Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

SCHOOL OF COMPUTING AND AUGMENTED INTELLIGENCE
engineering.asu.edu/scai

PO Box 878809
Tempe, AZ 85287-8809
Ph: 480.965.3190
FAX: 480.965.2751

PROJECT PORTFOLIO COVER SHEET

Instructions: Students must complete all sections below. The completed form must be submitted to the Project Portfolio Canvas course with the project portfolio by the posted semester deadline.

Section I: Student Information

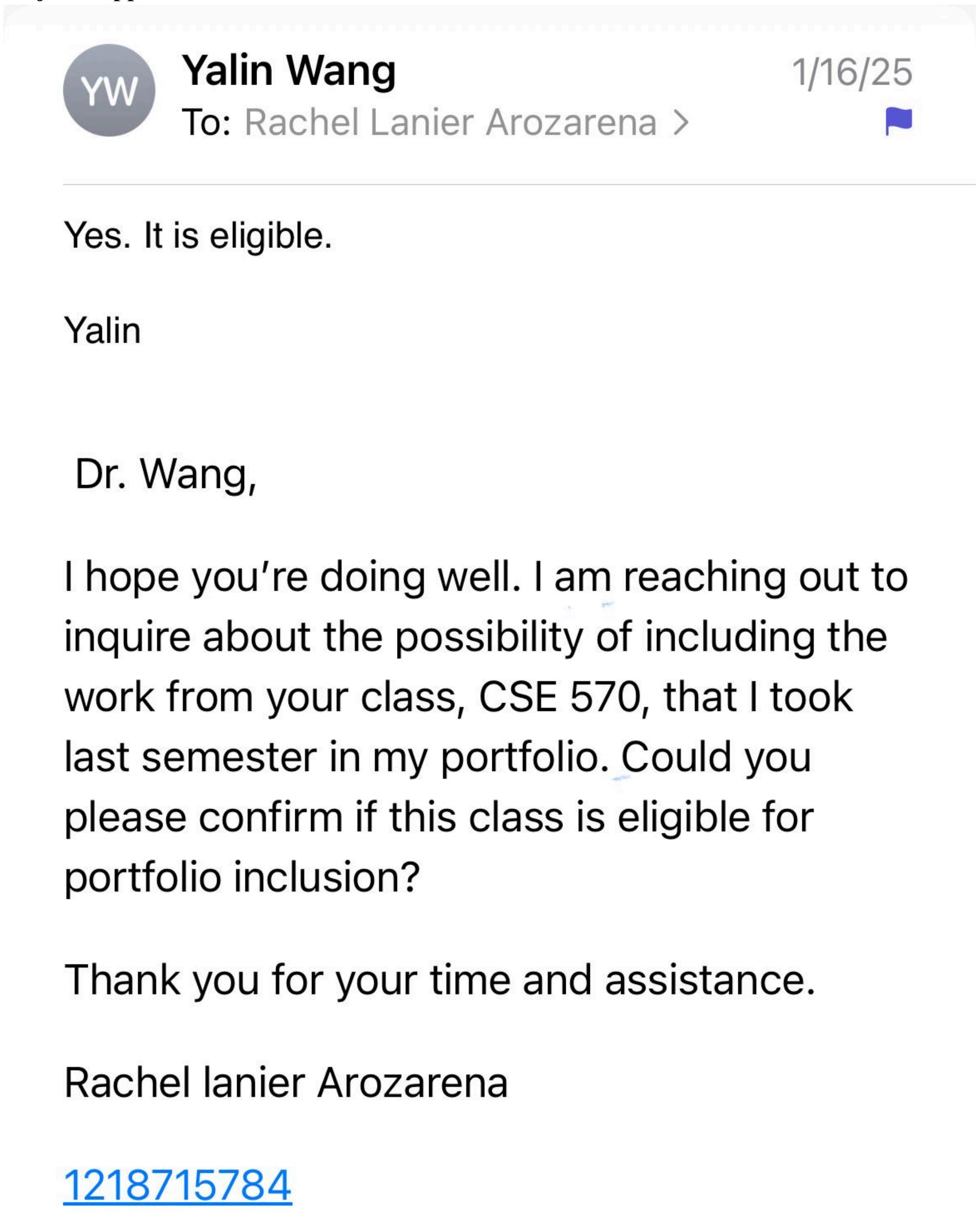
Last Name Lanier Arozarena	First Name Rachel	ASU ID 1218715784
Graduation Term Summer 2025	All GPAs above 3.0? [✓ Yes] ☐ No	Date submitted 07/14/2025

Section II: Project Information

PROJECT 1: [✓ Project worth at least 30%] OR ☐ Project under 30% with additional work done		
Course CSE 570 Advanced computer graphics I	Semester completed 2024 fall	Final grade in course B
Instructor Name Dr. Yalin Wang (has certified that the project is applicable for the portfolio) [✓ Verification of instructor approval is attached] ☐ Instructor approval list has been sent to advising office		

PROJECT 2: [✓ Project worth at least 30%] OR ☐ Project under 30% with additional work done		
Course CSE 568 Biocomputing	Semester completed 2025 spring	Final grade in course B
Instructor Name Stephanie Forrest (has certified that the project is applicable for the portfolio) [✓ Verification of instructor approval is attached] ☐ Instructor approval list has been sent to advising office		

Project 1 approval :



Yalin Wang 1/16/25

To: Rachel Lanier Arozarena >

Yes. It is eligible.

Yalin

Dr. Wang,

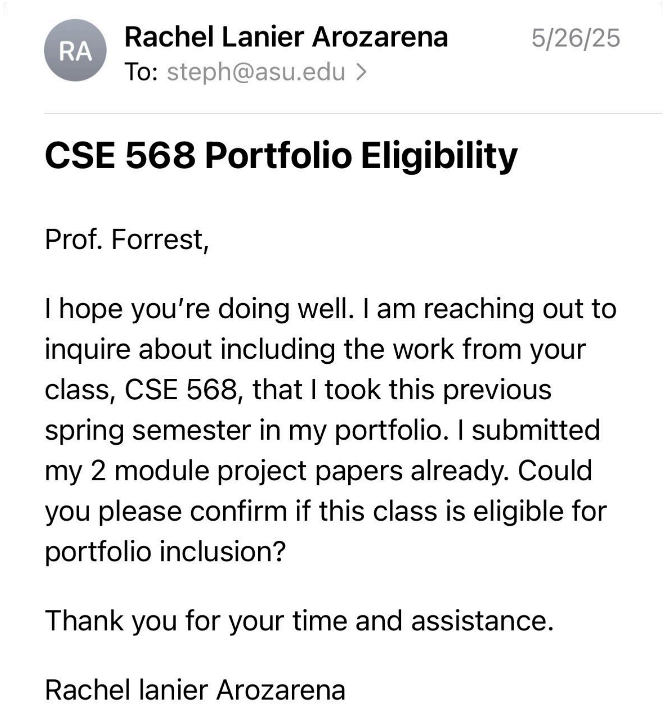
I hope you’re doing well. I am reaching out to inquire about the possibility of including the work from your class, CSE 570, that I took last semester in my portfolio. Could you please confirm if this class is eligible for portfolio inclusion?

