Triangular mesh generation from 2D CT image

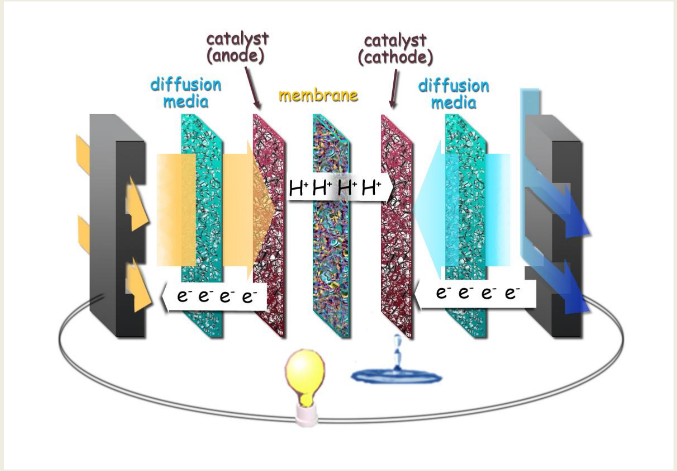
Arjun Kumar, Bo Ning, Xiaodong Wei

Outline

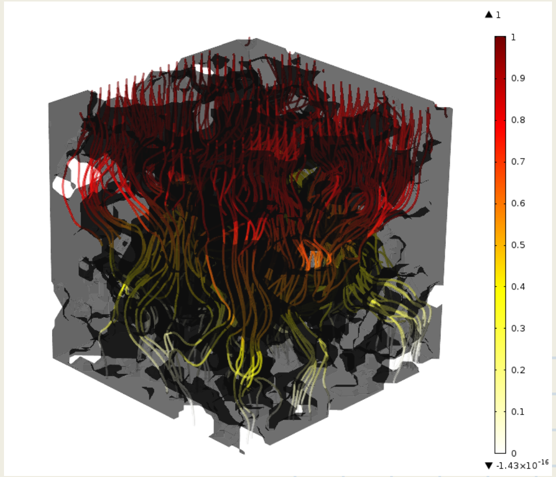
- Problem Definition
  - Application: Fuel cells
  - Input: CT Image
  - Output: Triangular mesh of 2D slice
    - Uniform and adaptive
- Pre-processing
  - Quadtree subdivision
- Algorithm
  - Marching Squares
- Implementation
- Results

Our Specific Application

- ➔ Fuel cells - Diffusion media, anode and cathode are porous materials, whose geometry can be important to the performance of the cell
- ➔ Their geometry is captured using nanoscale X-ray CT imaging
- ➔ Meshes are required to run simulations using that geometry
  - ◆ Possible applications also for 3D printing



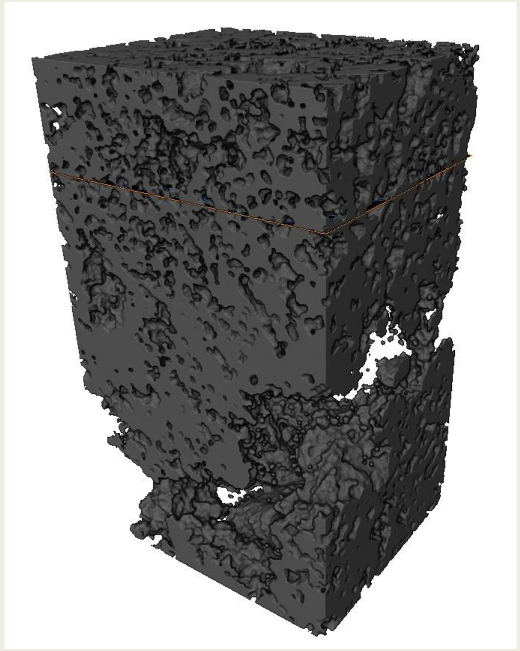
Schematic of diffusion media, anode and catalyst in fuel cell



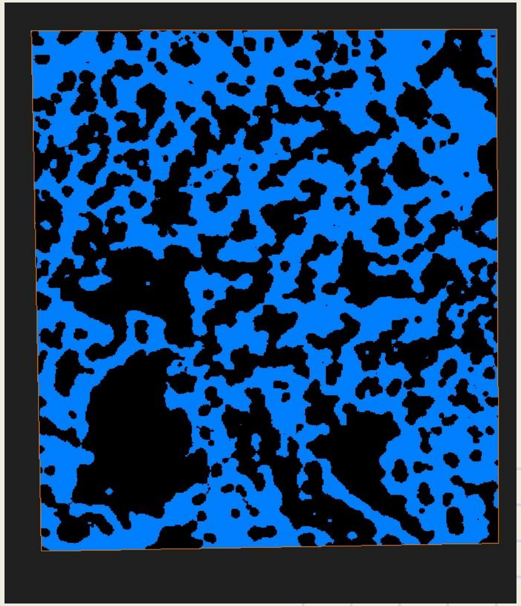
Streamlines showing results of a diffusion simulation in diffusion media

