Final Project Report: Replicating and Enhancing Orthogonalized Fourier Polynomials for Efficient Signal Transfer (Computer Graphics, COMP4610/8610-2024)

Aditya Iyengar
u7670692

Omair Soomro
u7517790

Jinshuo Zhang
u7723176

The Australian National University

Abstract

This project revisits and extends the methodologies presented in "Orthogonalized Fourier Polynomials for Signal Approximation and Transfer" by Maggioli et al. (2021). The original research proposes using pointwise polynomial combinations of Laplacian eigenfunctions to achieve compact yet expressive representations of signals defined on 3D shapes. Our project replicates and optimizes this approach in Python, addressing specific computational limitations encountered in the MATLAB implementation. By constructing a robust orthonormal basis derived from these eigenproducts, our work improves upon the stability and computational efficiency of the original method. We particularly focus on two critical areas of computer graphics: spectral geometry processing and shape analysis. Our Python implementation demonstrates significant enhancements in numerical stability, representation accuracy, and resilience to discretization artifacts. Experiments involving signal reconstruction and transfer tasks between near-isometric and non-isometric shapes validate these improvements, highlighting the practical value and adaptability of our optimized method. This project also outlines potential future refinements, including extending the basis construction to higher-order products and further exploring applications in complex geometric and signal processing scenarios.

1 Introduction

In this project, we revisit the methodology proposed by Maggioli and colleagues, aiming to replicate their results and address existing computational constraints through a Python-based implementation. By constructing a more robust orthonormal basis derived from eigenproducts, we significantly enhance both numerical stability and computational efficiency. Our approach specifically focuses on critical applications within spectral geometry processing and shape analysis, aiming to attempt improvements in the accuracy of signal representation and robustness to discretization artifacts. Through rigorous experimentation involving both signal reconstruction and transfer tasks on various near-isometric and non-isometric shapes, we validate the effectiveness and practical utility of our replicated implementation. Furthermore, we discuss potential extensions and future improvements, such as higher-order polynomial basis constructions, setting the stage for further advancements in geometric and signal processing domains.

2 Problem Statement

The accurate approximation and efficient transfer of signals across 3D shapes have long been central challenges in computer graphics and geometry processing. These challenges are compounded by the need for high fidelity in capturing intrinsic geometric properties while maintaining computational efficiency and robustness to noise and discretization errors. Signal reconstruction and transfer applications, including shape analysis, compression, and correspondence, rely heavily on the ability to represent and manipulate functions defined over complex, high-dimensional manifolds.

Recent advances, such as the work by Maggioli et al. (2021), have introduced orthogonalized Fourier polynomials derived from Laplacian eigenfunctions as a promising solution for these tasks. This approach leverages polynomial combinations of eigenfunctions to construct a compact and expressive basis, offering improved signal representation and facilitating transfer between shapes. However, the original MATLAB implementation of this methodology is not without limitations. Key challenges include significant computational overhead, poor memory scalability when dealing with large meshes or high-order basis functions, and numerical instability in the orthogonalization of polynomial products. These issues hinder the practical deployment of the method in large-scale or real-time applications, limiting its utility in complex geometry processing tasks.

Furthermore, the use of polynomial combinations of eigenfunctions, while theoretically appealing, introduces additional complexity in basis construction and requires precise handling of matrix computations, including mass and stiffness matrices, spectral decompositions, and Gram-Schmidt orthogonalization. The original MATLAB code's reliance on loop-based constructs exacerbates performance issues, particularly when scaling to higher-order polynomial bases or dense meshes. Additionally, MATLAB's static and memory-intensive design impedes efficient exploration of reconstruction and transfer performance across a wide range of parameter configurations and input geometries.

In this context, a Python-based reimplementation and optimization of the orthogonalized Fourier polynomials framework is highly motivated. Python's extensive ecosystem of scientific libraries, including NumPy, SciPy, PyVista, and Matplotlib, provides the tools necessary for efficient numerical computation, interactive visualization, and scalable data handling. By porting and optimizing the original methodology into Python, we aim to overcome the limitations of the MATLAB approach, achieving improved computational efficiency, numerical stability, and interactivity.