Thank you for your time and assistance.

Rachel lanier Arozarena

1218715784

Project 2 approval:



RA Rachel Lanier Arozarena 5/26/25
To: steph@asu.edu >

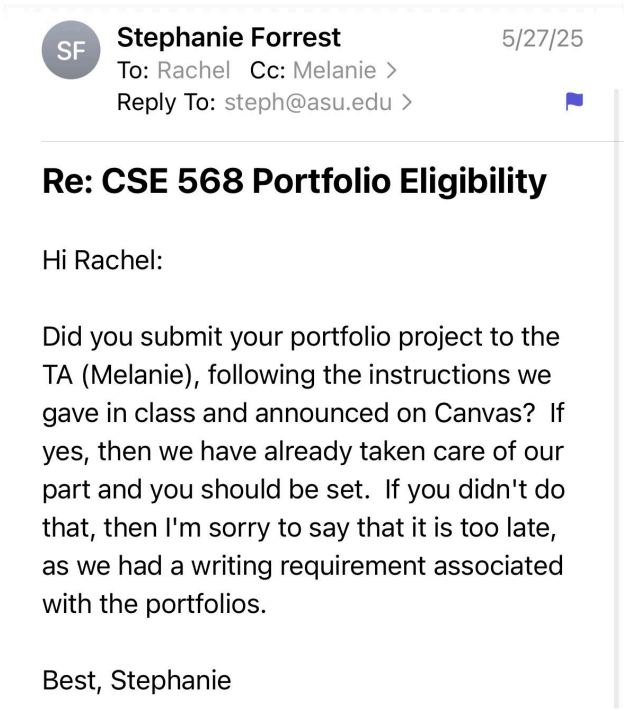
CSE 568 Portfolio Eligibility

Prof. Forrest,

I hope you're doing well. I am reaching out to inquire about including the work from your class, CSE 568, that I took this previous spring semester in my portfolio. I submitted my 2 module project papers already. Could you please confirm if this class is eligible for portfolio inclusion?

Thank you for your time and assistance.

Rachel lanier Arozarena



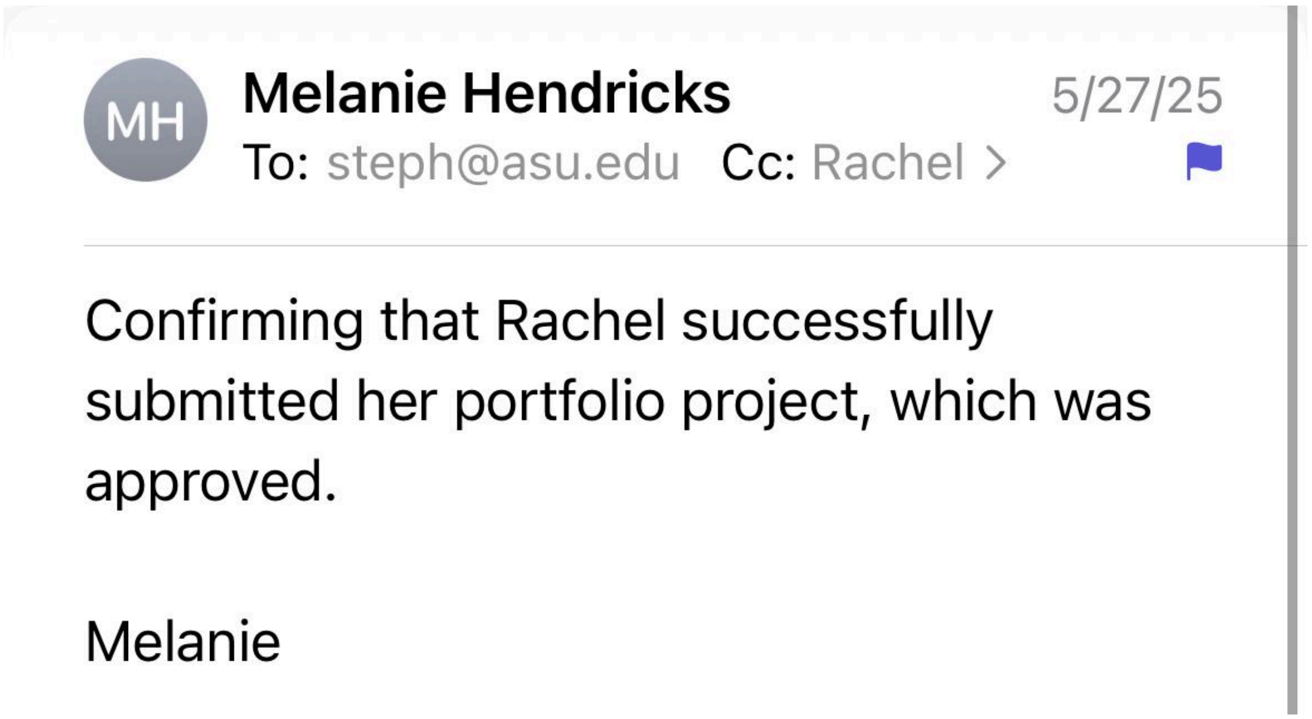
SF Stephanie Forrest 5/27/25
To: Rachel Cc: Melanie >
Reply To: steph@asu.edu >

Re: CSE 568 Portfolio Eligibility

Hi Rachel:

Did you submit your portfolio project to the TA (Melanie), following the instructions we gave in class and announced on Canvas? If yes, then we have already taken care of our part and you should be set. If you didn't do that, then I'm sorry to say that it is too late, as we had a writing requirement associated with the portfolios.

Best, Stephanie



MH Melanie Hendricks 5/27/25
To: steph@asu.edu Cc: Rachel >

Confirming that Rachel successfully submitted her portfolio project, which was approved.

Melanie

Portfolio Summary: Biocomputing and Advanced Computer Graphics Projects

Rachel Lanier Arozarena
Arizona State University
Tempe, Arizona, USA
rlaniera@asu.edu

1 INTRODUCTION

This portfolio includes highlights from two graduate-level courses: CSE 568 Biocomputing and CSE 570 Advanced Computer Graphics. All projects in both courses were completed independently. The project selections reflect my interest in biologically inspired algorithms, modeling, and 3D geometric data processing.

2 OVERVIEW OF CSE 570 REPORT

This report summarizes four individual projects completed during the CSE 570 course on Advanced Computer Graphics, each addressing a core topic from geometric analysis to learning-based 3D processing. Combining theory, implementation, and analysis, these projects provide practical insight into the 3D data manipulation, rendering, and understanding pipeline. The projects reflect independent work and experimentation and collectively account for 50% of the final course grade.