Input Image

- ➔ We take a slice of a 3D image of cathode of fuel cell
- ➔ We test our method with simple images for the slice of the 3D image
- ➔ One of the things we hope to achieve is to mesh both solid and air phases that are conformal to the boundary to simulate interphase transport

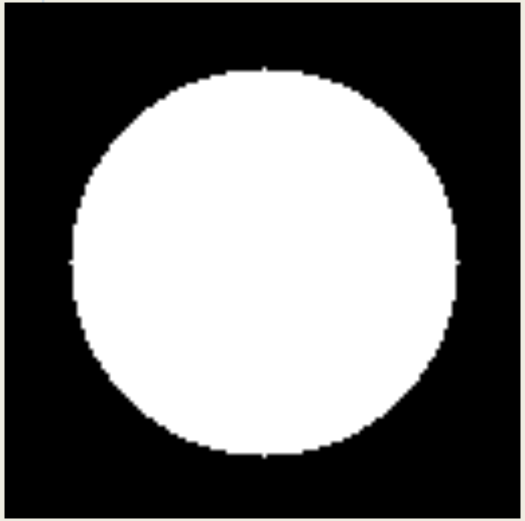


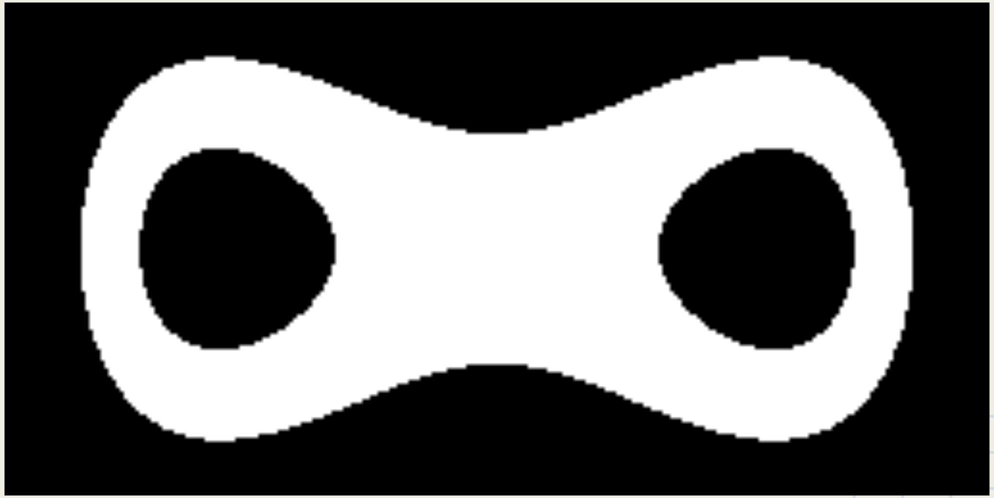
Segmented 3D image of cathode and slice selection



Input 2D image for project (blue = solid)

Images for Testing

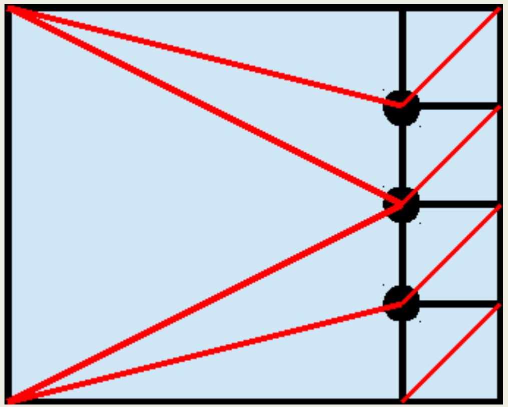




Pre-processing: Quad-Tree Subdivision

- For adaptive mesh generation, quadtree subdivision will allow us to use less triangles far away from the boundary and more more near the boundary

- Hanging Node - limit to one for convenience and quality of triangles

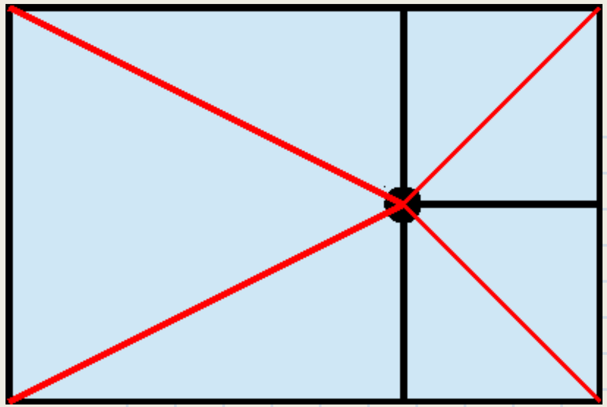


Multiple Hanging Nodes:

Poor

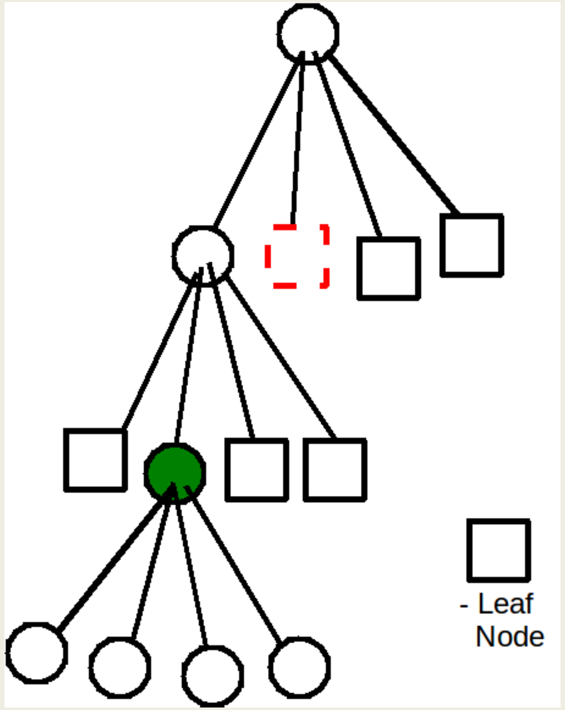
One Hanging Node:

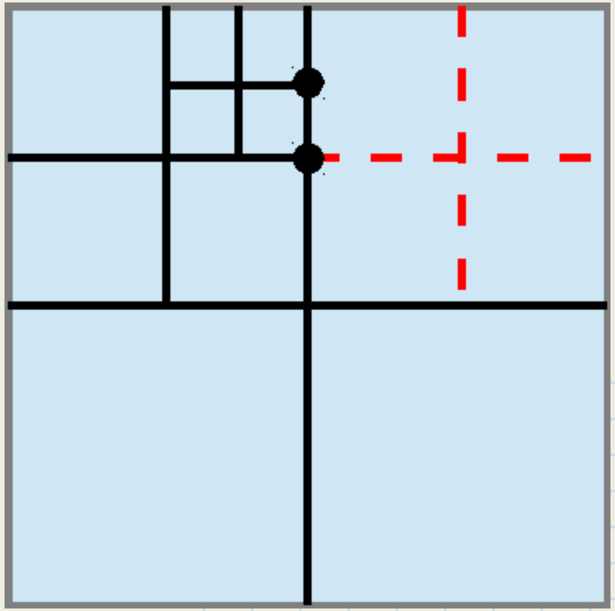
OK



Subdivision Implementation

- Breadth first search tree traversal
- Have to traverse backwards to keep depth of tree balanced





Transformation: Method

- **Marching squares**

➔ One of the most widely used methods in mesh generation from image data

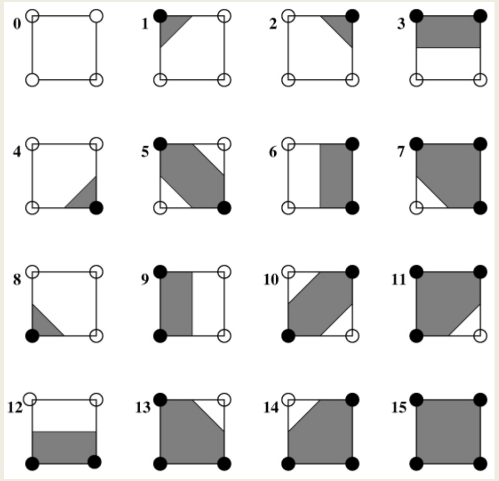
➔ Marching squares visits each cell of input image, and generate one or more polygons for each cell that represents the material

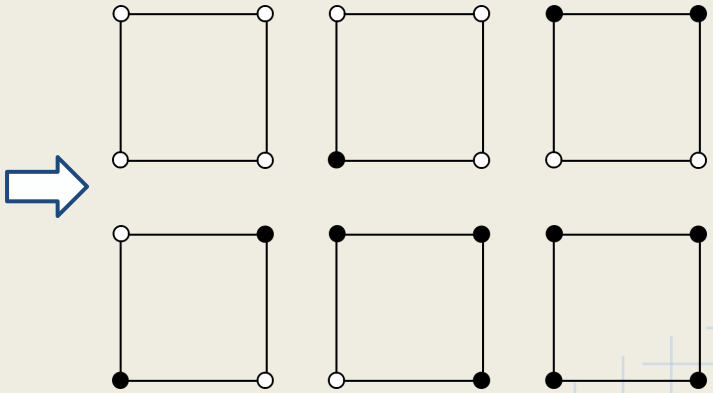
- ◆ Cell: each vertex has a value (0 or 1) to indicate material or air
- ◆ Material (1): black dots
- ◆ Air (0): circles

Transformation: Method

- **Marching squares**

→ Configuration of cells
- 16 cases depending on how vertices can be colored
- Can be mapped to 6 cases

