**3D Computer Graphics**

**Dr. Raúl V. Ramírez**

**OpenGL Project: Scientific Visualization**

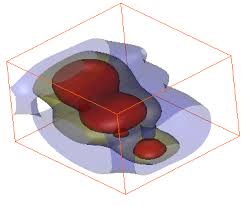
Using OpenGL and GLUT develop a visualization (mainly scientific visualization) application. You have been assigned one of the following projects:

|  |
| --- |
| Heightfield (Terrain) Generator |
| Isosurface Visualization (Marching Cubes Algorithm) |
| Volumetric Visualization (transparency map applied to 3D Volume) |
| 3D Fractal (MandelBulb, Mandebox, Julia bulb |
| Rotated Julia Profile |
| 3D Tree Generator |
| 3D Rotating Hierarchy Chart (Cone Tree) |
| Infinite 3D Rooms Storage |
| Infinite Menu (Perspective) Wall |
| 3D Connected Chart (for social networks) |
| 3D Bar Charts (over image) |

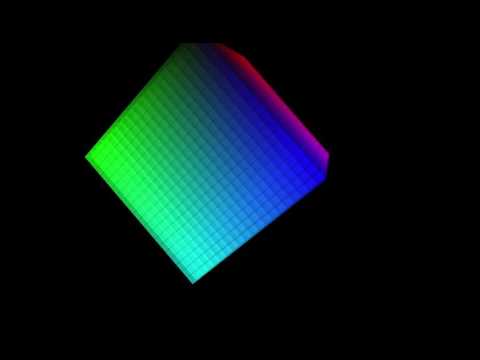
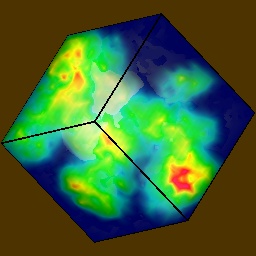
**Heightfield (Terrain) Generator**: Recreate your fractal terrain generator. Display the terrain in OpenGL using polygon primitives (such as GL\_QUAD\_STRIP). Include options to change the resolution of the lattice and roughness (fractal dimension) of the terrain. You must include options to give initial values to certain points (no necessarily in the corners) to control de general shape of the terrain.



**Isosurface Visualization**. Given a volume in (that is a 3D matrix) in which we have three dimensions (x,y,z)->(I,jk), the scalar contained Q could be a physical attribute such as temperature, density, stress, etc. Use the marching cube algorithm to extract a surface which connects points with similar Q attribute



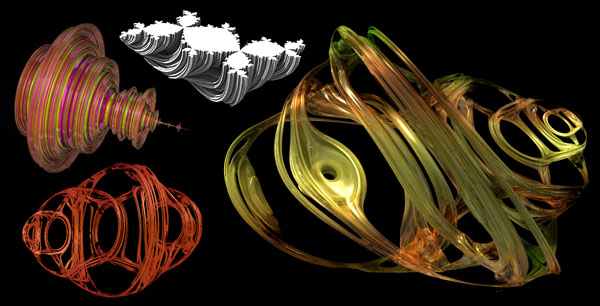
**Volumetric Visualization**. Given a volume in (that is a 3D matrix) in which we have three dimensions (x,y,z)- contained in a matrix with indexes (i,j,k), the scalar contained Q could be a physical attribute such as temperature, density, stress, etc. Divide the surface into voxels (small cubes) and determine a colour and transparency. The colour depends on the attribute Q. The transparency depends on the distance to the camera. The closer the voxel, the more transparent it is and the further away, the denser it is.

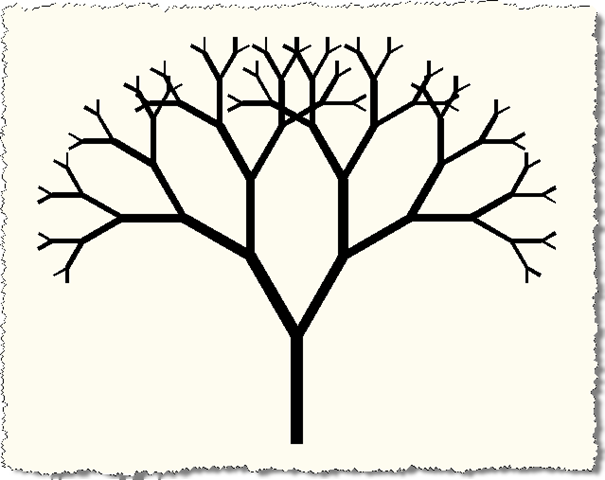
**3D Fractal: MandelBulb, Mandebox, Julia bulb**. With a procedure similar to the Mandelbrot and Julia set of equation z=z2+c with imaginary numbers, use the same equation with hyper complex numbers and find the Mandelbrot isosurface in a three dimensional volume. Work with an isosurface team to generate the code for the isosurface.