The projects include constructing a Bézier Coons patch and computing Gaussian curvature through partial derivatives and numerical methods, with visualization of control grids and vector fields. The Loop subdivision project implemented mesh refinement on standard models, correctly computing new vertex positions, though it encountered challenges in generating subdivided faces beyond two iterations. The spherical conformal mapping project achieved folding-free, angle-preserving parameterization of a genus-0 brain surface onto a unit sphere by minimizing harmonic energy via gradient descent and Möbius transformations. Finally, the MeshCNN deep learning project explored feature modifications and pooling removal, maintaining 100% classification accuracy on 3D meshes. It also analyzed GPU usage and model parameters, demonstrating the model's robustness under architectural changes.

3 OVERVIEW OF CSE 568 REPORT

This report summarizes three individual projects completed during the CSE 568 Biocomputing course, which included six projects: Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, Introduction to Negative Selection, Anomaly Detection with Negative Selection, Modeling Collective Behavior in Ants, and Ant Colony Optimization for Network Route Repair, together comprising 60% of the final grade. For this portfolio, I selected Projects 1, 2, and 5—Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, and Modeling Collective Behavior in Ants—which together account for 30% of the coursework and best represent my individual contributions and understanding. Each was implemented independently using the course materials and Zybook platform.

The first project applied genetic algorithms using DEAP to solve the NP-hard graph coloring problem by minimizing color conflicts through parsing graph files, encoding individuals as binary strings, tuning parameters, and evaluating fitness, achieving convergence to valid colorings. The second project extended this by adding balanced coloring constraints and neutrality analysis, modifying the fitness function to penalize uneven color distribution and compute 1-step neutrality, demonstrating increasing fitness and successful neutrality computation on multiple graphs. The fifth project converted an ODE model into an agent-based model of ant foraging using Mesa, simulating exploration, recruitment, and commitment behaviors on a grid; the simulation results showed robust convergence on dominant food sources with stochastic recruitment dynamics, reflecting biologically-inspired positive feedback and emergent collective behavior. Together, these projects deepened my understanding of genetic algorithms, fitness design, neutrality, agent-based modeling, and biological system simulation.

4 KEY TAKEAWAYS

These projects enhanced my skills in modeling, algorithm development, and analysis. One of the main things I learned from these projects was the interconnectedness of different subjects, specifically the applications of NP-hard problems and bio-computing algorithms, as well as the 3D brain and conformal mapping projects and their relevance to real-world scenarios. The work I completed in these courses reflects a strong foundation in computational methods and an interest in interdisciplinary problem-solving.

ACKNOWLEDGMENTS

I gratefully acknowledge the instruction and support provided by the faculty of the CSE 568 and CSE 570 courses. Their teaching and feedback were instrumental in deepening my understanding of the topics presented in this portfolio.

CSE 570: Advanced Computer Graphics Portfolio Highlights

Rachel Lanier Arozarena
Arizona State University
Tempe, Arizona, USA
rlaniera@asu.edu

ABSTRACT

This portfolio summarizes four key projects completed as part of CSE 570: Advanced Computer Graphics. These projects focused on fundamental and advanced topics in geometry processing, mesh analysis, and neural networks for 3D data. Each project was completed individually and contributed to the development of skills in both classical and modern computer graphics methodologies.

CCS CONCEPTS

• **General and reference**; • **Agent-based Modeling**; • **Computing methodologies**; • **Software and its engineering**;

KEYWORDS

Computer Graphics, Mesh Processing, Conformal Mapping, Subdivision Surfaces, MeshCNN, 3D Geometry

1 INTRODUCTION

This report presents a summary of four individual projects completed during the CSE 570 course on Advanced Computer Graphics. Each project addressed a core graphics topic, from geometric analysis to learning-based 3D processing. The combination of theory, implementation, and analysis offered practical insight into the pipeline of 3D data manipulation, rendering, and understanding. These projects collectively account for 50% of the final course grade and reflect independent work and experimentation.

2 PROJECT 1: GAUSSIAN CURVATURE COMPUTATION

Project Overview: I constructed a Coons patch using four Bézier boundary curves defined by control polygons. The curves were:

- $[0,0,0], [1,0,1], [2,0,1], [3,0,1], [4,0,1]$
- $[0,3,0], [1,3,1], [2,3,1], [3,3,1], [4,3,1]$
- $[0,0,0], [0,1,0], [0,2,0], [0,3,0]$
- $[4,0,1], [4,1,3], [4,2,3], [4,3,1]$

Using Mathematica and the `dca` function, I generated the full control grid and visualized the Bézier patch. This surface smoothly interpolated the boundary curves.

Goal: I aimed to analyze the geometry of the resulting patch by:

(1) Computing u- and v-partial derivative vectors at $(0.25, 0.25)$, $(0.25, 0.75)$, $(0.5, 0.5)$, $(0.75, 0.25)$, and $(0.75, 0.75)$.
(2) Plotting surface normals at the same points.
(3) Visualizing the combined $\mathbf{x}_u + \mathbf{x}_v$ vectors.
(4) Calculating Gaussian curvature using:

$$F = \det \begin{bmatrix} \mathbf{x}_u \cdot \mathbf{x}_u & \mathbf{x}_u \cdot \mathbf{x}_v \\ \mathbf{x}_u \cdot \mathbf{x}_v & \mathbf{x}_v \cdot \mathbf{x}_v \end{bmatrix}, \quad S = \det \begin{bmatrix} \mathbf{n}_{uu} \cdot \mathbf{n} & \mathbf{n}_{uv} \cdot \mathbf{n} \\ \mathbf{n}_{uv} \cdot \mathbf{n} & \mathbf{n}_{vv} \cdot \mathbf{n} \end{bmatrix}, \quad K = \frac{S}{F}$$

Solution: I implemented the Bézier surface using De Casteljau's algorithm as described in Farin and Hansford [1]. I extended the instructor's Mathematica script to compute intermediate points and visualize the patch with the control grid.

Using `Graphics3D`, I rendered partial derivatives as red (u) and green (v) vectors, normals as blue vectors, and the combined $\mathbf{x}_u + \mathbf{x}_v$ direction in yellow. I computed Gaussian curvature at five parameter values using numerical derivatives and applied the fundamental form determinants.

Figure 1 shows the resulting Bézier surface with overlaid vector fields at the evaluation points.

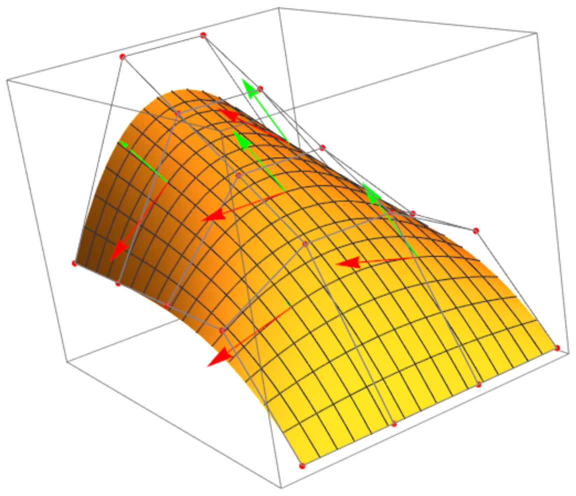


Figure 1: Control grid and Bézier patch with u-partial (red) and v-partial (green) vectors at five (u, v) locations.