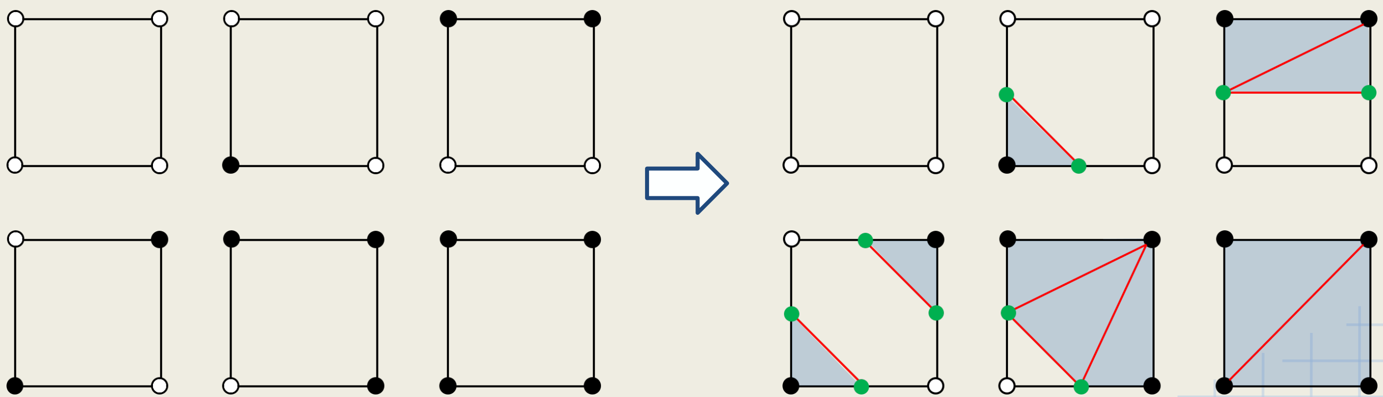




Transformation: Method

- **Marching squares**

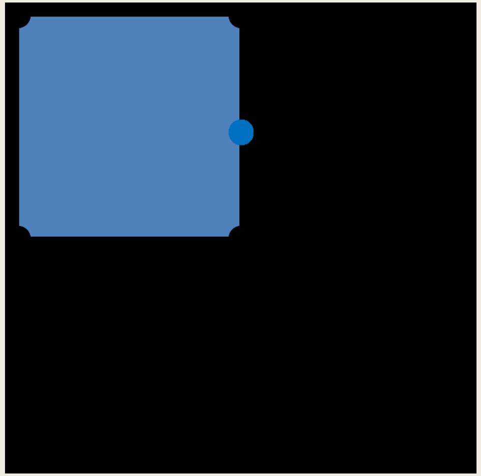
→ Triangulation of cells
  - ◆ Generate middle points of color-changed edges (green dots)
  - ◆ Add edges to obtain triangles (blue)

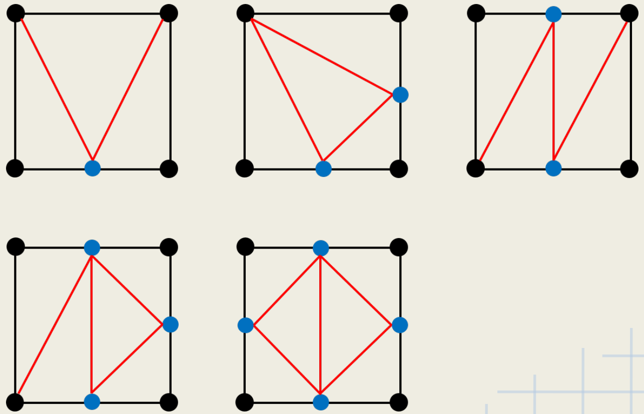


Transformation: Method

- **Marching squares**

➔ Connection with quad-tree subdivision
- ◆ Cells with hanging node on edges (blue dots)
- ◆ Restrict at most one hanging node on one edge
- ◆ 5 cases depending on the position and number of handing nodes



