o

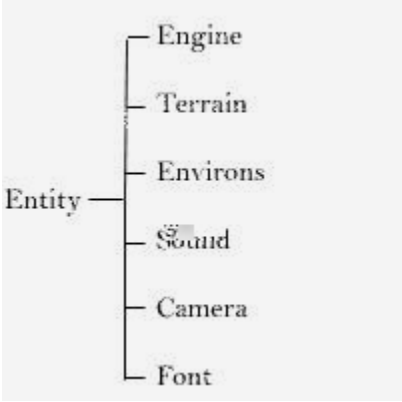
y, and z locations are now necessary because the vertices are no longer guaranteed to be a constant distance apart. A problem existing within pre-rendering reduction is that these data points are physically lost within the data model, so that for applications demanding visual representation based upon constant separation, such as charting or measurements of coordinates, would be unable to easily find a data-set readily available without costly interpolation. Further the data reduction may not occur in more trafficked areas, as such is within land simulation in which travel upon water-ways is not reasonable, and therefore rarely falls into the viewable area. [Savch00]

A model and management system is necessary therefore to both manage the data, in which data reduction can occur in the correct locations, and at the same time offers reasonable memory requirements and expandability of detail and scope. In order to manage a model for terrain rendering, it is necessary to create a scene graph model in which an order of data can be efficiently managed within the program. A scene graph is the hierarchical structure by which an entire tree- representing the virtual world- is organized for efficiency and easy management. This model allows not only the efficient rendering of terrain but also the management of data and objects within the world defined by our program. [Eberl99]

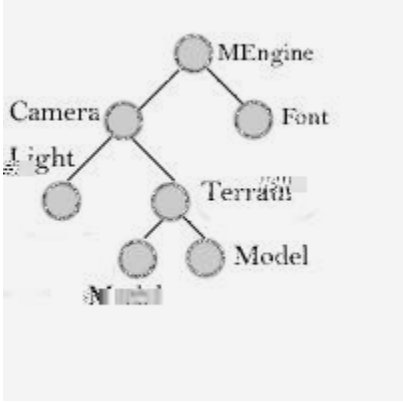
The scene graph within this paper is derived from the class , from which all other classes are derived from within the scene graph. The object at its simplest level is a node object comprised of a parent, a child, and two

siblings who share the same parent. This structure allows for a tree consisting of any object derived from the class, and therefore allows the creation of tree

maintain synchronized decisions among objects as well as output debugging as well as user-useful information to the screen.




*Super Class Structure*




*Scene Graph Tree*

The hierarchical design of the tree also allows for as many children as needed to be tested for visibility through a bounding sphere test used upon the parent

renderer and the objects contained within. It is also possible to include just empty bounding nodes as a binary or octree structure in order to partition the space.

The model used for the terrain rendering was based upon the principle of "Level of Detail" or LOD. The concept of a LOD algorithm is to create a model by which the sharpest amount of detail is reserved solely for the closest terrain to the camera view. The level of detail, or amount of vertices used per specific region, is inversely proportional to the distance the section of terrain is from the camera view. This level of detail, in most models, is bound by a lower limit of detail in which one primitive is used to draw the entire terrain, to the upper limit in which every vertex within the data set is being represented by a point on the map [Linds96]. A LOD algorithm therefore offers a distinct advantage over other data algorithms, in that the algorithm allows the data which is immediately being viewed in the near vicinity to be rendered in its fullest detail, while saving computation and rendering time by simplifying the data set in the distance. The reason why this concept is plausible is because any data rendered at a distance, when transformed through the models perspective matrix, will resolve to a much less detailed image. Objects therefore at a distance collapse there own detail, so it is wasteful for the graphics model to attempt to draw discarded vertices. This model however, cannot be pre-calculated as could a pre-rendering vertex reduction algorithm. LOD algorithms must be dynamic, and the computation time to create this revised structure of the data set must still maintain a desirable

frame rate. The LOD algorithm must also be scalable, in both detail and scope, and offer a reasonable requirement of memory. The final requirement when approaching an LOD algorithm is that it must be cost-effective to be able to integrate such features as lighting normals, primitive color blending, and texturing.