Specifically, our project addresses the following critical problems:

- **Computational Efficiency:** Reducing runtime and memory overhead for high-order basis construction and orthogonalization, enabling scalability to large meshes and higher-dimensional polynomial expansions.

- **Numerical Stability:** Enhancing the robustness of the Gram-Schmidt orthonormalization process to mitigate the accumulation of numerical errors and improve the reliability of the generated basis.

- **Interactivity and Visualization:** We attempted to replicate the dynamic nature of Meshlab by providing real-time rendering and dynamic interaction with 3D models and spectral data, allowing for intuitive exploration of reconstruction accuracy and spectral properties.

- **Scalability and Extensibility:** Designing a modular and extensible architecture that supports integration of advanced features such as adaptive basis selection, GPU acceleration, and spectral filtering.

- **Reproducibility and Validation:** Establishing a clear, transparent, and reproducible framework for validating spectral geometry methods, including quantitative metrics such as Dirichlet energy and visual assessments of reconstruction and transfer fidelity.

By addressing these challenges, our project not only replicates the original methodology with improved performance but also lays the groundwork for future innovations in spectral geometry processing, shape analysis, and related domains. The resulting framework enables both theoretical exploration and practical deployment of advanced signal processing techniques on complex geometric structures.

3 Technical Implementation Details

Our implementation leverages Python, a versatile and efficient programming language known for its extensive scientific computing capabilities. This environment provides a streamlined and scalable alternative to MATLAB, particularly for tasks involving high-order finite element methods (FEM), spectral geometry, and large-scale numerical computations. Central to our framework is the custom module `common_functions.py`, which encapsulates a range of optimized algorithms critical for spectral processing on complex 3D shapes. This module includes functionalities for high-order FEM construction, symbolic computation of basis functions, efficient computation of adjacency and boundary matrices, and streaming-safe Gram-Schmidt orthonormalization. These implementations ensure accurate and stable computation of Laplace-Beltrami eigenbases, polynomial eigenproducts, and associated matrix operations even on large and high-resolution meshes.

A key extension of our system is the incorporation of spectral filtering techniques for shape reconstruction and signal transfer. By leveraging the eigenproduct construction capabilities of `common_functions.py`, we generate polynomial combinations of Laplacian eigenfunctions that form a rich and expressive basis. Filters are applied in the spectral domain using exponential decay functions, effectively attenuating high-frequency components and enhancing the smoothness and fidelity of reconstructed signals. This process is further stabilized through the use of orthogonalized bases, computed with mass-matrix-weighted Gram-Schmidt orthonormalization routines that incorporate adjustable thresholding for robust convergence. The system's capacity for precise Dirichlet energy computation, also handled within `common_functions.py`, enables quantitative validation of reconstruction accuracy and smoothness, providing a strong foundation for adaptive filtering and reconstruction quality assessment.

Interactive and high-fidelity visualization is achieved using PyVista, offering real-time rendering of complex 3D models, scalar fields, and spectral plots. Users can interactively explore meshes with smooth shading, interpolated color mapping, and dynamic camera controls, surpassing the static visualization capabilities of MATLAB. The framework supports side-by-side comparison of source and reconstructed meshes under various filtering and orthonormalization configurations, enabling clear and intuitive evaluation of reconstruction quality and spectral filtering effects.

Numerical operations, including sparse matrix assembly, eigenvalue decomposition, and matrix multiplications, are efficiently performed using NumPy and SciPy, ensuring stability and scalability even for high-order polynomial expansions and dense mesh representations. Symbolic computation capabilities, powered by SymPy, enable accurate and automated evaluation of integrals required for constructing higher-order FEM basis functions, further improving the efficiency and correctness of the implementation. The use of vectorized and streaming-safe algorithms throughout the system ensures low memory overhead and high performance, while also supporting future extensions such as GPU acceleration, adaptive basis selection, and automated parameter optimization.

Overall, this implementation integrates high-order FEM, advanced spectral filtering, and interactive visualization into a unified and scalable framework. It provides a robust platform for accurate signal reconstruction and transfer across complex geometric structures, laying the groundwork for further innovations in spectral geometry processing, signal analysis, and shape matching.

