# Final Project: Mesh Morphing & Animation Toolkit

Names: Tamnhi Vu, Brandon Lai, Tara Pande, and Samuel German

Link to webpage:
https://tamnhivu.github.io/Final-Writeup/
Link to Video:
https://www.youtube.com/watch?v=bV2DG_qFEog
Link to Slides:
https://docs.google.com/presentation/d/1rngVR-UcvMoFuPJwmkLfqyWITVZGPm_yzBh-j0uAUo0/edit?usp=sharing

Our goal

## Milestone Report

Our mesh morphing and animation toolkit, of course, needs a user interface. Currently the user interface, hosted on Github, is able to load a variety of .dae files taken from HW2. These dae files can be presented under a variety of shading parameters (wireframe, flat, smooth, glossy, textured, reflective), and can be rotated and scaled by the user directly with their mouse in real time. The main backend utilizes three.module.js, which is responsible for creating the PerspectiveCamera, lighting, renderer, TextureLoader, the various MeshMaterials, and the Scene that the .dae lives in. There are a variety of other JavaScript files responsible for creating the GUI, ColladaLoader (loads the .dae), and OrbitControls (to control the camera movements).

Our implementation includes multiple vertex interpolation algorithms for 3D mesh animation. The codebase now features linear interpolation between individual vertices and entire meshes, cubic Bézier curve interpolation for smooth motion between control points, blend shape interpolation that combines multiple meshes with different weights, Catmull-Rom spline interpolation for smooth keyframe animation, and barycentric interpolation for projecting points onto triangular surfaces. The linear interpolation function provides the most basic transition between meshes, while Bézier curves offer greater control over the animation path. Blend shapes enable complex facial animations through weighted combinations of expression meshes. The Catmull-Rom spline implementation ensures C1 continuity between keyframes for fluid motion sequences, avoiding the abrupt transitions that simpler methods might produce. Barycentric coordinates enable point projections onto triangular faces, useful for texture mapping and deformation. These methods collectively provide the foundation for creating fluid animations and realistic transitions in our 3D animation system.

Our mesh processing allows us to form advanced ways for us to handle 3D mesh deformation and alignment. The Bezier curve allows us to do interpolation for vertex transition, the Laplacian smoothing allows us to get geometry refinement, and the Iterative Closest Point does the mesh registration. More specifically, the Bezier interpolation lets us have a smooth transition from vertex to vertex between two or more meshes that allows the meshes to blend more seamlessly. The Laplacian smoothing improves the quality of the mesh by incorporating averaging the vertex position. Our iterative closest point lets us align the original mesh (source) to the final mesh (target) by finding the closest points, calculating the centroids, and finally doing the transformation. We found this important since we want to be able to handle mesh registration and shape matching.

To refine and smooth triangle meshes, typically edges are split and flipped to add new vertices and faces. We created a loop subdivision algorithm that has no edge split or flips and which just moves vertices according to the laplacian fairing, aiming to preserve the underlying topology of the mesh.

Currently, we've started to incorporate the vertex interpolation, mesh processing, and topology preservation algorithms into our user interface. The best way to do this so far has been to turn our native algorithms in C into JavaScript, so that it can be read and used by the main HTML behind the user interface. So far, this is really great progress. Overall, we are still on track to have a toolkit that can animate and morph meshes between .dae files by the final project deadline since we were able to implement all the algorithms to make a slider that will help one mesh transform to the other. However, we are still trying to figure out how to make our implementations compatible with one another to get our final result and will update accordingly.