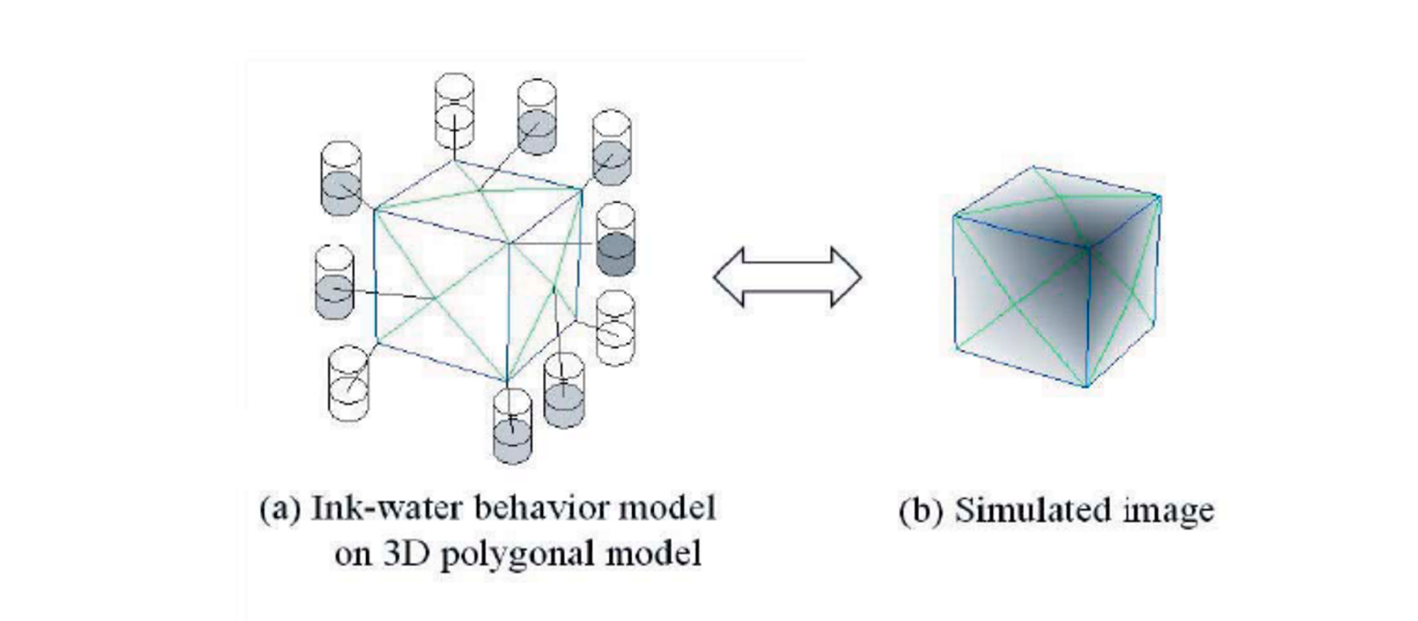
The basic idea of this project is to convert a Chinese traditional water-ink painting style animation into a virtual reality experience. The problem is obvious, all the techniques we know about water-ink painting can only be utilized in a 2d setting, how can we apply them in a 3d space?

After some diggings, we found several papers in non-photorealistic rendering that we think maybe helpful for our project. And later, we narrowed it down to one paper published by Korea University and decided to use the mathematic model described in that paper to implement our convertor. Here is the link to the paper:

<https://www.researchgate.net/publication/221434663_User-guided_3D_Su-Muk_painting>

The aim of the convertor is to render 3d models into a water-ink(Su-Muk) painting style with users’ guide. We implemented our convertor with python. The idea is to simulate the behavior of water-ink directly on 3d models so that after our simulation, ink and water will be spread over the surface appropriately.

We treat each vertex as a *water-tower* that contains certain amount of ink and water. More ink will reduce the brightness, increase the opacity, and the vice versa. During the simulation, the water will carry the ink, flowing from one vertex to another through the edge connecting them. The specific Ink-Water behavior model is based upon a cellular automaton which consists of an infinite amount of cells, each belongs to a finite number of states. In the cellular automaton, every cell has the same rule of updating based on values of its neighborhood. [1]. The picture below(from the paper), shows the 3d simulation model.



After taking account of the ***Transfer of Water Particles, Transfer of Ink Particles Accompanying Water Particles, Transfer of Ink Particles to Balance the Concentration, Evaporation of Water,*** we can then simulate the water-ink(Su-Muk) painting style on most of the 3d models.

We used python to do the simulations. In the development, we had several problems. The first one was performance. The 3d models we used for the simulations usually have many vertices and edges. The library we previously used couldn’t handle this amount of computation efficiently causing the debugging and parameters tuning a total pain in the ass (we need to wait a long period to see the results). We later found a library called PyMesh, a relatively young library published by an NYU PHD student. The computationally intensive part is written in C++ and the interface is written in python, making it easy to use and relatively fast for processing mesh geometries. Our convertor is heavily relied on this library and so far, it hasn’t let us down once. Here is a link to its github page: <https://github.com/qnzhou/PyMesh>

Another problem is vertex color and alpha channel. The result our convertor generated indicate how many water/ink particles are left on each vertex after simulation. The ratio of ink and water particles then suggests the alpha value of the vertex, the higher the ratio the larger the alpha value. However, most of the modeling software don’t support vertex RGBA color value. We spent almost one week trying to find a software that supports vertex RGBA color feature. Nothing came up. Thanks to Professor Shaffer’s advice, we started to implement our own alpha blender and it worked out pretty well in our settings. Below are some of our final rendered results.

It had been a great fun developing this converter. I hope you guys can enjoy our film.

To those nights in 1112 and VR lab in Siebel center.