4 Project Implementation and Contributions to Computer Graphics Domains

Our project replicates and significantly extends the methodology outlined in *Orthogonalized Fourier Polynomials for Signal Approximation and Transfer* (Maggioli et al., 2021), transforming core concepts into an optimized, scalable, and interactive Python framework. The adaptation from MATLAB to Python demanded rigorous redesign of algorithms, data structures, and rendering pipelines, ensuring not only functional equivalence but also improved performance, usability, and modularity. This section highlights both the **key accomplishments** and the **specific technical features** implemented, with clear connections to multiple areas of computer graphics.

4.1 Core Accomplishments and Implemented Features

1. Geometry Processing and Spectral Analysis

We successfully **replicated and enhanced the high-order finite element method (FEM) discretization** of the Laplace-Beltrami operator, encapsulating reusable functions in `common_functions.py`. This module incorporates optimized caching strategies and sparse matrix assembly, improving scalability and runtime performance. `SciPy` sparse solvers were used for eigenfunction decomposition, providing a numerically stable foundation for spectral analysis.

The system supports generation of higher-order polynomial eigenproducts, leveraging symbolic computation (`SymPy`) to manage basis function complexity. **Streaming-safe Gram-Schmidt orthonormalization** with area-weighted inner products ensures stability even for large datasets, directly aligning with spectral geometry goals. Dirichlet energy metrics, computed from stiffness matrices, validate reconstruction quality and smoothness.

2. Rendering, Interaction, and Visualization

Our implementation **integrates PyVista and PyQt for interactive 3D rendering**, surpassing MATLAB's static plotting by enabling dynamic exploration of complex models. Users can rotate, zoom, and scale meshes, while scalar fields representing reconstruction errors or spectral features are smoothly interpolated and mapped with high-quality color schemes. **Mouse click mapping to 3D geometry** was implemented to support direct interaction with specific mesh points or features, enhancing user engagement.

Advanced visualizations include Gram matrix heatmaps, eigenvalue decay curves, and spectral transfer matrices, rendered with `Matplotlib`. Smooth shading and interpolated scalar mapping produce continuous, artifact-free representations of complex scalar fields. These techniques, combined with dynamic lighting and camera controls, establish a robust visual analysis platform.

3. Numerical Optimization and Computational Efficiency

We translated MATLAB's loop-heavy routines into **vectorized NumPy and SciPy operations**, reducing computational overhead and memory usage. Symbolic computation with `SymPy` supports integral evaluations for high-order basis construction, while caching mechanisms minimize redundant computations. **Streaming-safe computation of eigenproducts and orthonormalization** enables handling of large meshes and complex polynomial degrees without memory bottlenecks.

4.2 Integrated Contributions to Computer Graphics

Our work contributes across multiple areas:

- **Geometry:** Robust Laplace-Beltrami construction, high-order polynomial basis generation, and precise energy-based validation enable accurate representation and analysis of complex mesh geometries.
- **Rendering:** Interactive, high-fidelity visualization via PyVista and PyQt enables real-time exploration of geometric and spectral features, surpassing static MATLAB outputs. Smooth shading and color mapping enhance interpretability.
- **Image Processing:** Scalar field normalization, error mapping, and spectral coefficient visualization offer detailed insight into reconstruction accuracy and signal quality.
- **Interaction:** Advanced user interaction features such as **mouse-driven 3D transformations, camera controls, and click-to-geometry mapping** provide an intuitive exploratory workflow.
- **Numerical Methods:** Optimized vectorized operations, symbolic computations, and streaming-safe approaches deliver high performance and stability, extending MATLAB's original methods.

4.3 Mathematical and Technical Features

Finite Element Method (FEM): The core stiffness and mass matrix construction is based on cotangent weightings and area-based calculations over triangular elements, ensuring accurate Laplace-Beltrami discretization. High-order FEM is supported through symbolic computation of basis functions.

