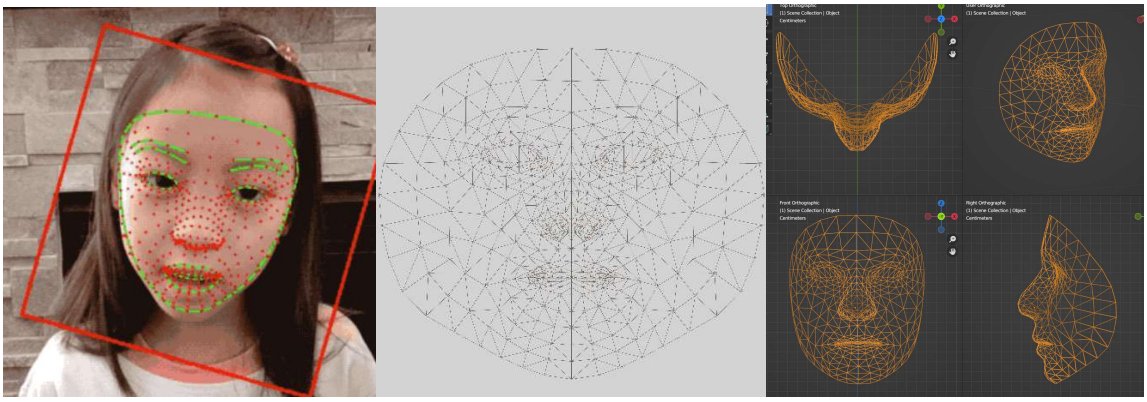
Using MediaPipe to automatically transfer texture map UV layouts between arbitrary face meshes



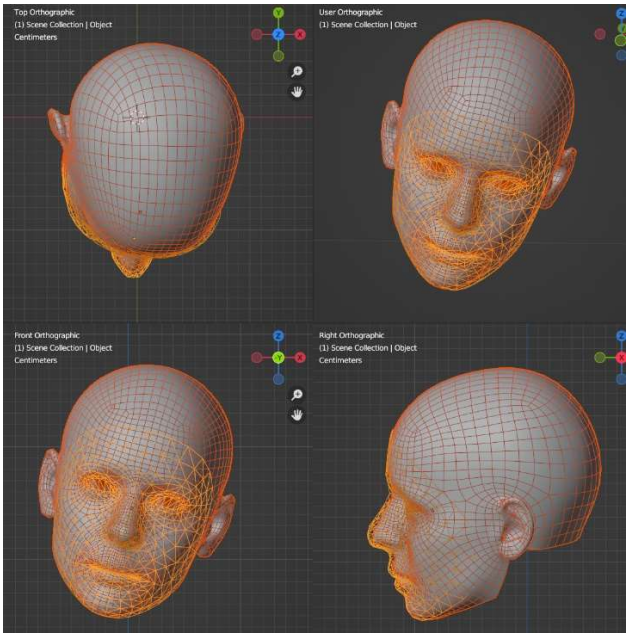
MediaPipe's face mesh projected onto a girl's face

MediaPipe is an open-source Google library that given a 2D image of a human face, the MediaPipe *3D face mesh* is three dimensionally superimposed onto the 2D image such that 468 independent facial landmarks are placed onto the 2D image in the same manner as if the MediaPipe *3D face mesh* were three dimensionally rendered over (and fit to the shape of) the human face in the 2D image.

Essentially, MediaPipe provides enough information that with a video of a person turning their face in different directions both a good quality 3D mesh of the shape of their face can be constructed, as well as a good quality texture map. MediaPipe provides enough information such that a decent quality 3D likeness, or 3D avatar, of the person's face may be created.

Note that two correlated mappings take place here: a 3D position map of the person's face to the MediaPipe face mesh, and a 2D mapping of the person's face into the *UV Space defined by the MediaPipe face mesh*. The middle image in the figure above is the UV space of the MediaPipe face mesh; it is simply the 3D points of the 3D mesh flattened to 2D and spread out a bit to compensate for the area each polygon represents in 3D.

Now, what do we get when MediaPipe is given a 3D rendering of an arbitrary 3D mesh of a human face? Just as before, we receive the MediaPipe *3D face mesh* three dimensionally superimposed onto the 2D rendered image of our 3D mesh:



However, that is not just a mapping of the MediaPipe 3D face mesh into a 2D image, because we know the 3D orientation of the arbitrary 3D mesh, it is also a mapping between the two independent 2D UV Spaces contained within each 3D mesh: the MediaPipe face mesh and the other arbitrary 3D face mesh.

Both meshes represent a 3D surface mapping to a 2D texture space. The 2D texture space is referred to here as the UV Space defined by each mesh. Just as each mesh has its own UV Space, when MediaPipe superimposes its face mesh over a 2D rendering of another mesh, MediaPipe is providing the information to translate between the UV Spaces of MediaPipe and the other geometry. The other geometry may have a completely arbitrary topology, and MediaPipe's superimposition shows us where the two meshes correlate; where to copy texture information if one wanted to create a texture map in the non-MediaPipe face mesh UV Space.