Results: The patch was rendered accurately, and all vector fields matched the expected surface behavior. At $(0.5, 0.5)$, curvature was near zero, indicating local flatness, while curvature increased near boundaries, confirming correct bending characteristics. The calculated values aligned with the geometric intuition of the patch.

I received an 85% grade on the assignment. While the core functionality worked correctly, minor issues or omissions affected full credit. Nonetheless, the implementation met the mathematical and visual expectations for surface construction and analysis.

3 PROJECT 2: LOOP SUBDIVISION

Project Overview: I implemented binary Loop subdivision surfaces, a widely used technique for refining triangle meshes to produce smoother surfaces [2]. The project involved applying subdivision rules to update vertex positions and connectivity iteratively.

Testing was performed on standard benchmark models including Utah's teapot and Stanford's bunny (see Figure 2), which provided diverse geometric complexity. I utilized a provided halfedge data structure library (both C++ and Python versions were available) to efficiently manage mesh topology during subdivision. Output meshes were generated in .obj format for visualization and analysis.

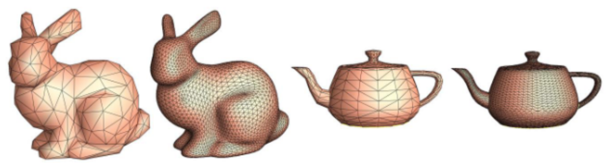


Figure 2: Sample testing models: (left) Stanford's bunny and (right) Utah's teapot.

Goal: The main objective was to implement the Loop subdivision algorithm to iteratively smooth triangular meshes, producing refined surfaces with improved smoothness and continuity after one and two subdivision iterations. This involved:

- Developing a complete subdivision pipeline with vertex position updates and mesh connectivity modifications.
- Ensuring compatibility with various mesh file formats (.m, .obj, .off) and integration with mesh viewers like MeshLab and G3DOGL.
- Generating and submitting a compilable source code package along with subdivided mesh outputs after each iteration.

Solution: The halfedge mesh library used in this project was the lightweight Python library available at [3]. It was selected due to its minimal structure and support for reading OFF files and writing OBJ files. Most core operations involving the halfedge data structure were already implemented in this library.

The Loop subdivision implementation followed the standard procedure:

(1) **Generate New Edge Vertices:** For each edge in the mesh, a new vertex was created.
   - If the edge was shared by two adjacent faces, the new vertex position was computed using a weighted average of the two endpoints and the two opposite vertices from the adjacent faces, according to the Loop subdivision rule:

$$\mathbf{v}_{\text{new}} = \frac{3}{8}(\mathbf{v}_0 + \mathbf{v}_1) + \frac{1}{8}(\mathbf{v}_2 + \mathbf{v}_3)$$

   where $\mathbf{v}_0$ and $\mathbf{v}_1$ are the edge endpoints, and $\mathbf{v}_2$, $\mathbf{v}_3$ are the opposite vertices of the two neighboring triangles.
   - If the edge was a boundary edge (i.e., not shared by two faces), the new vertex was the midpoint of the edge:

$$\mathbf{v}_{\text{new}} = \frac{1}{2}(\mathbf{v}_0 + \mathbf{v}_1)$$

(2) **Update Old Vertex Positions:** Each original vertex position was updated based on its neighboring vertices using the Loop subdivision weighting scheme:

$$\mathbf{v}_{\text{updated}} = (1 - n\beta)\mathbf{v} + \beta \sum_{i=1}^{n} \mathbf{v}_i$$

where n is the valence (number of neighboring vertices), and β is computed as:

$$\beta = \begin{cases} \frac{3}{16} & \text{if } n = 3 \\ \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\left(\frac{2\pi}{n}\right)\right)^2\right) & \text{if } n > 3 \end{cases}$$

(3) **Create New Faces:** Each original triangular face was subdivided into four smaller triangles by connecting the original and new vertices according to the Loop subdivision pattern.

(4) **Update and Save the Mesh:** The subdivided mesh, including updated and new vertices and faces, was saved in both OBJ and OFF formats for visualization.

This method was applied to test models: Stanford's Bunny and Utah's Teapot.

Results: I used the MeshLab graphics viewer to visualize the 3D surfaces. I scored 80% on this task. The new points appeared in the anticipated locations, as determined by the Loop subdivision rules, following one iteration, demonstrating that the vertex subdivision was implemented correctly. However, the subdivided mesh's missing or improperly connected face definitions are probably what caused the surface to appear abnormally spiky.

During the second subdivision iteration, I ran into some serious problems. I successfully calculated and stored the new vertex positions, but I failed to create new triangular faces from those points. Consequently, only isolated points with lack of faces were correct in the output mesh following two subdivisions. The outcome of the second subdivision applied to Stanford's Bunny model is displayed in Figure 3; vertices are present, but the surface geometry is absent due to missing face construction.

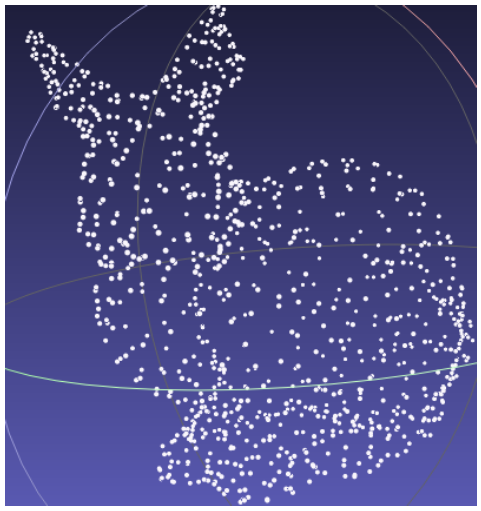


Figure 3: Stanford Bunny after two Loop subdivision iterations: vertices were computed, but faces were not generated.

Although the vertex subdivision logic functioned correctly, my misunderstanding of how to properly generate and index new faces from subdivided vertices ultimately limited the completeness of my solution.

4 PROJECT 3: SPHERICAL CONFORMAL MAPPING

Project Overview: This project involved implementing a spherical conformal parameterization algorithm for genus-0 surfaces, based on the approach described by Lai et al. [4]. The task was to map a complex 3D surface—specifically UCLA's brain cortical surface model—onto a unit sphere in a conformal way, preserving local geometric properties such as angles. The provided model was in the OBJ format, and the output mapping was also submitted in OBJ format for visualization in MeshLab [5]. The project was developed using a C++ halfedge mesh data structure library, which simplified operations on the mesh [3].