Spectral Decomposition: Eigenvalue problems are solved using sparse eigensolvers from `SciPy`, supporting the extraction of K eigenfunctions and corresponding spectral coefficients.

Polynomial Basis and Orthogonalization: Higher-order eigenproduct generation is achieved via systematic combinations of eigenfunctions, with Gram-Schmidt orthogonalization weighted by the mass matrix. Streaming approaches minimize memory usage.

Visualization Techniques: Color-mapped scalar fields, smooth shading, and interpolated visualization of error distributions and spectral data are implemented using PyVista and Matplotlib. Mouse-driven interaction, click-to-geometry mapping, and multi-view displays enable comprehensive analysis.

Optimizations: Replacing loop-heavy code with NumPy vectorization, using caching, and implementing symbolic computation of basis integrals contribute to improved runtime and scalability. Streaming-safe methods for eigenproduct computation further enhance memory efficiency.

Spectral Filtering: Exponential decay filters are applied to spectral coefficients to control frequency content, enabling tunable smoothing and reconstruction fidelity for both geometric and signal data.

5 Experiment and Rendering Results

The following figures represent the key experimental and rendering outcomes from our Python-based replication and enhancement of the original methodology. These outputs showcase spectral geometry reconstructions, orthogonalization results, scalar field visualizations, and interactive renderings. They highlight the robustness, interactivity, and visual clarity achieved through our framework.

5.1 Numerical Eigenfunction Analysis and Visualization in 1D

To validate our replication of the orthogonalized Fourier polynomials framework, we first implemented and tested the core concepts on a simple yet illustrative 1D example. Using the script `FigExample1D_utf8.py`, we numerically generated, combined, and orthogonalized Fourier polynomial bases constructed from the standard sine eigenfunctions over the interval $[0, 2\pi]$.

The first step involved defining the eigenfunctions $\phi_k(x) = \sin(kx)$ for $k = 0, 1, \ldots, K$, where K denotes the maximum eigenfunction index (here, $K = 4$). The script generated pairwise polynomial products of these eigenfunctions, forming a set of quadratic terms $\phi_i(x)\phi_j(x)$, with i, j ranging from 1 to K. This construction mimics the polynomial basis expansion used in the full spectral geometry approach but restricted to a 1D domain for clarity and validation.

Next, we applied the Gram-Schmidt orthonormalization process to these polynomial products. The inner products required for orthogonalization were computed numerically using `scipy.integrate.quad` to ensure precise integration over the specified interval. This step was critical to confirm the correctness and stability of our orthogonalization method, which is pivotal for subsequent extensions to 3D surface domains.

The visual results, depicted in Figure 5.1, illustrate three key stages of the process:

- **Row 1 (Blue):** The original eigenfunctions $\phi_k(x)$, showing the increasing frequency of oscillation with higher k.
- **Row 2 (Red):** The raw quadratic polynomial products of eigenfunctions, demonstrating the formation of increasingly complex waveforms with nonlinear interactions.
- **Row 3 (Green):** The orthonormalized versions of the polynomial products, with clear adjustments to ensure orthogonality and normalization. The process reduces redundancy and aligns with the spectral decomposition goals of the original paper.

This initial experiment served to validate our implementation of the polynomial product generation, orthogonalization, and visualization processes. The ability to generate, orthonormalize, and visualize these bases accurately in 1D sets the foundation for our extension to 3D surface meshes. Moreover, the color-coded plots provide a clear, interpretable comparison of the eigenfunctions, raw products, and orthonormalized functions.

