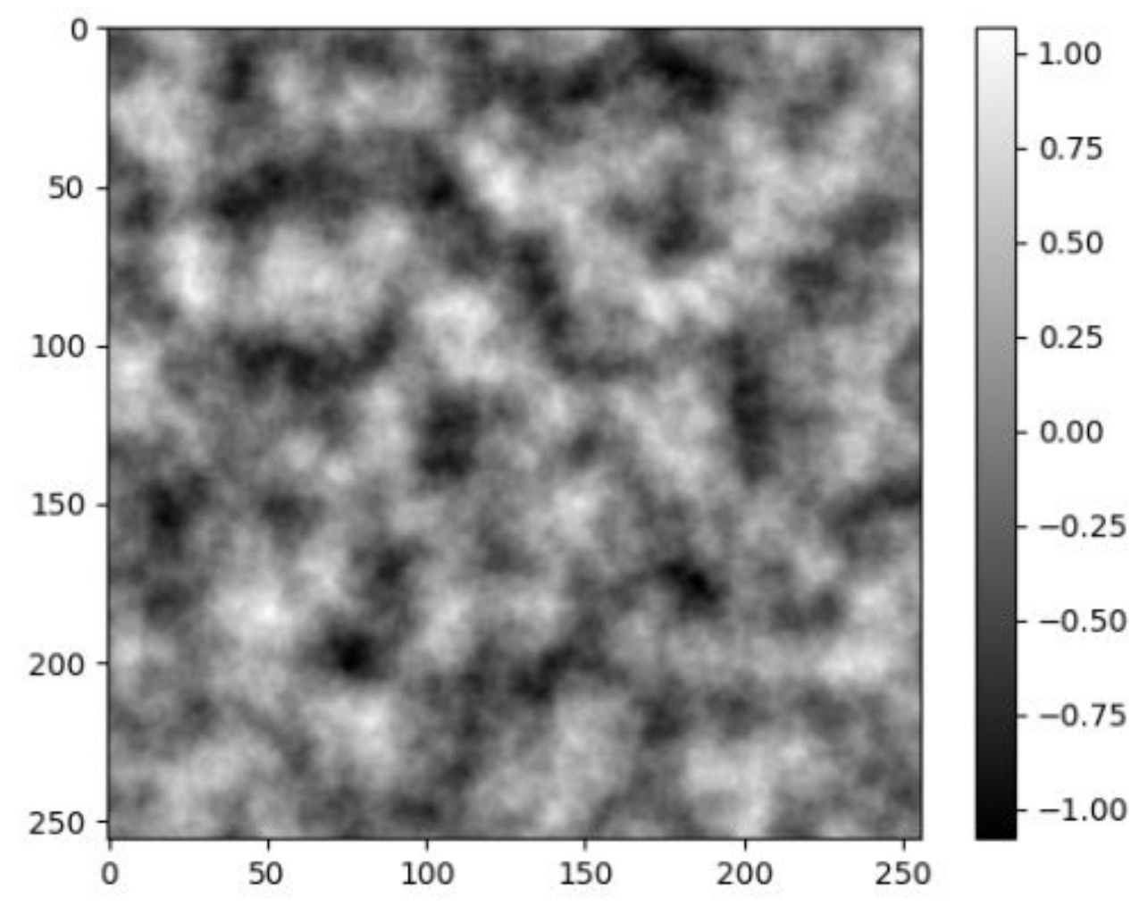
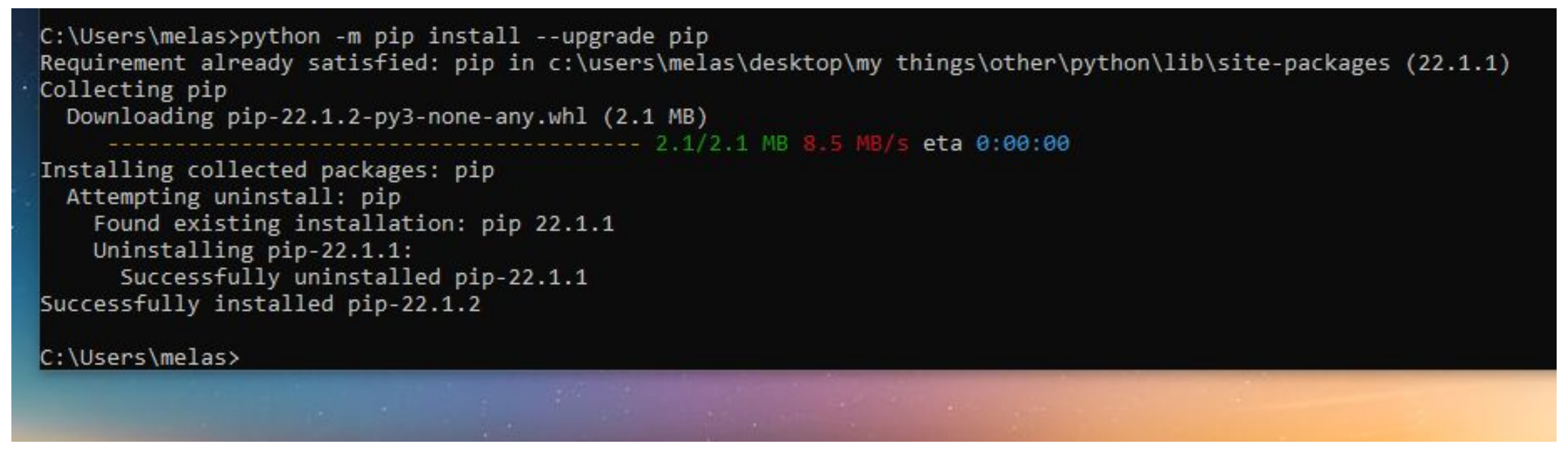