In exploring the world of LOD models, the search for such a model which coincided with the demands listed above was found in part within a quad-tree LOD algorithm originally explored by Stefan Rottger.[Rottg98] Rottger's algorithm was chosen as a base for a final LOD model because of its simplicity in design, and its ability to be easily integrated with other features of the scene graph. The description that will follow in this paper will pay close attention to the details of the author's modified quad tree algorithm, and will note those feature which were originally a part of Rottger's design, as well as those sections added or modified.

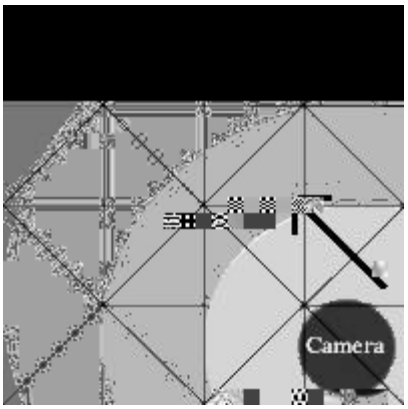
The concept of the quad-tree algorithm is the representation of a height field through series of recursive quads, for which the root quad encompasses the four corners of the height field as well as the center of the height field. By this description alone, a quad-tree then requires a height field of equal height and width, and in which the length of a side is an odd integer. Further more the recursive nature of the quad-tree, in which a quad is broken into four equal area quads, requires that the width of each quad be a power of two. Therefore the height field is bound to sizes 2^n + 1.  This ensures that a height field can be

recursively split using quads until each vertex is covered by the side or center of a quad. This model therefore allows for the total representation of a data set if each of these highest level quads are rendered. The most efficient choice for rendering these quads however is through the use of triangle fans. The triangle fan model with a quad has a center point located at the center vertex of the quad, and then up to 8 vertices to which it draws. Within the quad-tree model, the highest level of detail therefore is a quad that is 3 X 3 with the center point located at the location (2, 2). The triangle fan model therefore can, at highest resolution represent every vertex in the data-set.

The essential concept of the quad-tree LOD algorithm though is the function which decides when a quad requires more detail, and in what way does the quad tree split to give that new detail. The method by which Rottger's model determines the need for further detail is begun through the expression:

$$L / Q < C$$

In the above expression *L* is the distance from the camera view to the center of the quad; Q is the quad width, and C is some constant intended to control the amount of detail represented in the scene. As the value of C increases, the level of detail increases8 (l) - 11 (c) (e) 8(s) shil r het ghril h ohhesshashar h8 (l) - (e) 8 li (il ) (s) shrease(o)




*Rings Of Detail*

The process within Rottger's model as well as other designs is based upon the principle of rendering the absolute minimum vertices necessary while maintaining a certain resolution and frame rate. This has led the Rottger model to follow the principle of testing for a quad's necessity to split into further detail, and then to test within each child quad whether the split within this quadrant is necessary. This process allows only some of the quads within the parent quad to be split with more detail, which appears at its creation to offer a distinct savings in rendering, but in truth it introduces many additional disadvantages. One disadvantage is that extra distance calculations must occur when the quad is split to decide if the new child quads are within the area deemed necessary for further detail. This forces the child node to decide whether it should split further, represent its new vertices without split, or simply represent the vertices visible by the parent. Further, upon a split, by testing each child to see if it should reflect the new vertices or if it should maintain the detail of the parent, it removes the

children's ability to know the detail level of each other, and therefore put that information into saving computation time.

algorithm.  To remedy this problem, Geo-Morphing manages the newly introduced by ensuring that there initial introduction value will not vary from the average of the two points across the line which the new point is introduced. The split function supports this easily by continually producing values within the range of 1.0 - 2.0. By subtracting from these values by 1.0, a weight average can be used on the new point so that at initial introduction, where the split function S is equal to 2.0, the weighted average of point pNew across parent