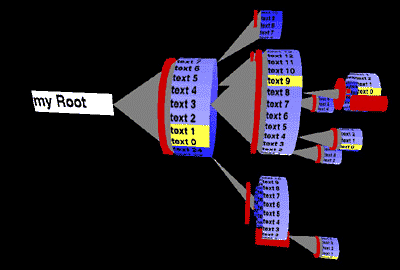
**Rotated Julia Profile**. Remake your Julia code using OpenGL. Create a connected isoline (a line with the same number of iterations from z=z2+c). Then rotate the line creating polygons (such as GL\_QUAD\_STRIP). Colour your polygons.

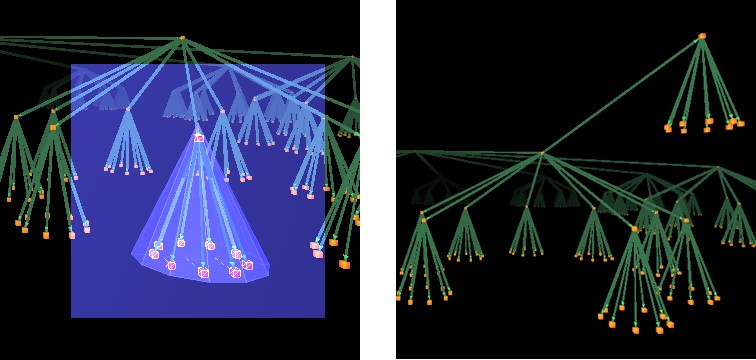


**3D Tree Generator**. Research and algorithm to generate trees. Generate several kinds of trees with different values of branching and leaves.

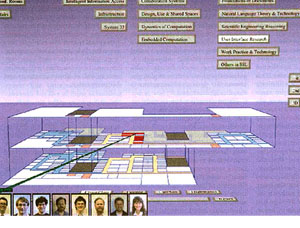


**3D Rotating Hierarchy Chart**. Implement a 3D version of a hierarchy chart resembling a baby’s hanging decoration. It could possibly store infinite hierarchies. In order to visualize different parts of the charts it must be able to zoom certain areas and rotate. The branches should also move infinite band style (like the doors in Monsters Inc.) to visualize all the elements in a category.

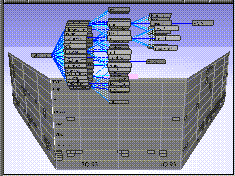
 



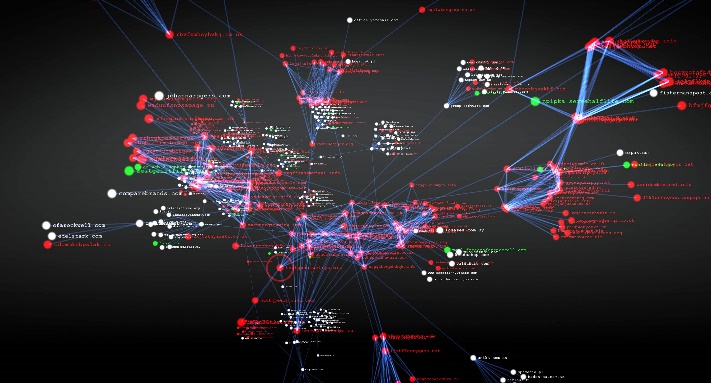
**Infinite 3D Rooms Storage**. Create a world of in which space is divided in square rooms. Labels inside the rooms associate data items (use anything). Inside each room there can be more diminutive rooms (labelled) within which more diminutive rooms exist.