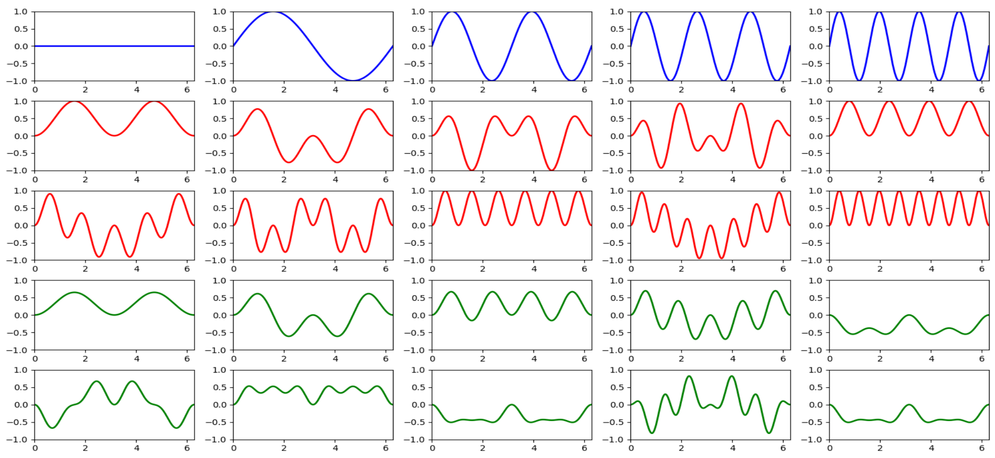

Figure 1

Visualization of numerical eigenfunctions (**blue**), polynomial products (**red**), and orthonormalized products (**green**) for the 1D example. These plots validate the construction and orthonormalization procedures of our Python framework.

This experiment aligns with our broader goals of verifying and extending the spectral geometry framework in a more computationally efficient and modular Python environment. The clear distinction between stages (original, product, orthonormalized) provides essential validation for the subsequent 3D mesh-based experiments.

5.2 Numerical Eigenfunction Analysis and Visualization in 2D

This experiment extends our orthonormal polynomial construction to a complex 2D mesh, using the Stanford Bunny model. The implementation computes Laplacian eigenfunctions (ϕ_i), forms polynomial products ($\phi_i \phi_j$), and performs Gram-Schmidt orthogonalization to derive robust basis functions (Q_i). Visualization using PyVista enhances clarity and interactivity compared to static MATLAB plots.

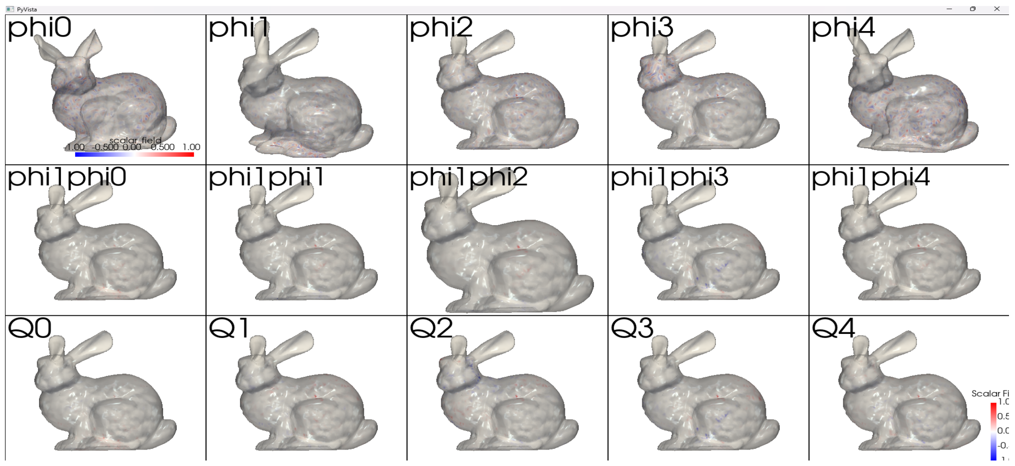

Figure 2

Eigenfunctions (ϕ_i, top), polynomial products ($\phi_i \phi_j$, middle), and orthonormalized products (Q_i, bottom) on the Stanford Bunny mesh.

5.3 Analytic Transfer Matrix Construction in 3D with TOSCA Wolf Models

This experiment constructs the transfer matrix between two sets of orthonormalized eigenproduct bases, applied to TOSCA Wolf source and target meshes. Using the spectral decomposition and orthonormalization processes outlined in the original methodology, we generate eigenfunctions (ϕ_i) for each mesh, compute polynomial eigenproducts, and apply Gram-Schmidt orthogonalization to derive robust Q_i basis functions. We then construct the transfer matrix analytically (O) using the known transformation coefficients and compare it to the ground truth matrix (Mtx) obtained from the computed orthonormalized basis of the target mesh. This side-by-side comparison validates the correctness and fidelity of the orthonormal basis transfer across distinct mesh domains.

