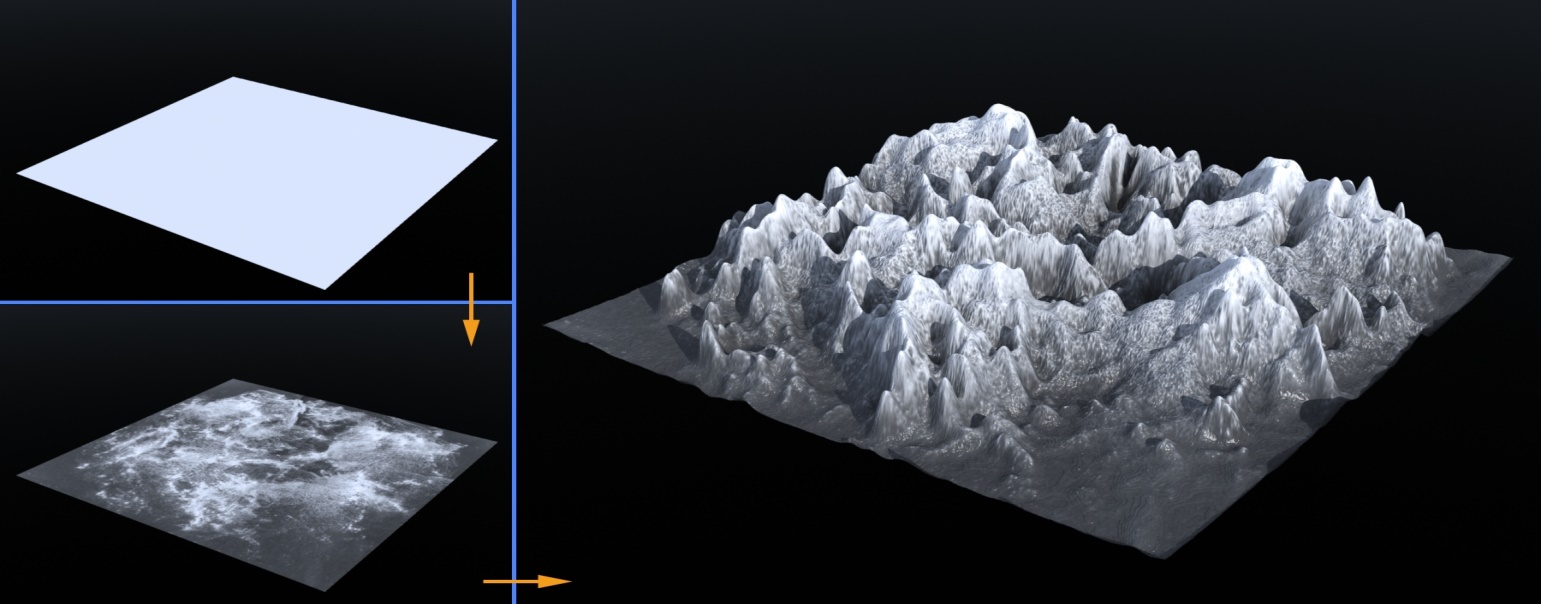
# Displacement Mapping in OpenGl

By: Rana Shujaat Ali

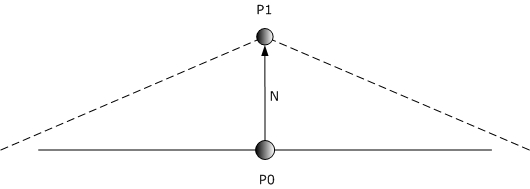
**Introduction:**

**Displacement Mapping or Vertex Displacement Mapping is a technique which allows adding details to a surface by deforming the underlying mesh.**



**Basic Idea:**

**Basic Idea of displacement mapping is to get the displacement height from displacement map and displace the original vertex by its magnitude. One can get the displacement map from any function generator or through any external graph/map. The most common approach is to use an image that can either hold RGB values or it can be gray scaled image.**

 **Reading the RGB values from an image and scaling them gives us the magnitude of the displacement.**

**Basic idea of Displacement.**

**Where P0 is the original point and the P1 is the same point after displacement.**

**The scheme for getting the height was:** **pixels[bytesPerRow \* y + 3 \* x + (2 - c)], where x, y are the dimensions of image and pixels is a char array populated by reading the binary image file by default read() function.**

**Issue of the Normal:**

**Problem with displacing the vertex is that it split down the original surface hence destroying out the surface normal from which the light was supposed to be reflected. This can be ignored if there is static lighting and we are using some light mapping, but this becomes critical if there are different light sources.**

**We have a whole surface and now if we want a normal to a surface then it’s simply the cross product of its two neighbor vectors defined by the vertices. Hence we can calculate the normal of our whole surface. Vector (vec3) data structure was same as provided in previous assignments.**

**I came up with an efficient approach not to calculate the normal on runtime but during the loading of image I calculated all the normal in a two dimensional array and assigned them while displacing the vertices.**

**Refinement of surface (Normal Refinement):**

**After displacing and assigning the normal the displacement map was successfully generated but on going closer to the surface it was observed that triangular strips were visible.**

**One resolution could be adding more details per unit area by increasing the pixel of our map but it will lead to more computation which would slow down our program. My approach was to add the scaled down contribution of the neighbors of the neighbors by 0.5 amounts which successfully removed the problem of visible triangles on going closer to the surface.**

**The basic command for displacement map was:**

* **glNormal3f(); used to assign the normal to surface**
* **glVertex3f(); used to draw the vertex**

**Texture Mapping:**

**Texture mapping is simply taking a texture/image and scaling it over the range of (0 🡪 1) and assign the corresponding texture to our surface. In openGl if the surface is hardcoded texture can be simply assigned, but in dynamically generated surface we need one extra step to count the position of the corresponding texture position relative to our target surface.**

**In my project the mesh was created by a open GL triangular strip so the s and t were calculated by:**

* **t0 = (float)z / (\_terrain->length() - 1); // t0 in [0,1]**
* **t1 = (float)(z + 1) / (\_terrain->length() - 1); // t1 in [0,1]**
* **s = (float)x / (\_terrain->length() - 1);**

**Where x and z are the length and width iterator.**

**The command used to assign the texture using there s&t values was:**

* **glTexCoord2f**

**Creating a Landscape Scene:**

**There was a mountainous surface which was generated by displacement mapping, and surrounded by a cylinder creating a sense of surrounding, and the water at some level down the mountain to add more realistic effect to our scene.**

**Although the project was displacement mapping but still the scene was not visually realistic due to the limitation of openGl, it can be implemented better in Shedders.**

**User Interface:**

**There was complete freedom for the user to go inside and outside the scene to analyze how a 3D scene can be created. Both mouse and keyboard functions were available as:**

* **Camera Rotation: user can rotate the camera in all three dimension using keyboard interface.**
* **Zoom: user can go inside and outside the scene very easily using w and a key from the keyboard.**
* **Object translation: user can move the object up, down, left, and right using keyboard.**
* **Object Rotation: user can rotate the object using mouse by clicking on the scene and dragging the pointer to anywhere.**

**Auto Motion:**

**glutTimerFunc() was implemented to set auto rotation of the scene.**

**Fun with Displacement Mapping:**

* **I displaced facial feature(nose, lips, checks) using openGL**
* **Created a new face using only displacement map.**

**Note: I had all the data in my previous report, but I have lost my laptop and all data was in that so I cannot add the picture now of what I did in past.**