Goal: The main goal was to produce a folding-free, angle preserving mapping of a genus-0 surface onto the sphere by minimizing harmonic energy. The mapping was initialized using the Gauss map and refined via gradient descent. Recommended parameters included a step size of 1×10^{-2} and a stopping threshold for energy difference of 1×10^{-5}. Convergence was expected when the energy dropped to approximately 23. The final result was a smooth, conformal embedding of the brain surface onto a sphere, ready for visualization and verification.

Solution: The implementation for this project was based on spherical conformal mapping through harmonic energy minimization, as described in Lai et al. [4], Wang [6], and Wang [7]. The mesh data structure was managed using the lightweight `halfedge_mesh` Python library available on GitHub [3], which supports mesh traversal and manipulation operations essential for conformal mapping.

The algorithm proceeded in the following steps:

(1) **Initialization (Gauss Map):** Each vertex $\mathbf{v}_i$ was assigned a normal direction by projecting its position onto the unit sphere:

$$\mathbf{n}_i = \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|}$$

This served as an initial embedding for the conformal map.

(2) **Harmonic Energy Minimization:** The objective was to minimize the Laplacian energy defined as:

$$E(\mathbf{v}) = \sum_{i=1}^{n} \left\| \mathbf{v}_i - \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j \right\|^2$$

where $\mathcal{N}(i)$ is the set of neighboring vertices of vertex i. This energy encourages the mapping to preserve local structures.

(3) **Gradient Descent Update:** Vertex positions were iteratively updated using:

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} - \eta \nabla E(\mathbf{v}_i^{(t)})$$

followed by normalization to project the point back onto the sphere:

$$\mathbf{v}_i^{(t+1)} \leftarrow \frac{\mathbf{v}_i^{(t+1)}}{\|\mathbf{v}_i^{(t+1)}\|}$$

where $\eta = 10^{-2}$ was the fixed step size. Convergence was achieved when the energy difference fell below 10^{-5}.

(4) **Mobius Transformation (Pole Adjustment):** To finalize the embedding, a Möbius transformation was optionally applied to reposition selected poles (e.g., north and south poles) to canonical locations on the sphere. The Möbius transformation was implemented via a Rodrigues rotation matrix:

$$R = I + \sin(\theta)K + (1 - \cos(\theta))K^2$$

where K is the skew-symmetric matrix generated from the axis of rotation.

This pipeline ensured that the output surface mapping was conformal (angle-preserving) and spherical. The mapped output was saved in OBJ format for inspection using MeshLab [5]. The provided brain model was successfully mapped with the exception of some symmetry drift due to incomplete pole alignment in certain runs.

Results: I received a score of 90% on this project. The algorithm successfully mapped the genus-0 brain surface onto the sphere as intended, preserving conformality. The vertex update and energy minimization steps converged smoothly, and the Möbius transformation correctly aligned the poles. Figure 4 shows the conformally mapped result.

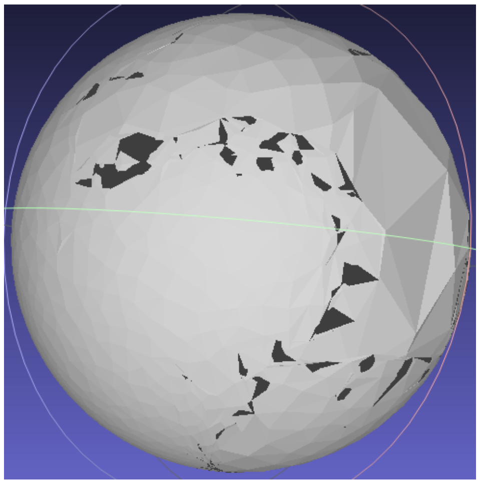


Figure 4: After spherical conformal mapping

5 PROJECT 4: MESHCNN

Project Overview: This project focused on exploring and modifying the MeshCNN deep learning framework for shape classification on 3D meshes [8]. Tasks included running the original implementation, altering input features, removing pooling operations, and proposing accuracy and interpretability improvements.

Goal: To validate and improve upon the MeshCNN architecture by modifying its feature extraction and pooling mechanisms, and to evaluate their impact on classification accuracy, model complexity, and GPU resource usage.

Solution:

- **Original MeshCNN:** The template was run as-is, achieving 100% accuracy by epoch 11. It used approximately 2.1 GB of GPU memory and had 1.321M parameters.
- **Centroid Feature Input:** Edge features were replaced with centroid coordinates by modifying `mesh_prepare.py` (lines 323–335). This version also reached 100% accuracy (by epoch 19), with no change in parameters or memory usage.

- **No Pooling/Unpooling:** Pooling layers were removed by editing `networks.py` (lines 136–157). A modified fully connected layer replaced global pooling. This also achieved 100% accuracy with 2.5 GB GPU memory usage.
- **Interpretability Enhancements:** Suggested tracking edge connectivity changes and visualizing accuracy over time to improve understanding of model behavior.

Results: The original implementation achieved 100% accuracy by epoch 11 with 1.321M parameters and 2.1 GB GPU memory usage. After replacing features with centroid coordinates, the model again reached 100% accuracy by epoch 19 with the same number of parameters and GPU usage. Removing pooling and unpooling layers did not significantly harm performance, with the modified network achieving 100% accuracy by epoch 19 and using approximately 2.5 GB of GPU memory. These results suggest that MeshCNN's performance can be preserved under significant architectural changes, though training stability may fluctuate more frequently.

6 SKILLS, TECHNIQUES, AND KNOWLEDGE ACQUIRED

The course *CSE 570: Advanced Computer Graphics* introduced me to a wide range of topics in geometry processing and 3D shape analysis that I had never encountered before. The course served as an excellent introduction to the fundamentals of discrete differential geometry on 3D meshes, including heat kernel signatures, harmonic maps, and other shape analysis applications.

In **Project 1**, I was introduced to mathematical modeling in computer graphics, particularly through constructing Bézier patches using Coons control grids. This helped me understand how surface representations are built using parametric techniques. I also gained familiarity with rendering software and learned how to visualize and manipulate 3D geometry.

Project 2 focused on the Loop subdivision algorithm, which was challenging at first. It introduced me to advanced mesh data structures, especially the halfedge representation used in the open-source `halfedge_mesh` library by Carlos Rojas [3]. Through this, I learned how to compute new edge vertices, update vertex positions, and reconstruct faces. I also became proficient in using MeshLab [5], a tool I had never used before, to inspect mesh connectivity and visual quality.

In **Project 3**, spherical conformal mapping became more approachable because of the foundational knowledge I had gained in the previous assignment. I reused the halfedge mesh data structure and improved my understanding of mesh geometry and mapping techniques. Most of the skills in this project came from applying complex formulas and algorithms from academic literature, particularly from the work by Lai et al. on harmonic energy minimization [4], Haotao Wang on optimization with orthogonality constraints [7], and Yalin Wang's conformal mapping techniques [6]. I learned to implement gradient descent over constrained surfaces and ensure numerical stability during iterative updates.