Figure 1. Raw three dimensional perlin noise visualized with a darkness gradient.

Simulation Methodology

The respective coding of this project can be broken down into four primary segments. First, perlin noise must be generated and stored in a format that is applicable to a graphics user interface. Second, multiple chunks of perlin noise must be continuously generated and then precisely fit together in order to model an infinite microstructure. Third, a player coordinate manager must be designed to keep track of the player's position to relative individual chunks, allowing for memory efficiencies and retractable pathways. Finally, an overall code must call on all these smaller scripts to assemble a workable game with user input. In order to achieve these ambitions, special Python modules are needed as shown in Figure 2.



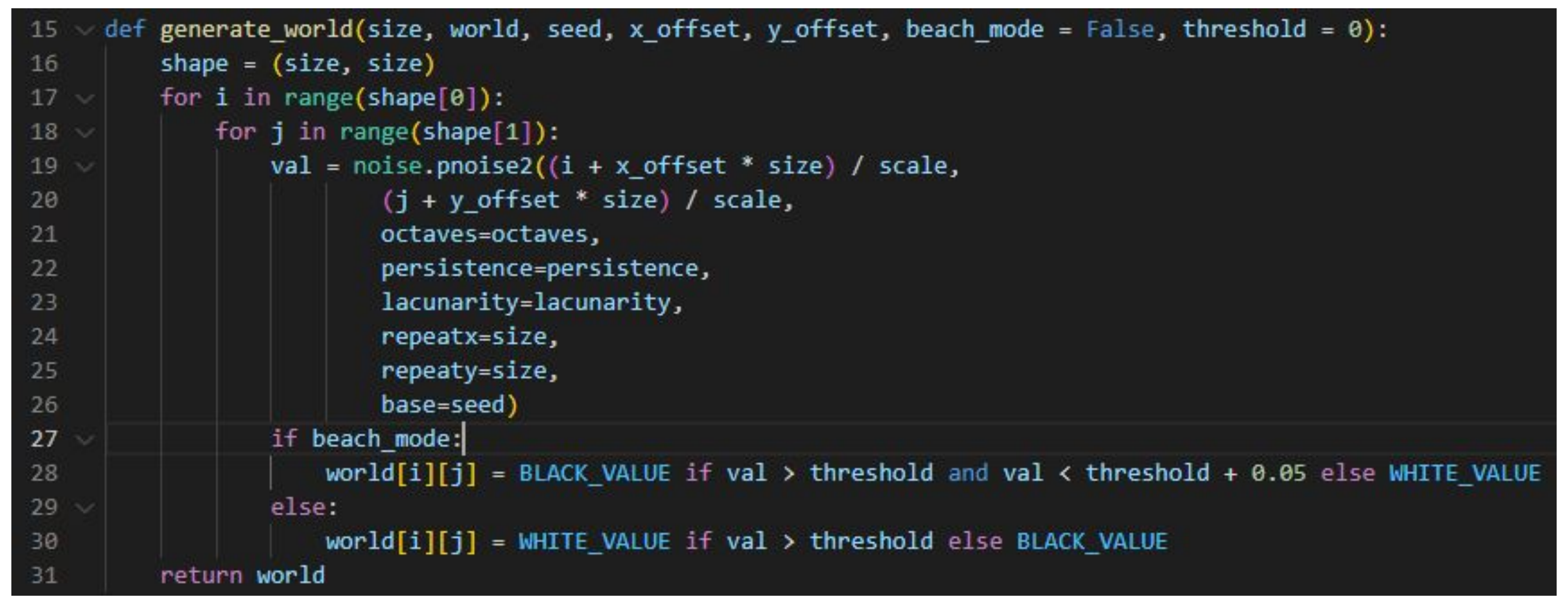
```
C:\Users\melas>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\melas\desktop\my things\other\python\lib\site-packages (22.1.1)
Collecting pip
  Downloading pip-22.1.2-py3-none-any.whl (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 8.5 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.1.1
    Uninstalling pip-22.1.1:
      Successfully uninstalled pip-22.1.1
Successfully installed pip-22.1.2

C:\Users\melas>
```