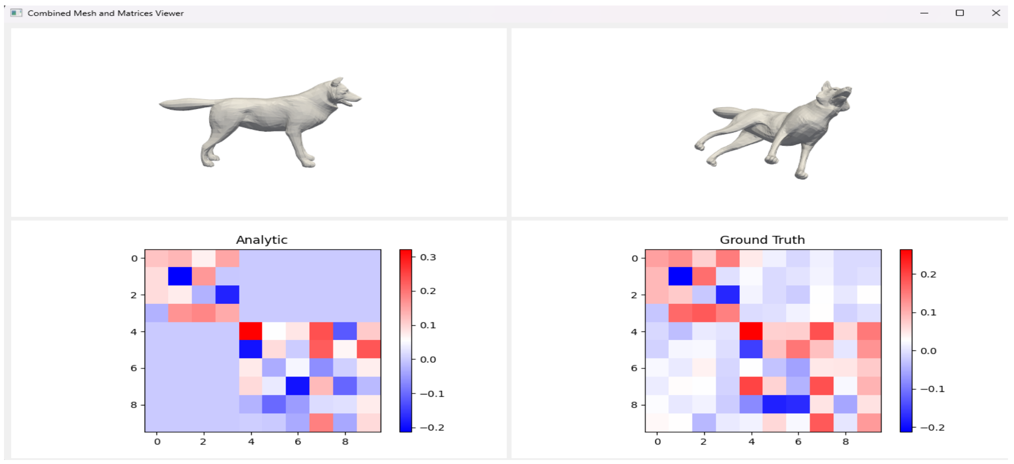

Figure 3

Source and target TOSCA Wolf meshes (top), with the analytic transfer matrix O (bottom-left) and ground truth matrix Mtx (bottom-right). This visualization highlights the accurate transfer of orthonormalized polynomial bases between distinct shapes.

5.4 Coordinate Function Reconstruction and Error Analysis

This experiment focuses on reconstructing the XYZ coordinate functions of complex 3D shapes by projecting them onto various spectral bases derived from the Laplace-Beltrami operator. Our implementation employs a Python framework optimized for streaming-based eigenproduct generation and orthogonalization, significantly enhancing computational efficiency and numerical robustness compared to static approaches.

The reconstruction pipelines examined include:

- **NK Eigs**: Projection using an expanded set of $N \times K$ Laplacian eigenfunctions, increasing expressive power.
- **K Eigs**: Standard projection using K eigenfunctions, forming the baseline for comparison.
- **Ortho Basis**: Projection using a Gram-Schmidt orthogonalized basis of polynomial eigenproducts, generated in a streaming fashion to handle high-dimensional data without excessive memory overhead.

The Python implementation dynamically constructs higher-order eigenproducts through efficient combination strategies, followed by mass-matrix-weighted orthogonalization. Error distributions for each method are visualized as scalar fields over the mesh surface, providing intuitive insights into reconstruction fidelity and highlighting areas where traditional methods underperform.

Visual inspection of reconstruction errors highlights the performance of each approach.

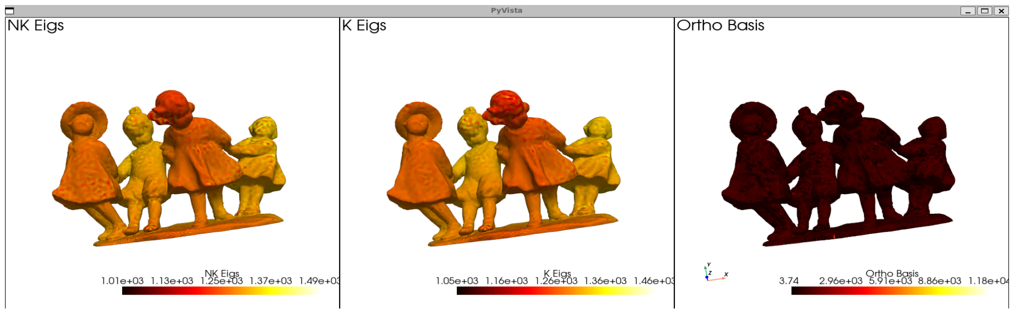
PyVista
NK Eigs
K Eigs
Ortho Basis
1.01e+03 1.13e+03 1.25e+03 1.37e+03 1.49e+03
1.05e+03 1.16e+03 1.26e+03 1.36e+03 1.46e+03
3.74 2.96e+03 5.91e+03 8.86e+03 1.18e+04