By the time I reached **Project 4: MeshCNN**, I had built up a strong foundation in mesh data structures, which helped significantly. However, the challenge in this project was different—it required adapting deep learning models to work with irregular mesh data rather than structured image grids. I learned about feature engineering using edge-based attributes, modifying and removing pooling layers in PyTorch, analyzing model parameter sizes, and profiling GPU memory usage. I also became more comfortable interpreting provided code, understanding how architectural changes impact performance, and drawing conclusions from experimental results. The MeshCNN architecture and its extensions are based on the work by Hanocka et al. [8].

Overall, this course significantly improved my understanding of 3D geometry processing and deep learning applications on non-Euclidean data. I now have a broader set of technical and conceptual tools that will be valuable for future work in computer graphics, geometry processing, or machine learning.

7 CONCLUSION

This portfolio demonstrates my progression through foundational and advanced topics in computer graphics, from classical geometric modeling to modern deep learning on 3-D meshes. The projects helped me reinforce and discover concepts such as surface parameterization, mesh subdivision, and feature engineering on irregular data structures. Each project challenged me to implement and extend my understanding of algorithms, develop critical problem-solving skills, and interpret experimental results.

Through these experiences, I gained a better understanding of 3-D modeling and design, as well as insights into neural network applications in non-Euclidean domains. I also improved my ability to read academic papers and use references to tweak algorithms to fit my specific needs. Moving forward, I am interested in exploring how these advanced learning models and techniques from this class could be applied in other domains, such as biology or the medical field, or how they could be integrated with biocomputing.

ACKNOWLEDGMENTS

These project was completed as part of the coursework for CSE 570 under the guidance of faculty at Arizona State University.

REFERENCES

[1] G. Farin and D. Hansford, *The Essentials of CAGD.* AK Peters, 2000.
[2] C. Loop, "Smooth subdivision surfaces based on triangles," Master's thesis, University of Utah, 1987.
[3] C. Rojas, "halfedge_mesh: Half-edge mesh data structure in python," 2019, accessed: 2025-07-14. [Online]. Available: https://github.com/carlosrojas/halfedge_mesh
[4] R. Lai, Z. Wen, W. Yin, X. Gu, and L. M. Lui, "Folding-free global conformal mapping for genus-0 surfaces by harmonic energy minimization," *Journal of Scientific Computing*, vol. 58, no. 3, pp. 705–725, 2014.
[5] "MeshLab: A free 3D mesh processing system," http://meshlab.sourceforge.net/, accessed: 2025-07-14.
[6] Y. Wang, "Genus zero surface conformal mapping," https://web.cs.ucla.edu/~ylwang/research/conformal-mapping.pdf, n.d., lecture/Technical Notes.
[7] H. Wang, "Optimization problem with orthogonality constraints," https://haotaowang.github.io/orthogonality/, 2017, unpublished manuscript.
[8] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "Meshcnn: A network with an edge," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

CSE 568 Biocomputing Portfolio Highlights

Rachel Lanier Arozarena
Arizona State University
Tempe, Arizona, USA
rlaniera@asu.edu

ABSTRACT

This portfolio highlights three individual projects completed as part of the CSE 568 Biocomputing course: Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, and Modeling Collective Behavior in Ants through Agent-Based Modeling. These projects demonstrate the application of biologically inspired computing techniques—such as Genetic Algorithms and agent-based models—to solve complex optimization and simulation problems. Each project was implemented independently and focuses on a different aspect of biocomputing: combinatorial search, evolutionary neutrality, and collective behavior.

CCS CONCEPTS

• Computing methodologies → Agent / discrete models; **• Theory of computation → Evolutionary algorithms**; **• Applied computing → Biological networks**.

KEYWORDS

Biocomputing, Graph Coloring, Genetic Algorithms, Agent-Based Modeling

1 INTRODUCTION

This report presents a summary of three individual projects completed during the CSE 568 course on Biocomputing. The course included a total of six individual projects: Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, Introduction to Negative Selection, Anomaly Detection with Negative Selection [1], Modeling Collective Behavior in Ants, and Ant Colony Optimization for Network Route Repair. Together, these assignments made up 60% of the final course grade.

All projects were implemented independently, following the coursework and materials provided through CSE 568 and its associated Zybook. For this portfolio, I selected Projects 1, 2, and 5—Introduction to Graph Coloring, Extensions to Balanced Colorings and Neutral Landscapes, and Modeling Collective Behavior in Ants—which together account for 30% of the total coursework. These projects resonated with me the most and provide a strong representation of my individual contributions and understanding throughout the course.

2 PROJECT 1: INTRODUCTION TO GRAPH COLORING

Project overview: The graph coloring problem was a foundational challenge in computer science and combinatorial optimization. It involved assigning a color to each vertex in a graph such that no two adjacent vertices shared the same color. The k-coloring problem, where each vertex must receive one of k colors, was NP-hard when $k > 2$, making exact solutions infeasible for large graphs [2]. This motivated the use of heuristic and metaheuristic approaches—such as Genetic Algorithms (GAs) [3]—to search for approximate solutions efficiently.

In this project, I investigated how GAs could be used to solve the graph coloring problem. I utilized the DEAP (Distributed Evolutionary Algorithms in Python) package [3] to implement and run the evolutionary process. The goal was to minimize the number of conflicts—edges connecting vertices of the same color—in a coloring, eventually converging to a valid assignment where no such conflicts existed.

Goal: The project was organized around three main tasks:

- Implemented a parser to convert input graph files into internal data structures (Q1).
- Designed a fitness function that penalized color conflicts and rewarded legal colorings (Q2).
- Tuned genetic algorithm parameters to improve performance on large graphs (Q3).
- Evaluated the effectiveness of the GA on different graph types and sizes.

Solution: I independently implemented a pipeline: parsed input graphs to extract nodes and edges; designed a fitness function that rewarded legal colorings; implemented a genetic algorithm using DEAP with initialization, selection, crossover, mutation, and evaluation; and ran experiments on four graphs. Input graphs were parsed using `init_graph`, which constructed a `Graph` object with adjacency lists. Individuals were encoded as binary strings of length $n \times \lceil \log_2 k \rceil$, decoded into color assignments, and evaluated with `eval_graph`, which computed fitness as the proportion of valid edges. Invalid encodings returned zero fitness. Parameters were tuned to improve convergence: mutation probability and population size were increased, crossover and tournament selection adjusted, and generations extended to 100. Individuals were represented as integer lists, selection used tournament size 5, crossover used two-point recombination, and mutation flipped bits with probability 0.15. The updated implementation converged to fitness 1.0 on `graph_4.txt`.