Figure 2. Relayed above is the code required to install pip via terminal. Through this relay, the necessary modules required to design and run this program are downloaded and automatically installed. Those modules include Pygame, Math, Numpy, and PIL.

In order to generate perlin noise, the function noise.pnoise2 was utilized to generate a tuple of data, which in this case was better suited than an array due to Python's negative coordinate system. Color multipliers filter the values into black and white, resulting in a binary system. The threshold determining which multiplier to use can be altered, which can increase or decrease phase concentration. The process is repeated with a finer threshold to model grain boundaries and the two data structures are overlaid to produce a simple microstructure.

A dictionary of data structures was pieced together in a format similar to a matrix. This allowed for the loading of multiple microstructures that fit together, referred to as chunks. New adjacent chunks were generated and old chunks were deleted as a data structure was selected to minimize computation and enable a procedural realm. A player was drawn and position tracked in order to mimic the feeling a microscope operator would endure when piering through the optical lens. Lastly, all essential steps were linked into a final code. Figure 3 showcases the perlin noise definition.



```python
def generate_world(size, world, seed, x_offset, y_offset, beach_mode = False, threshold = 0):
    shape = (size, size)
    for i in range(shape[0]):
        for j in range(shape[1]):
            val = noise.pnoise2((i + x_offset * size) / scale,
                    (j + y_offset * size) / scale,
                    octaves=octaves,
                    persistence=persistence,
                    lacunarity=lacunarity,
                    repeatx=size,
                    repeaty=size,
                    base=seed)
            if beach_mode:
                world[i][j] = BLACK_VALUE if val > threshold and val < threshold + 0.05 else WHITE_VALUE
            else:
                world[i][j] = WHITE_VALUE if val > threshold else BLACK_VALUE
    return world
```

Figure 3. Various parameters can alter the noise geometry like octave, persistence, and lacunarity. It is the threshold value that separates phase concentrations.

Results & Discussion

This project yields an infinite binary microstructure simulator. Travelling in any new direction will randomly generate unique chunks. The player is moved via the W, A, S, and D keys. Holding down the N and M keys will change phase and boundary concentration, modeling different compositions or perhaps heat treatments. The program runs at 10 frames per second to reduce RAM consumption, however this number can be further changed. Future versions can improve on this inefficiency. Additionally, the J and K keys could also be programed to modify grain boundary thresholds separate from phase concentrations, increasing the precision of microstructural modeling. Figure 4 displays window captures from the running program.

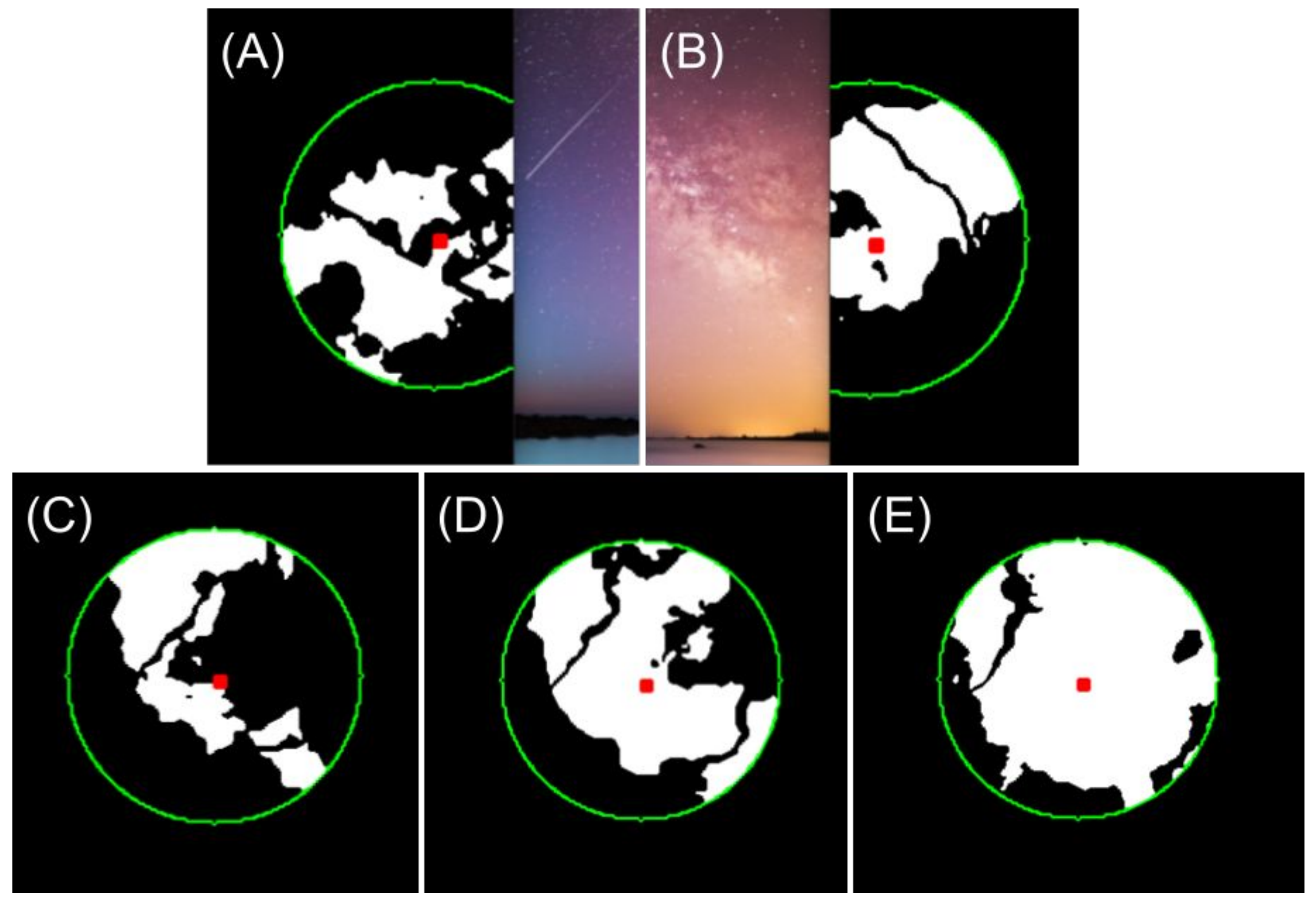


Figure 4. (A / B) Conveyed above is the relationship between the player, chunks, and the screen edge. Distinct random data structures are generated as the player hits a border. (C / D / E) The difference in microstructure phase and boundary concentration can be seen as the player alters the color scheme threshold filter.

References

[1] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions," *Computer Graphics Forum*, vol. 29, no. 8, pp. 2579–2600, 2010.

[2] F. Conde-Rodríguez, Á. L. García-Fernández, and J. C. Torres, "Modelling material microstructure using the perlin noise function," *Computer Graphics Forum*, vol. 40, no. 1, pp. 195–208, 2020.

Special Acknowledgement

Khalid Samir Elassaad is a University of California Irvine rowing national champion and computer science graduate. He is also the wise older brother of Mohammad Samir Elassaad. Khalid decided to participate in this project to share his coding experience and unique problem solving skills. This project would not exist without his extraordinary design imagination and fundamental scripting contributions.

This project was supervised by Dr. Mohsen Beyramali Kivy for MATE 403 / Spring 2022 (Computational Materials Analysis)