CTDB 3D Viewer/Editor
Richard Pater, Matthew Hsieh
Faculty Advisor: Barry Silverman

Abstract:

• Build an intuitive and user friendly Compact Terrain Database (CTDB) 3D viewer and editor to allow users with no experience with our software or the CTDB format to quickly and easily change or add markups to a CTDB tile.

CTDB:

• CTDB format widely used in US military training simulations.
• Efficiently stores large amounts of complex terrain information in tiles.
• Terrain features are stored as a collection of vertices with other information depending on the specific type of feature: building, road, river, tree, political boundary, etc.
• Provides a search space query interface to return objects of interest inside a box specified in geocentric coordinates.

3D Graphics:

Development Tools

• We developed our software in C and C++ and used OpenGL, a graphics standard developed by SGI, to draw the features queried from the CTDB tile. We tried to make our viewer as realistic and visually appealing as possible. We will highlight some of the features of our viewer.

Fly-Through Camera

• In a 3 dimensional space, a camera can move along three angles. To simulate a camera as a person, we restrict our camera to rotations only around the x-axis (pitch) and y-axis (yaw or heading). The camera can also be translated relative to its own local coordinate system (forwards, backwards, upwards, downwards and side to side). By instantiating each structure with a unique color id, we would be able to build a 3D model of each structure. We also mapped out linear features such as roads and rivers.

Drawing Features

• From the CTDB, we extract the local features to draw to the screen. We drew linear and structural features. Buildings are stored as a set of points in a Cartesian coordinate system. Using these points, we were able to build a 3D model of each structure. We also mapped out linear features such as roads and rivers.

Textures & Anti-Aliasing

• We mapped photorealistic images to the sides of each building, to the rooftops, and to the linear features to make them look more realistic. Besides using built-in anti-aliasing algorithms to smooth lines, we also used mipmaps to increase rendering speeds and reduce sampling artifacts for our textures.

Fog & Lighting

• Even though there are built-in functions for lighting in OpenGL, we need to calculate the normal vector of each surface in order for lighting to work. From linear algebra, to find an orthogonal vector, we take the cross product of two vectors in the plane. We used a fog effect to act as the boundary to the search space.

Fog Enabled

Sky Box Background

• Z-writing is disabled to superimpose a bounding cube around the camera. Texture is applied to each side of the cube to create the illusion of an infinite horizon.

Fog Disabled

Conclusion:

• Working with the camera, adding textures, colors, and selection in OpenGL were all new concepts that we researched and implemented into our viewer/editor.
• The CTDB format was also new to us and extremely poorly documented. Working with it was a significant challenge.
• Our software is an intuitive, easy-to-use, free alternative to more expensive, bloated, commercial products.

Editing and XML-RPC Interface:

Selection Using the Back Buffer

• To edit a structure in the viewer, we needed a way to “click” on a building. By instantiating each structure with a unique color id, through drawing in the back buffer, we could compare the color of the pixel under the mouse pointer to the color ID of a structure to find a match.

Advantages:

• Easy to implement.
• Selection of top-most pixel fits with our selection process.

Disadvantages:

• Slower when many objects on screen
• Causes a redraw in the back buffer every time the mouse is clicked.