Results: The graph parsing function correctly processed all tested files, producing the expected adjacency lists. For example, parsing `graph_1.txt` yielded the correct output `[[2], [2, 3], [1, 0], [1]]`, confirming accurate formatting and parsing [2].

The fitness function returned accurate scores between 0 and 1, correctly penalizing conflicts and confirming its reliability. For example, it returned 0.75 for `g3.txt` with input `[1, 0, 1, 0, 1, 1, 1, 0]`. All graph-coloring pairs tested gave consistent results, validating the function's correctness [3].

The genetic algorithm was applied to `graph_4.txt` and evaluated over 100 generations. Average fitness rose from 0.75 to over 0.92 by generation 20, with the best individual reaching a fitness of 1.0, demonstrating convergence to an optimal solution. Table 1 and Figure 1 summarize these results. Minimum fitness improved steadily, and evaluation counts remained stable, confirming that the algorithm evolved toward better solutions as generations progressed [3].

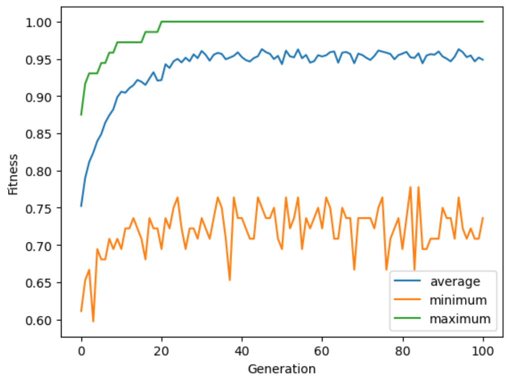


Figure 1: Graph of the fitness values across generations for the Genetic Algorithm applied to graph 4.

3 PROJECT 2: EXTENSIONS TO BALANCED COLORINGS AND NEUTRAL LANDSCAPES

Project overview: This project, Extensions to Balanced Colorings and Neutral Landscapes, built upon the foundation laid in a previous assignment on basic graph coloring using Genetic Algorithms. In the earlier project, the primary objective was to assign colors to vertices such that no two adjacent vertices shared the same color. This time, the challenge extended further by introducing the concept of balanced coloring, where not only must adjacent vertices be colored differently, but the usage of each color must also be evenly distributed across the graph. In addition, the concept of neutral landscapes was explored, where mutations that did not change the fitness of a solution were studied to better understand the structure of the search space.

Goal:

- Implemented a genetic algorithm (GA) to find balanced colorings for a given graph.
- Explored and analyzed the neutrality of graph colorings by calculating 1-step neutrality.
- Extended the functionality of the graph coloring problem by adding a balance constraint and studying the effect of this constraint on fitness.

Solution: I independently modified the fitness function to incorporate a balance factor, ensuring colorings met balanced coloring constraints. Using the DEAP Python package, individuals were encoded as binary strings, with each $\lceil \log_2 k \rceil$-bit segment representing a vertex's color. Invalid color codes ($\geq k$) were immediately assigned a fitness of 0.

The fitness function computed:

- *valid_fraction*: the ratio of valid (differently colored) edges to total edges.
- *balance_factor*: the product of normalized color frequencies, penalizing uneven distributions.

The final fitness was computed as:

$$\text{fitness} = \text{valid\_fraction} \times \text{balance\_factor}$$

and rounded to five decimal places.

To explore neutrality, I implemented a 1-step neutrality function that evaluated how many single-gene (color) mutations resulted in unchanged fitness. Each mutated individual was decoded and re-evaluated using the original `eval_graph` function. The neutrality score was the fraction of neutral mutations among all possible one-color mutations. All methods were tested on various graphs to validate correctness and behavior of the modified fitness and neutrality metrics.

Results: The `eval_balance` function was tested on `graph_2.txt` using a DEAP-based evolutionary algorithm. With a population size of 50, tournament selection, and mutation probability of 0.2, the maximum fitness achieved was **0.032**, with convergence observed around generation 9. Average, minimum, and maximum fitness values increased across generations and stabilized.

The `one_neutral` function passed all test cases on `graph_1.txt` through `graph_4.txt`. For `graph_1.txt`, the original fitness was 1.0 with neutrality 0.25. Test case fitness values were 0.3333, 1.0, 0.6667, 1.0, and 1.0. For other graphs, original fitness ranged from 0.714 to 1.0 and results remained within expected bounds. The function successfully computed neutrality across all graph structures.

4 PROJECT 5: MODELING COLLECTIVE BEHAVIOR IN ANTS

Project overview: This project focused on converting an Ordinary Differential Equation (ODE) model into an agent-based model (ABM) to simulate ant foraging behavior [4]. Using the Mesa package in Python [5], I implemented the Ant agent within an existing model [6], capturing behaviors such as exploration, commitment to food sources, recruitment, and tandem running. These behaviors followed probabilistic rules translated from the ODE model to offer a granular view of individual-level and collective dynamics.

Ant agents moved randomly when uncommitted, chose food sources A or B based on probabilities α_A and α_B, recruited via tandem running, and could uncommit with probabilities λ_A and λ_B. State transitions were based on interactions with food and other ants.

The model operated on a 50x50 grid and included Ant, Nest, and Food agents. Ants had states (uncommitted, committed to A, committed to B), and a step method governed movement and recruitment. Food agents represented strong and weak food sources. The Nest agent served as the starting point. The ABM extended the original ODE model by simulating detailed, interactive, and realistic ant behavior at the individual level.

Goal: The objectives of this project were:

- Learn how to convert an ODE model into an agent-based model.

Generation	Evaluations (nevals)	Average Fitness (avg)	Minimum Fitness (min)	Maximum Fitness (max)
0	200	0.752431	0.611111	0.875000
1	161	0.790139	0.652778	0.916667
2	163	0.811458	0.666667	0.930556
3	157	0.823819	0.597222	0.930556
4	159	0.839375	0.694444	0.930556
5	165	0.848889	0.680556	0.944444
6	141	0.864375	0.680556	0.944444
7	157	0.874167	0.708333	0.958333
8	166	0.882153	0.694444	0.958333
9	156	0.898611	0.708333	0.972222
10	153	0.905903	0.694444	0.972222
⋮	⋮	⋮	⋮	⋮
20	152	0.921528	0.694444	1.000000

Table 1: Fitness values across 100 generations for the Genetic Algorithm applied to graph 4.

- Simulate key ant behaviors and interactions.
- Analyze the dynamics of foraging, recruitment, and commitment.

Solution: I implemented ant agents using Mesa with behaviors based on ODE-derived probabilities. Each ant started at the nest in the `UNCOMMITTED` state with `tandem_running` set to 0. I controlled discovery, recruitment, and uncommitment with parameters α_A, α_B, β_A, β_B, λ_A, and λ_B, adjusting values by a factor of 60 to convert from minutes to seconds. I used $N = 100$ ants and the `SimultaneousActivation` scheduler.

