

Visual Navigation of Large Environments Using Textured Clusters

Paulo W. C. Maciel*

Peter Shirley†

Abstract

A visual navigation system is described which uses texture mapped primitives to represent clusters of objects to maintain high and approximately constant frame rates. In cases where there are more unoccluded primitives inside the viewing frustum than can be drawn in real-time on the workstation, this system ensures that each visible object, or a cluster that includes it, is drawn in each frame. The system also supports the use of traditional “level-of-detail” representations for individual objects, and supports the automatic generation of a certain type of level-of-detail for objects and clusters of objects. The system supports the concept of choosing a representation from among those associated with an object that accounts for the direction from which the object is viewed. The system as a whole can be viewed as a generalization of the level-of-detail concept, where the entire scene is stored as a hierarchy of levels-of-detail that is traversed top-down to find a good representation for a given viewpoint. This system does not assume that visibility information can be extracted from the model and is thus especially suited for outdoor environments.

1 Introduction

This paper describes a new approach to the “walkthrough” problem, where a viewer interactively moves through a static scene database at high and approximately constant frame rates.

Traditional approaches to this problem use a hardware graphics pipeline and attempt to minimize the number of polygons sent to the system. This minimization is achieved both by culling the entire model or the part of it that is potentially visible in the next few frames against the viewing frustum and using geometrically coarse representations (levels of detail, or LODs) of individual objects.

The approach described in this paper attempts to extend the domain of traditional approaches by assuming that sets of potentially visible objects cannot easily be computed and at any given frame the visible scene can contain more graphics primitives than state-of-the-art hardware can render in real-time even if the lowest detail LODs are used for every object.

The basic strategy underlying the system described in this paper is the use of *impostors*. An impostor is an entity that is

faster to draw than the *true object*, but retains the important visual characteristics of the true object. Traditional LODs are a particular application of impostors.

The key issue is how to decide which impostors to render to maximize the quality of the displayed image without exceeding the available user-specified frame time. The best approach so far to solve this problem attempts to predict the complexity of the scene at the current frame and selects impostors accordingly and is described by Funkhouser and Sequin [3].

The system described in this paper can be viewed as an extension of Funkhouser and Sequin’s system with the following new properties:

- The entire database is a single hierarchy which contains drawable impostors (including LODs) for objects as well as clusters of objects. This is a global generalization of the LOD concept to the entire model.
- The system uses the graphics hardware to automatically create this hierarchy, generate impostors, compute their rendering cost, and compute a static portion of their benefit according to the direction from which they are viewed.

In Section 2 we revisit the work done by Funkhouser and Sequin, briefly presenting the main components of their system and showing why it doesn’t scale well to arbitrary environments. In Section 3 we discuss how to extend the benefit concept to account for cluster primitives and view-dependent LODs. In Section 4 we show how the selection process can be formulated as an NP-complete tree traversal problem, and present a heuristic solution that generates a complete, if non-optimal, representation of the model for display. In Section 5 we discuss our implementation. Finally, we discuss the limitations of the system in Section 6 and the conclusions in Section 7.

2 Predictive Approach Revisited

The predictive approach described by Funkhouser and Sequin assumes that the system runs on a machine in which the rendering cost of each object in the model can be estimated. This rendering cost is estimated by empirically obtaining performance parameters of the machine and using these parameters in a simple formula.

Since effective walkthrough systems need to achieve a balance between interactivity and visual quality, they use a benefit heuristic to decide about the amount of contribution to the overall scene caused by rendering an object with a particular accuracy. This heuristic takes into consideration factors associated to a representation of the object such as image-space size of object, focus, speed relative to view point, semantics, accuracy of a LOD, and hysteresis with respect to switching between different LODs.

Objects are selected to render using an incremental optimization algorithm that prioritizes the selection of objects

*Department of Computer Science, Lindley Hall, Indiana University, Bloomington, Indiana, pmaciel@cs.indiana.edu

†Program of Computer Graphics, Cornell University, