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.

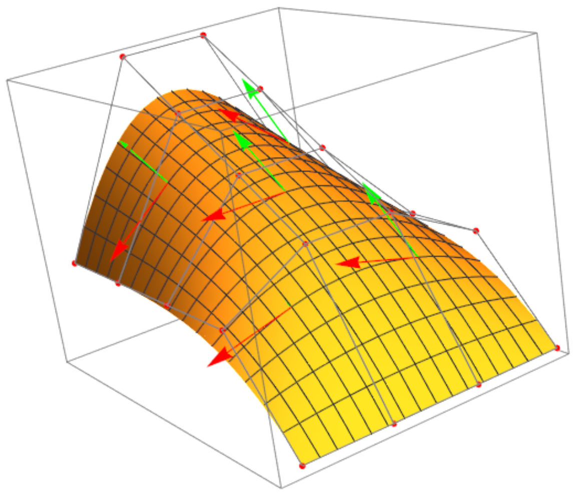

Figure 1: Control grid and Bézier patch with u-partial (red) and v-partial (green) vectors at five (u, v) locations.

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.

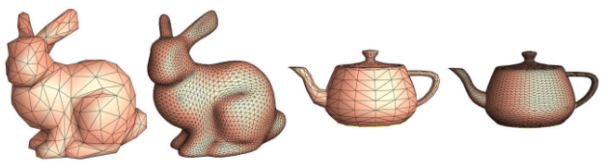

Figure 2: Sample testing models: (left) Stanford's bunny and (right) Utah's teapot.

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.

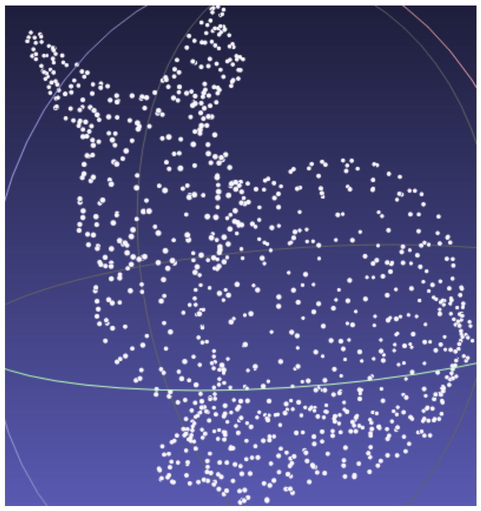

Figure 3: Stanford Bunny after two Loop subdivision iterations: vertices were computed, but faces were not generated.

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.

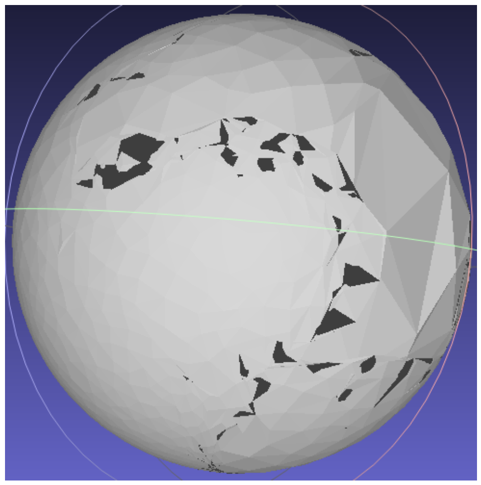
Figure 4: After spherical conformal mapping

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.