ASSIGNMENT 2

1) Lighting
It's done in fragment shader. For each vertex, normal has ben calculated which is equal to the average of normals of all the triangles that share the vertex. Normals has been calculated in main.cpp (in CreateVertexBuffer())and passed to shaders.

CPU-Based

2) Scalar Field Visualization
I) For each unnormalised vertex in the protein mesh, scalar field value is calculated using Trilinear interpolation.
Then these scalar values were normalised between 0 and 1. This normalised scalar value is then used to linearly interpolate color values to find color on each vertex. Done in main.cpp (CreateVertexBuffer()).

II) In main.cpp, slideGrid() function computes the slicing plane and scalar field restricted to that plane.
In each voxel, if voxel doesn't interesect the plane then ignore it.
Else, find the intersection points on all the 12 edges (if exists).
These intersection points form 5 basic polygons. I added all the possible traingles in the polygon to the Indexed Buffer.
Press 's' to enable/disable slicing.
Press 'z','x','c','v','b', to display different slicing planes.

III) In main.cpp, sliceModel() function computes the slice of protein mesh.
For each triangle in protein mesh, compute on which side of plane it lie. Create two buffers and keep adding triangles, normals, color accordingly.
Press 's' to enable/disbale slicing.
Press 'z','x','c','v','b', to display different slicing planes.
Press 'n' to display left slice of protein mesh and 'm' to display right slice of protein mesh.

3) Iso-contour Visualization
Traverse each triangle.
There are 3 cases:
Case 1: When all points have scalar value > iso value OR all points have scalar value < iso value, ignore it.
Case 2: When scalar value at one vertex = iso value, find other iso point if it exist else add this point twice in iso point buffer.
Case 3: When there are 2 points on triangle edges, such that they have scalar value = iso value, find them using linear interpolation and add to iso point buffer.
Press 'i' to enable/disable iso-contour;
Press Up and Down arrow keys to increase/decrease the iso value.

GPU-Based

2) Scalar Field Visualization
i) Scalar field is loaded into 3D texture.
ii) Position, Texture coordinate and normals are stored in buffer objects for each vertex.
Texture coordinate are specified in texture space. Field->vmin and field->vmax are mapped are mapped to color Red and Blue. Linear interpolation is used to create a colormap which is stored as 1D Texture.
Iii) Sampling operation is performed in Fragment shader to map texture coordinate to texture. This gives the scalar field value at the texture coordinate which is used to get color from 1D texture (colormap).

3) Iso-contour Visualization
Iso value is passed to Fragment shader as uniform variable. Sampling operation gives scalar field value at the texture coordinate. This value is then compared to iso value, if it equal to iso value then FragColor is set to Black, else it is given from colormap.
Press Up and Down arrow keys to increase/decrease the iso value.

PERFORMANCE REPORT

1) Preprocessing time

Following analysis are done with respect to 1grm.off and 1grm.pot.
CPU-Based:
i) Total time taken is 1578 msec on average.
ii) Time taken to read the model off file, pot files, trilinear interpolation, normal calculation is 351 msec.
iii) Time taken in slicing the gird and model is 1312 msec.
iv) Time taken to compute the iso contour is 10 msec on average.

GPU-Based:
i) Total time taken in preprocessing is 310 msec.

2) FPS

Following are average FPS:
CPU-Based:
i) Loading model and Scalar field visualization: 47.62
ii) Slicing: 20.01 (When the slicing function is called and calculations are performed)
iii) Iso-contour: 34.85

GPU-Based:
i) FPS for iso-contour: 40

Observation:
i) GPU-Based approach requires lesser memory than CPU-based.
ii) GPU-Based approach is faster than CPU-Based.
iii) Iso-contour is smoother in GPU-based approach.
iv) Image quality is better in GPU-based approach.