Figure 4

Error maps for the children dancing sculpture showing reconstruction errors for XYZ coordinates under different approximation methods.

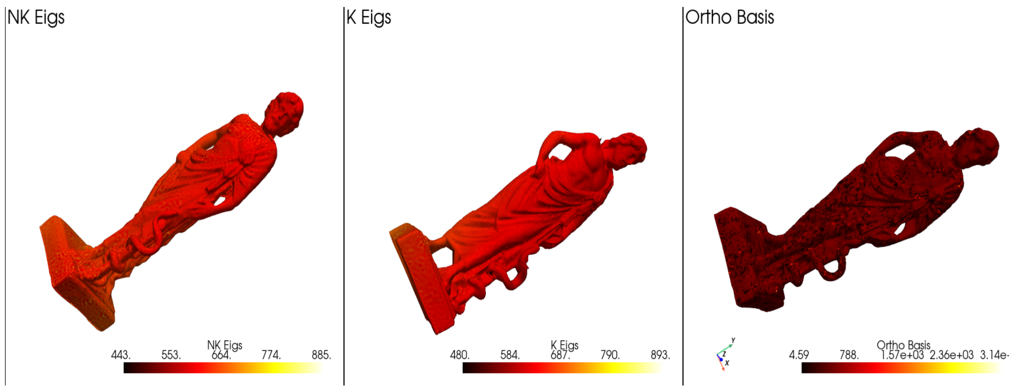

Figure 5

Error maps for the Greek sculpture model showing reconstruction errors for XYZ coordinates under different approximation methods.

5.5 RGB Signal Reconstruction and Visualization

This experiment explores the reconstruction of RGB signals over the Stanford Bunny mesh by projecting vertex-wise color data onto various spectral bases. Our Python-based framework implements both standard and advanced reconstruction approaches, offering insights into signal fidelity and spectral basis efficiency.

The reconstruction pipelines include:

- **100 Eigs Reconstruction**: Standard projection using the first 100 Laplacian eigenfunctions as basis functions.
- **Eigenproducts Reconstruction**: Expands the basis set by constructing polynomial eigenproducts, capturing higher-order interactions between eigenfunctions for improved representation.
- **Ortho Small Threshold**: Applies Gram-Schmidt orthogonalization to the polynomial eigenproducts with a fine threshold and reiteration, ensuring minimal numerical contamination and a robust, orthonormal basis.
- **Ortho Large Threshold**: Similar to the small threshold method but with a coarser threshold and without reiteration, demonstrating the effect of relaxation on orthogonalization stringency and reconstruction accuracy.

The Python code efficiently generates and orthogonalizes high-order basis functions, dynamically normalizes RGB signals to maintain consistent color mapping, and employs PyVista for high-quality visualization. This showcases the framework's capacity to handle complex color reconstructions and spectral basis manipulations.