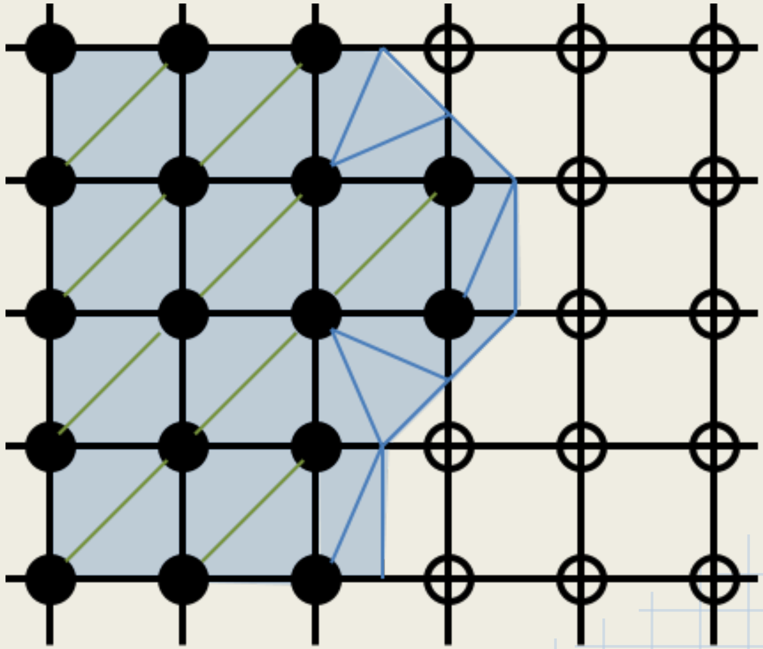


Transformation: Method

- **Marching squares**

➔ Uniform mesh generation
  - ◆ Without quad-tree subdivision
  - ◆ Done

➔ Adaptive mesh generation
  - ◆ With quad-tree subdivision



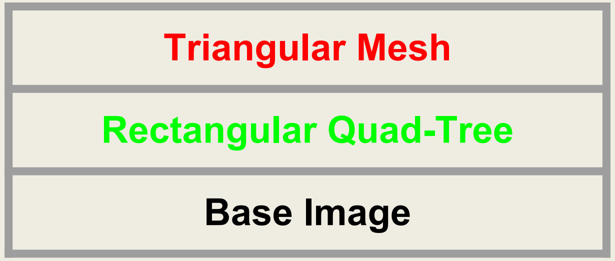
Post-processing:
STL and VRML File Generation

- **STL file**

→ File with simple syntax to represent the generated mesh
→ No many feature for color, polynomial and other information
→ Not very effective to demonstrate our workflow

- **VRML file**

→ File with many features to represent the geometric information
→ Large files for very complex geometry
→ Our major way to work on



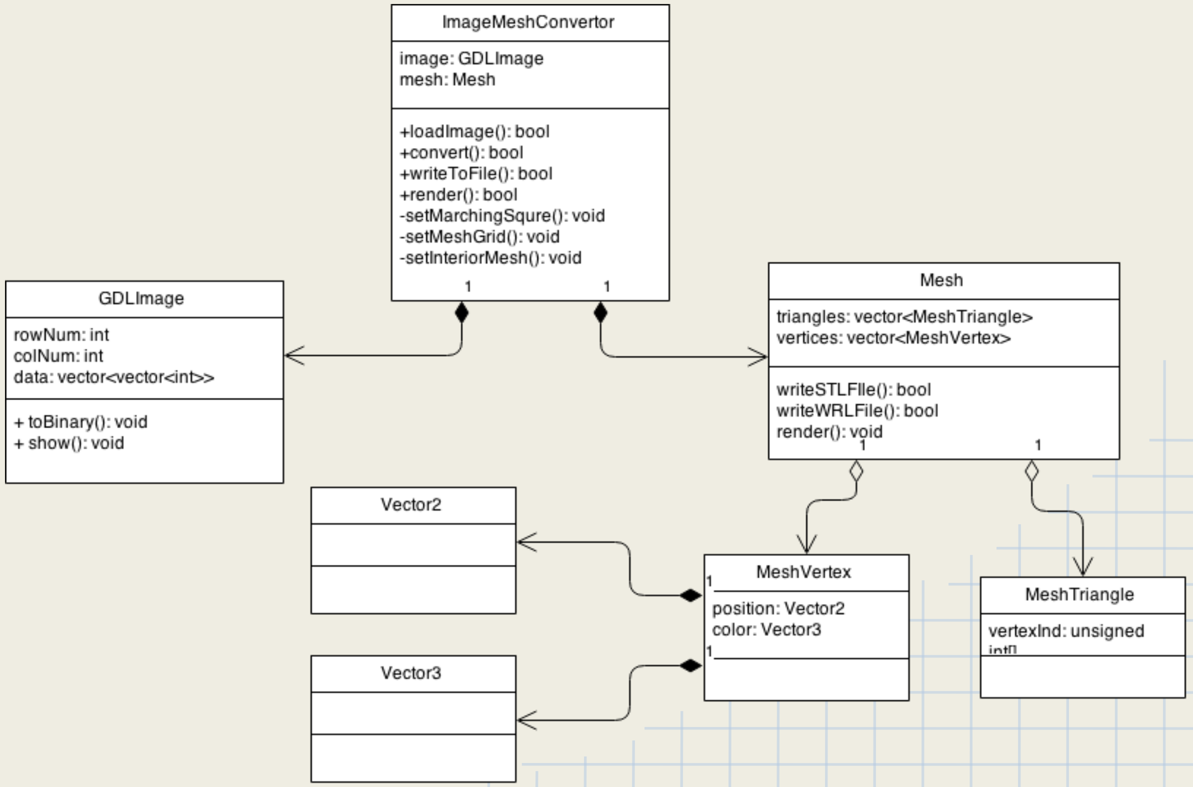
Object Model

- **Initial Design**

➔ ImageMeshConvertor
➔ GDLImage
➔ Mesh

- **Issues**

➔ No way to perform varied methods in concrete implementation
➔ Objects are intervening with each other
➔ Not good for task distribution for three members