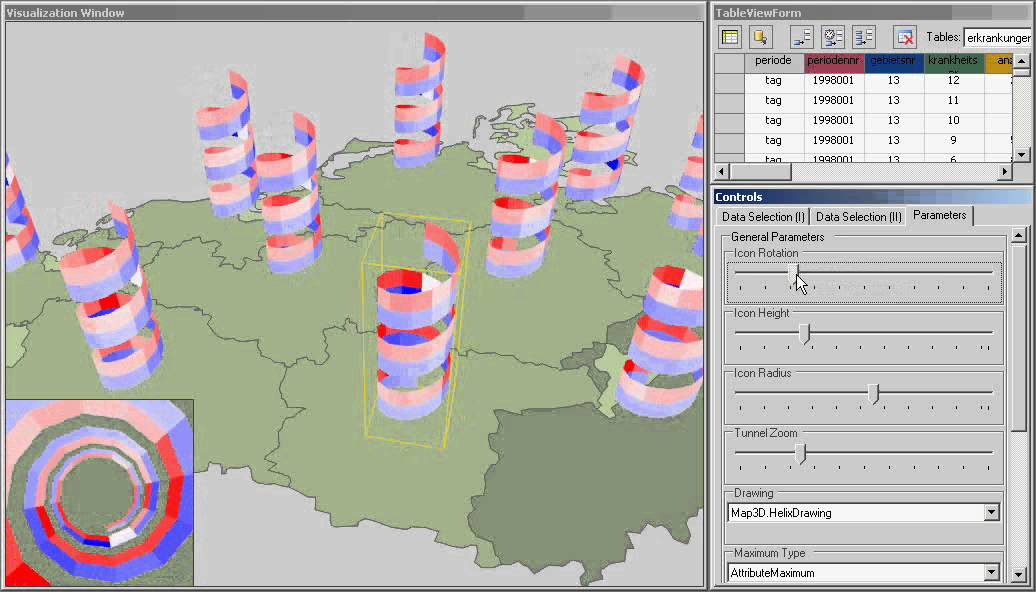
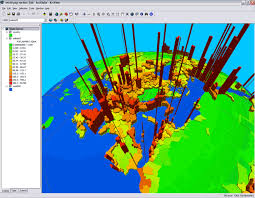
**Infinite Menu (Perspective) Wall**. Build a rotating wall that will create an infinite menu. ¿Can you create submenus? Use colour to differentiate sections of options.

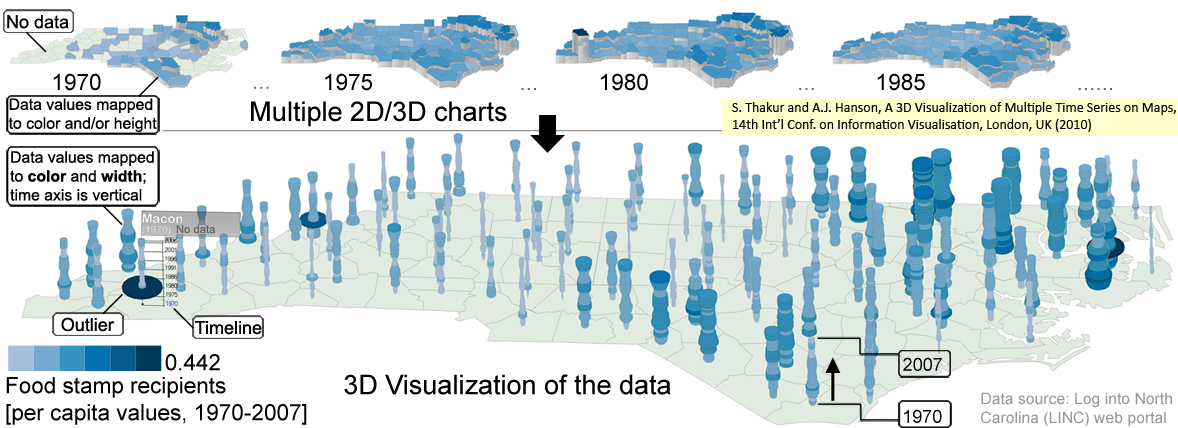
 

**3D Connected Chart (for Social Networks)**. Create a visualization tool that is designed to show connections between data.

**3D Bar Charts over Image**. Create a visualization tool that is designed to compare geographically distributed scalar data.



*For all projects use colouring schemes to shade the polygons and create a simple GUI to move the camera, and rotate the object. Develop code to store objects on disk and import it to an OpenGL program (maybe using the OBJ file format).*

**Rubric**

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Beginner (50%) | Competent (75%) | Proficient (100%) |
| Modelling (30%) | Understanding of the mathematical concept is proved (for example, with a simple non programmed example) | Code transfers model from mathematics to OpenGL primitives (2D or 3D) | Parameters to generate the object can be changed and the object appearance changes. GUI to change parameters |
| Rendering (25%) | A basic visualization of the object is achieved | Objects can be visualized from different angles. 3D components are used | The number of polygons used to generate the object can be changed, increasing definition. Components or the whole move and/or rotate independently using a GUI |
| Colouring, Illumination, Cameras (25%) | Cameras and lights are applied | Cameras can be moved. Polygons are coloured according a described scheme | A simple GUI is developed to navigate the camera for better visualization |
| Storage (20%) | Some form of storage is achieved (at least configuration parameters). Session can be repeated. | Objects can be exported as polygons | OBJ storage is achieved. Objects can be imported to Blender or POV |