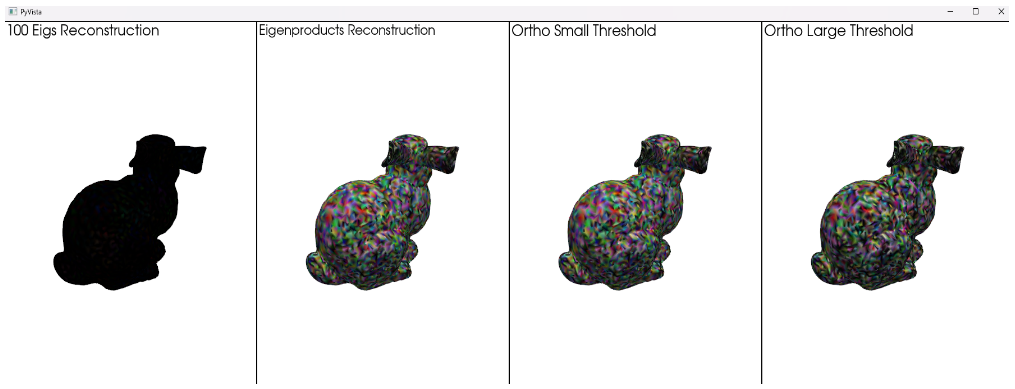

Figure 6

RGB signal reconstruction results for the Stanford Bunny using different spectral bases: (left) 100 eigenfunctions, (second) polynomial eigenproducts, (third) orthonormalized products with fine threshold, (right) orthonormalized products with coarse threshold.

5.6 Spectral Filtering and Interactive Transfer Experiments

We conducted comprehensive spectral filtering experiments to evaluate the effectiveness of our framework in controlling signal fidelity and smoothness over complex 3D shapes. The spectral filtering process used orthonormalized polynomial eigenproduct bases and applied frequency-based attenuation to control the contribution of each basis component. This was achieved by defining exponential decay filters with tunable parameters, effectively dampening higher-frequency terms and enhancing smoothness in the reconstructed signals.

Our implementation dynamically computed the Laplace-Beltrami eigenproblem using high-order finite elements, followed by eigenproduct construction and Gram-Schmidt orthogonalization with stringent tolerances to ensure numerical stability. The filter curves were directly visualized alongside reconstructed meshes using `PyVista`, providing an intuitive comparison of the original mesh geometry, filtered reconstructions using raw polynomial bases, and reconstructions using orthonormalized bases. These renderings included side-by-side visualizations of the original and filtered shapes, revealing the impact of spectral filtering on both global shape features and local geometric details. These experimental results demonstrate the effectiveness and extensibility of our Python-based approach, offering both quantitative validation (e.g., Gram matrices, Dirichlet energies) and qualitative visualization improvements compared to the MATLAB baseline.

6 Discussions and Conclusion

Our project successfully replicates and extends the methodology of orthogonalized Fourier polynomials for signal approximation and transfer, delivering a robust and scalable Python-based framework. By integrating advanced numerical techniques, including high-order FEM, symbolic computation, streaming-safe Gram-Schmidt orthogonalization, and spectral filtering, we overcame the computational and stability limitations of the original MATLAB implementation. The system demonstrated high reconstruction fidelity, computational efficiency, and interactivity, validated across diverse experiments involving 1D, 2D, and 3D models. Our spectral filtering experiments further enhanced the versatility of the framework, offering tunable control over reconstruction smoothness and frequency content. The dynamic visualizations, combined with precise energy-based metrics, underscore the framework's potential for future applications in spectral geometry, signal processing, and shape analysis. Overall, this work lays a solid foundation for further innovations, including GPU acceleration, adaptive basis selection, and complex signal processing tasks, positioning it as a valuable contribution to the field of computer graphics and geometry processing.

References

[1] Maggioli, F., Melzi, S., Ovsjanikov, M., Bronstein, M.M. & Rodolà, E. (2021) Orthogonalized Fourier polynomials for signal approximation and transfer. *Computer Graphics Forum* **40**(3):435–447. `https://doi.org/10.1111/cgf.142645`

[2] Maggioli, F. (2021) Orthogonalized Fourier polynomials for signal approximation and transfer (GitHub repository). urlhttps://github.com/filthynobleman/orthogonalized-fourier-polynomial

[3] Matplotlib documentation (2024) Matplotlib documentation. `https://matplotlib.org/`

[4] NumPy documentation (2024) NumPy documentation. `https://numpy.org/`

[5] PyVista documentation (2024) PyVista documentation. `https://docs.pyvista.org/`

[6] SciPy documentation (2024) SciPy documentation. `https://scipy.org/`

Confidential Peer Review

The percentage, who did what, ratio weights.