Now, what do we get when two arbitrary face meshes, each with a different topology and UV layout as the other, has a MediaPipe face mesh superimposed on each? MediaPipe becomes *the neutral UV Space in which the UV Space (UV Layout) of the first face mesh can be projected onto the UV Space of the second face mesh.*

Note: to account for the variations of stretch between UV Spaces of each mesh, the MediaPipe UVs are only used as the common mapping space when translating polygon by polygon on the target mesh, back to the MediaPipe UV Space, and from there locating the corresponding UV locations (target polygon by target polygon) in the source texture UV Space, UV Layout, source texture. This double indirection is key to account for the baked in non-linear distortion that naturally occurs when a 3D surface area is flattened to a 2D map.

The process of transferring, for example, a FaceBuilder texture and mesh to a SkaterXL texture and head mesh, requires these steps:

The overall process:

1) Generate a FaceBuilder model with a MediaPipe face mesh superimposed onto the surface of the FaceBuilder model;
2) Generate a Game model with a MediaPipe face mesh superimposed onto the surface of the Game model;

3) Starting from the Game model, polygon by polygon map each Game model polygon's UV space to its corresponding UV values in the FaceBuilder UV Space. This is accomplished by calculating the UV values of each vertex of the Game mesh polygon from the corresponding same polygon as is lying on the Game mesh as is lying on the FaceBuilder mesh at that same location.

Both overall steps 1 and 2 are using MediaPipe with renders of some 3D face model to calculate the location of each MediaPipe mesh vertex on the surface of the other 3D face model. These two steps are repeats of the same process, just with different 3D models having the MediaPipe face mesh placed on its surface. Performing this *MediaPipe face mesh attachment* requires the use of *raycasting* to determine the location of each MediaPipe mesh vertex on the surface of the other mesh (FaceBuilder or Game mesh).

An additional issue in performing *MediaPipe face mesh attachment* is the fact that MediaPipe's 3D superimposition is approximate. The MediaPipe face mesh placement on the rendered face image is more accurate for portions of the face facing the camera:

3 renders of the SkaterXL head mesh with the MediaPipe face mesh superimposed

In the above three images, the left two provide a more accurate placement for the middle polygons of the face, while also providing a less accurate placement of the left and right sides of the face polygons. The accuracy variation between the left two image's middle polygons also has a bias between them: the left-most image has lower polygons placed more accurately – those around the jawline, lips, and the vertices forming the bottom of the nose. The middle image has an accuracy bias closer to the top of the eyebrows.

The approximate nature of the MediaPipe face mesh placement is the clearest in the right-most image: notice the left side of the MediaPipe face mesh is not even on the SkaterXL model. Meanwhile, the MediaPipe polygons near the camera facing ear and cheekbone are very accurate.

Due to the approximate nature of the MediaPipe face mesh placement, multiple rendered images of the 3D model need to be processed by MediaPipe with only some of the MediaPipe face mesh vertices recovered from each rendered image.

> For example, in the left two rendered images above, a weighted average may be used to filter away the left and right side of the head vertices, while the left-most has greater weights on the lower vertices and the middle image has stronger weights near the top of the face.

This performing a weighted average of the MediaPipe vertices is purely for the purpose of determining the 3-space location of each MediaPipe face mesh polygon. All the individual polygons of the MediaPipe face mesh need to be placed in 3-space using appropriate weighted averages. This is so each MediaPipe polygon's ***surface normal*** can be calculated. We need each MediaPipe polygon surface normal so we can calculate the **weighted average surface normal for each MediaPipe vertex**. That weighted average surface normal of each vertex is then used as one of "forward" or "backward" direction vectors in *raycasts* used to locate that vertex on the surface of the SkaterLX model.

Now raycasting may be used, sending rays along the weighted average surface normal for each MediaPipe vertex. If a ray does not hit the SkaterXL model, send another ray in the opposite direction. The MediaPipe face mesh placement could have some vertices "in fron" and some "behind" the SkaterXL mesh surface. So sending rays forward and backward insure you'll hit the surface.

To complete the process of attaching the MediaPipe face mesh to the surface of the SkaterXL model, using the UV coordinates of the SkaterXL model, calculate the corresponding UV coordinates of the MediaPipe vertices on the surface of the SkaterXL model. Those UV coordinates can be used by other models to remap this model's texture space into their texture space. When completed, each MediaPipe polygon will have 3 vertices, and each of those vertices will have UV coordinates from the SkaterXL UV Space.

This attaching of the MediaPipe face mesh process is performed twice: once for the FaceBuilder mesh, and once for the SkaterXL head mesh.

Once both the FaceBuilder and the SkaterXL models have MediaPipe face meshes attached to them, Step 3 (top of the page) is ready to be performed.

Step 3 requires looping over the polygons of the SkaterLX model, looking to see if that polygon is either covered by MediaPipe polygons or is close enough to a MediaPipe polygon that the UV coordinates of the polygon's vertices may be calculated. However, do not get the UV coordinates from the SkaterXL model, locate the same MediaPipe polygon associated with that vertex *but from the FaceBuilder mesh*. By retrieving UV coordinates from the FaceBuilder's attached MediaPipe vertices, one is retrieving MediaPipe vertices loaded with the UV coordinates of the FaceBuilder UV Space. This process effectively remaps the FaceBuilder UV Space into the SkaterXL UV space, enabling the FaceBuilder generated texture to be copied polygon by polygon into the SkaterXL texture map.