Object Model

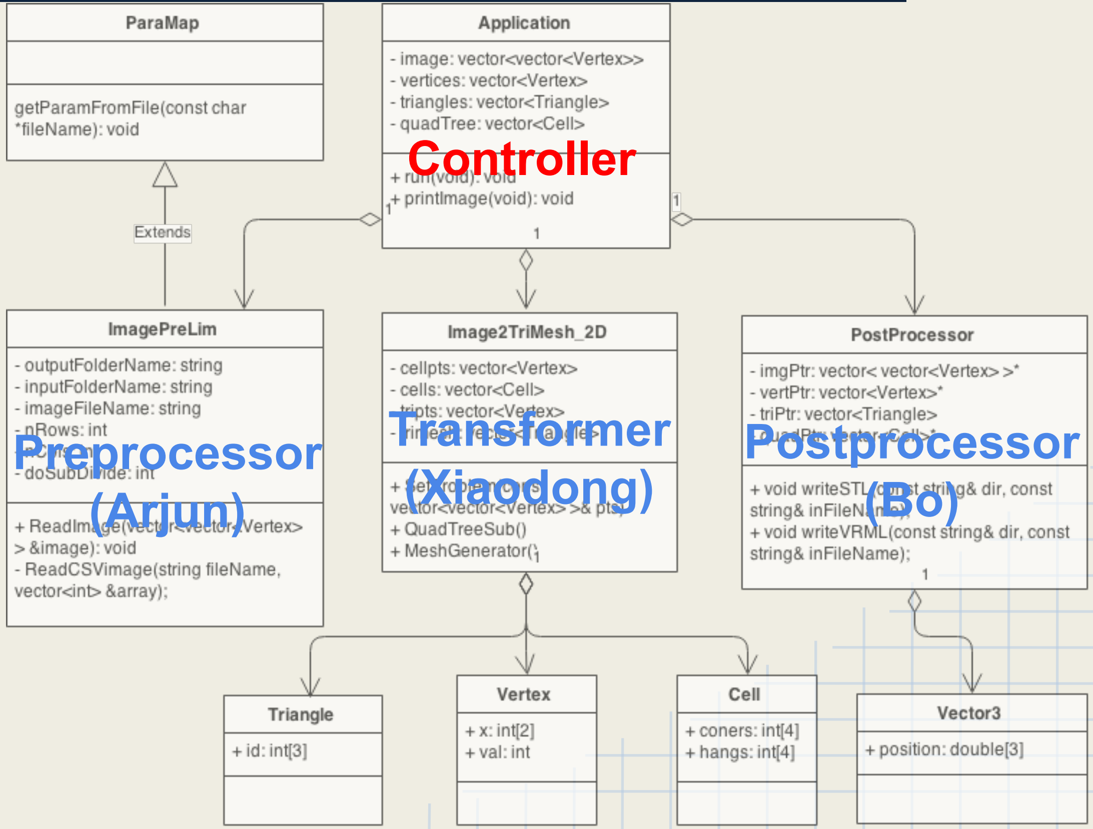
- **Final Design**

➔ Application Class
Controller to store the shared data, invoke the major routines

➔ ImagePreLie Class
Preprocessing class to read images and generate quad-trees

➔ Image2TriMesh_2D Class
Transformation class to do image to mesh transformation

➔ PostProcessor Class
Post-processing class to collect the data and generate files


ParaMap
getParamFromFile(const char *fileName): void
Extends
Application
- image: vector<vector<Vertex>>
- vertices: vector<Vertex>
- triangles: vector<Triangle>
- quadTree: vector<Cell>
+ run(void): void
+ printImage(void): void
Controller
ImagePreLim
- outputFolderName: string
- inputFolderName: string
- imageFileName: string
- nRows: int
- doSubDivide: int
+ ReadImage(vector<vector<Vertex>> &image): void
- ReadCSVimage(string fileName, vector<int> &array);
Preprocessor (Arjun)
Image2TriMesh_2D
- cellpts: vector<Vertex>
- cells: vector<Cell>
- tripts: vector<Vertex>
+ QuadTreeSub()
+ MeshGenerator()
Transformer (Xiaodong)
PostProcessor
- imgPtr: vector< vector<Vertex> >*
- vertPtr: vector<Vertex>*
- triPtr: vector<Triangle>
+ void writeSTL(const string& dir, const string& inFileName);
+ void writeVRML(const string& dir, const string& inFileName);
Postprocessor (Bo)
Triangle
+ id: int[3]
Vertex
+ x: int[2]
+ val: int
Cell
+ coners: int[4]
+ hangs: int[4]
Vector3
+ position: double[3]
1

Implementation

- **Object Oriented Programming with C++**

➔ We are implementing the program with C++ in Linux Platform
➔ Cmake to build the project
➔ Build automation