The use of Geo-Morphing however introduces a problem in the interpolation of new points of detail. If two quads are both rendering points at the same level of detail, there will exist a small difference in the values of their respective split functions, so that the interpolated values calculated by each quad for the shared points will be slightly different. This causes cracks in the terrain, which are considered unacceptable as a feature of an LOD algorithm. This causes a revision of both the LOD algorithm implemented within this paper as well as most other LOD algorithms. In order to solve this problem, two passes

B. Include North, West, East, and South data points if the respective neighbor is of an equal or greater level of detail.

C. For each side, if the neighbor is of equal level of detail, use the greater of the two split function values to ensure continuity.

D. For each side, if the neighbor is of a lesser level of detail, ensure that corner point of the node which is the middle point of the neighbor node is computed by using the split function of the neighbor

The necessity of knowing the level of detail of a node's neighbor is therefore critical during the rendering process. This necessity is simplified and optimized by altering Rottger's algorithm by splitting a quad into 4 individual quads when the split value is dropped below the threshold. So within the model developed within this paper, a node always has critical information always about three of its neighbors because it knows that its siblings were split as well. This becomes important when features such as texturing, primitive coloring, and lighting becomes important facets in the model design.

One factor introduced into the concept of LOD algorithm for which Geo-Morphing cannot solve is the case of extreme changes in height values within the height field. This can observed in the case where a quad's split value has recently fallen under the split value threshold, and in which the new data points do not yet change the shape of the surface. The problem however is that for larger and larger differences between the averaged value of the parent nodes and the final

value for the new node, the greater the changes will appear for each step closer. So while Geo-morphing works well for introducing slight variations into the data field, it cannot handle extreme aberrations from the normal without compromising the guarantee of a fully detailed and realistic view.

*CenterNodeError*= ((

naturally disallow further splitting of the quad because of the wastefullness of further calculations for only minor detail. The split function is therefor modified to incorporate error through the function:

*Split Value = Distance / (Quad Width * Maximum Detail * Node Error)*

Within the LOD model described in this paper, the error value has been minimized because of its lack of accuracy to described the uniqueness of the new data point introduced. This is realized because of the incorporation of both primitive coloring as well as lighting effects. Therefore while the previous computation to find the error involved in introducing a new data point provided an optimized interpretation of height data, it does not take into consideration the possible introduction of lighting and material effects or primitive coloring that would have provided further detail. Due to this lack of incorporation, it seems reasonable to further revise the split function so that it only offers more detail and never attempts to remove detail from a quad. This obviously means a higher number of triangle fans being drawn, but is a necessary step so that important detail in lighting and colors are not removed permanently. This newly revised function can be expressr

The use of textured primitives within the LOD algorithm can only supply a minimal amount of variation across the terrain. Even the use of multiple textures offers only a small rm

points, so that their new detail is slowly introduced into the scene. Lighting however suffers from the problem that the vertex normals in the field computed, as being the average of the face normals of the 8 triangles sharing the vertex, will incorrectly represent the lighting normals of a single vertex, based upon the

# References


[Linds96] Lindstrom et al., Real- Time Continuous Level of Detail Rendering of Height Fields, *Proceedings of SIGGRAPH, 1996*

.

[Rottg98] Rottger et al., Real-Time Generation of Continuous Levels of Detail for Height Fields, *Proceedings of SIGGRAPH, 1998.*

[Eberl99] Eberly, D. *3D Game Engine Design*. *San Diego:* Academic Press, 1999.

[Savch00] Savchenko, S. *3D Graphics Programming Games and Beyond. Indianapolis:* Sams Publishing, 2000.

[Morle00] Morley, M. *Frustum Culling in OpenGL*. Http://www.markmorley.com/opengl/frustumculling.html