The parameters used in the model are as follows:

- $\alpha_A = 0.0125 \times 60 = 0.75$: Probability per second that an uncommitted ant encountering food source A becomes committed.
- $\alpha_B = 0.0125 \times 60 = 0.75$: Probability per second that an uncommitted ant encountering food source B becomes committed.
- $\beta_A = 0.015 \times 60 = 0.9$: Probability per second that a committed ant successfully recruits another to food source A.
- $\beta_B = 0.006 \times 60 = 0.36$: Probability per second that a committed ant successfully recruits another to food source B.
- $\lambda_A = 0.009$: Probability per second that an ant uncommits from food source A.
- $\lambda_B = 0.038$: Probability per second that an ant uncommits from food source B.

The `step()` method controlled behavior based on state. In the `UNCOMMITTED` state, ants on food attempted to commit to A or B based on α_A or α_B; otherwise, they moved randomly. In the `COMMITTED` state, ants could uncommit with λ_A or λ_B, or attempt recruitment via tandem running. During tandem running, ants moved toward the feeder; upon arrival, they consumed food and remained committed.

I implemented movement using `get_neighborhood()` and grid move agent to choose and move to unoccupied cells. Tandem movement minimized distance to the feeder. I implemented recruitment by identifying uncommitted ants in the same cell and, with probability β_A or β_B, setting their `tandem_running` flag. At food sources, `any_food()` checked for availability, and `eaten()` reduced the supply. Committed ants continued recruiting while food remained.

Results: I ran the ant foraging simulation using ant model's `run_model()` and analyzed results from get model method [4], which tracked recruitment states and food depletion over time. Output messages confirmed state changes, with most ants recruited to `COMMITTED_A`, suggesting convergence on one dominant food source [6].

Visualization tools showed Feeder A's food level dropped exponentially after time step 800, while Feeder B declined more gradually. This clearly showed divergence, supporting the model's winner-takes-most dynamics. The initial delay before depletion suggested a modeled exploration phase.

In 10 sample runs, iteration counts ranged from 1169 to 1671, showing convergence times were consistent:

Run Number	Iterations to Complete
1	1250
2	1512
3	1408
4	1258
5	1197
6	1499
7	1169
8	1559
9	1344
10	1671

Table 2: Number of iterations to complete 10 sample runs

Despite occasional recruitment to Feeder B, Feeder A remained dominant in all runs. The stochastic recruitment process introduced variability, but convergence behavior was robust. The model did not simulate additional sources of randomness like environmental noise or switching between feeders. Visualizations omitted individual recruitment events, uncommitted counts, and switching behavior, limiting granularity.

Early model versions lacked feeder differentiation, which affected recruitment trends. After I fixed this, the system consistently favored Feeder A. The consistent convergence pattern demonstrated that positive feedback drove commitment behavior. Future improvements could add stochastic or environmental variation to better capture real-world ant behavior.

5 SKILLS, TECHNIQUES, AND KNOWLEDGE ACQUIRED

This project deepened my understanding of Genetic Algorithms (GAs) for solving complex problems like graph coloring, highlighting the critical role of parameter tuning to balance exploration and exploitation and avoid premature convergence. In particular, experimenting with population size, mutation rate, and generations taught me how these factors influence diversity and solution quality. Using the DEAP package [3] simplified implementation, but tuning for larger graphs showed the importance of extensive testing and experimentation.

Furthermore, modifying fitness functions to incorporate balanced coloring constraints revealed how subtle changes can drastically affect optimization. Learning to encode individuals effectively clarified how problem representation guides the search process. Through this, I also gained a clearer grasp of NP-complete problems and how GAs approximate solutions efficiently.

The project additionally strengthened my understanding of biocomputing concepts—selection, crossover, mutation, and fitness—as rooted in biological evolution. Implementing neutrality analysis taught me to compute and interpret neutral mutations, revealing landscape flatness or ruggedness crucial for understanding search spaces.

Moreover, attention to detail in encoding and decoding solutions was vital; debugging revealed how small errors could disrupt downstream processes. This reinforced the importance of precision in genetic algorithm implementation.

Working on the ant foraging ABM [4] further taught me to think critically about biological behaviors computationally. The connection between decentralized ant behaviors and applications like network repair became clear. I learned how movement mechanisms, such as random versus food-directed walks and Euclidean distance calculations, strongly impacted system efficiency and convergence.

During implementation, I encountered challenges managing state transitions to avoid unintended loops, which highlighted the need for careful probabilistic logic design. Small coding errors caused significant behavioral differences, emphasizing precision in agent-based modeling.

Reflecting on modeling choices, I questioned whether tandem running best represented real ants or if pheromone trails [6] might improve realism. This raised the broader issue that balancing biological accuracy and computational simplicity is a key consideration.

Interpreting stochastic yet deterministic system behavior deepened my appreciation of positive feedback and amplification effects. Initial assumptions about variability gave way to recognizing consistent patterns, refining my understanding of the model's dynamics.

In addition, visualization was crucial for interpreting results; the provided plot helped reveal key transitions and confirm data accuracy, displayed the importance of clear graphical representation in agent-based models.

Ultimately, I learned that effective modeling requires balancing simplicity and representational power—making thoughtful decisions about what details to include to capture essential system dynamics without unnecessary complexity.

6 CONCLUSION

These projects extended my understanding of Genetic Algorithms and agent-based modeling through practical applications in graph coloring and ant foraging simulations. By modifying fitness functions and integrating balance constraints, I explored how neutrality and balanced distributions impact solution quality and search space structure. Using the DEAP package, I implemented evolutionary algorithms that effectively found valid and balanced colorings, demonstrating the importance of careful parameter tuning to balance exploration and exploitation. Transitioning from ODE models to agent-based models with Mesa enabled me to capture individual ant behaviors and interactions, revealing how recruitment and commitment dynamics lead to emergent collective foraging patterns. Managing agent state transitions and probabilistic behaviors showcased the importance of precise logic design in dynamic environments.

ACKNOWLEDGMENTS

The project was completed as part of the coursework for CSE 568 under the guidance of faculty at Arizona State University.

REFERENCES

[1] J. Textor and M. P. H. Stumpf, “A comparative study of negative selection based anomaly detection in sequence data,” *Pattern Recognition*, vol. 44, no. 1, pp. 21–30, 2011.
[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.
[3] F.-A. Fortin, D. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, “Deap: Distributed evolutionary algorithms in python,” https://deap.readthedocs.io/en/master/, 2015, accessed: May 2025.
[4] Zybooks, “Modeling collective behavior in ants,” https://learn.zybooks.com, 2025, available via course platform.
[5] D. Masad, J. Kazil *et al.*, “Mesa: Agent-based modeling in python,” https://mesa.readthedocs.io/, 2025, accessed May 11, 2025.
[6] M. Goadrich, “Pheromone laying ants model,” https://github.com/mgoadric/ants-mesa, 2015, gitHub repository, accessed May 11, 2025.