- **Version Control & Collaboration**

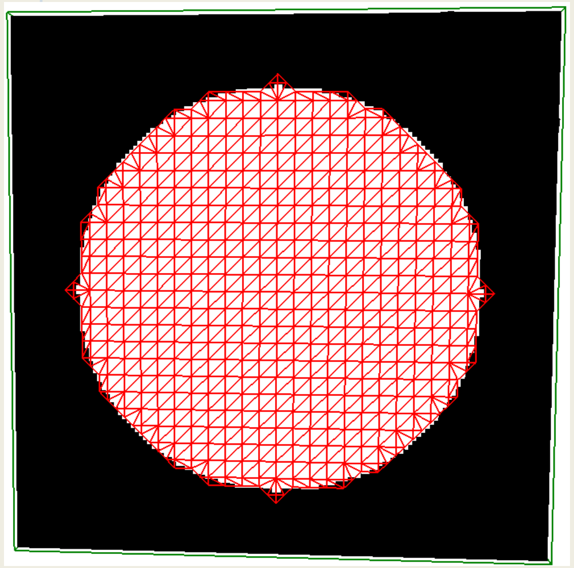
➔ We are using Github to work on the project collaboratively, the URL of our project repository is in: https://github.com/bning/CADCourseProject

- **Open Source Resources**

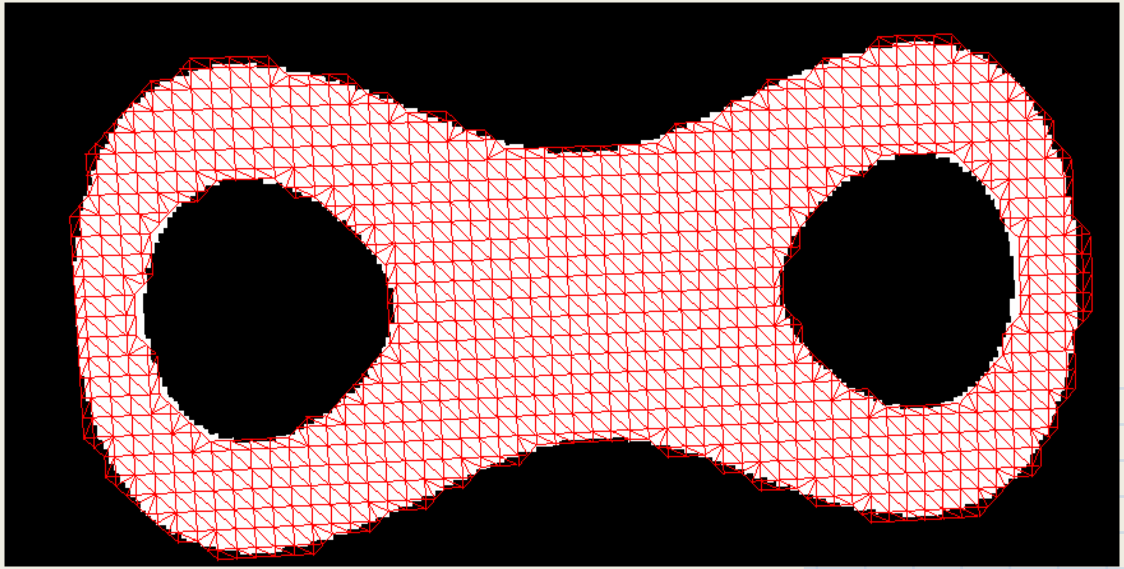
➔ glm library for some geometric calculation
➔ Cortona3D to render the images, quad-trees and meshes

Results

- **Uniform Mesh Generation**



TestCircle



TestDoubleTorus

Results

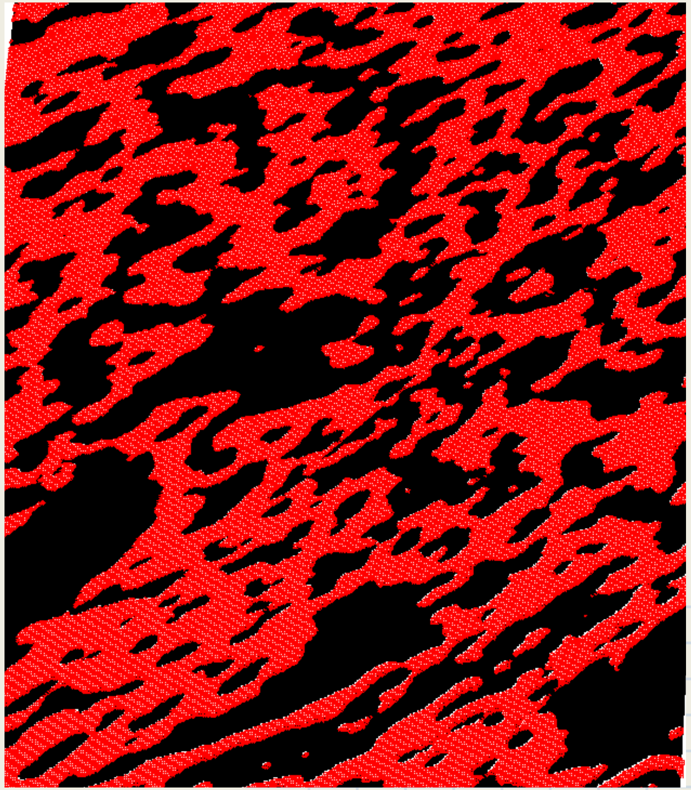
- **Issues**

→ Not representative for the details and fine regions

→ Low efficiency in transformation and post-processing

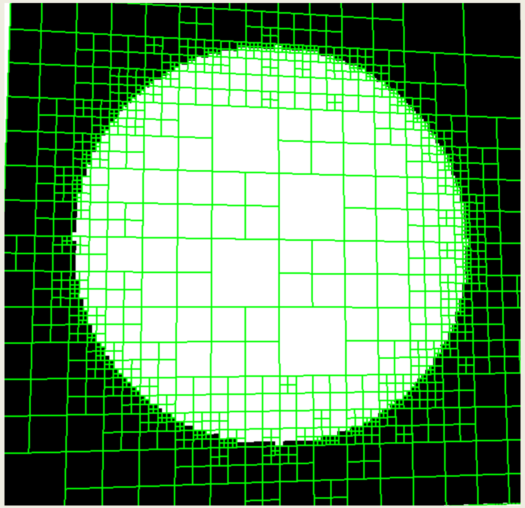
→ Would give very large files that are slow to load and render

GDLImage

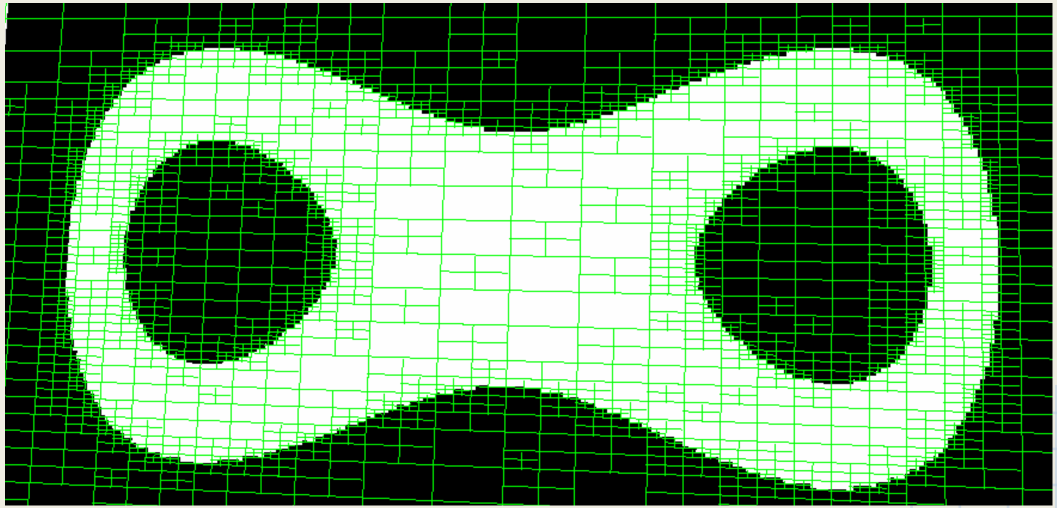


Results

- **Quadtree for adaptive mesh generation**



TestCircle

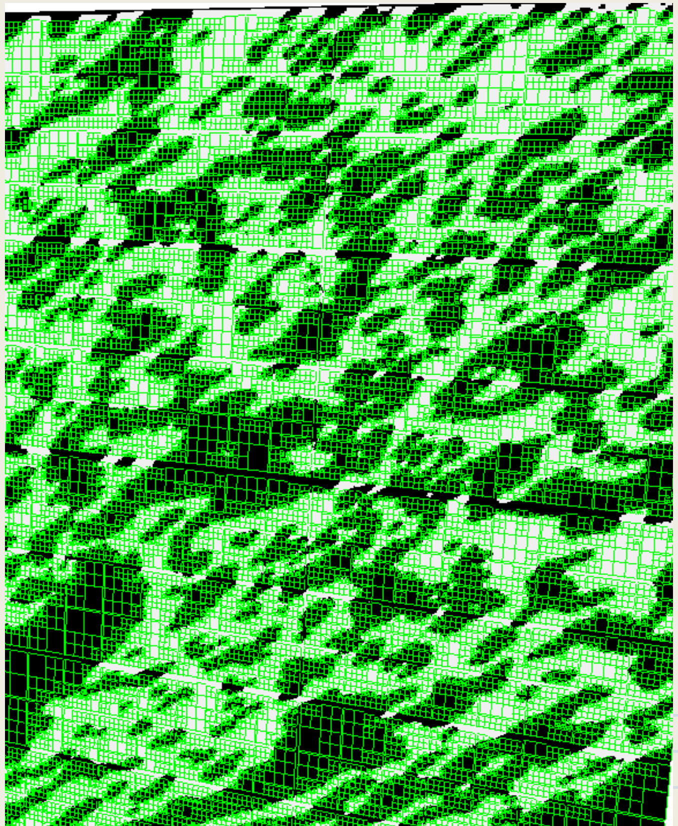


TestDoubleTorus

Results

- **Quadtree for adaptive mesh generation**

GDLImage



Thanks

Questions?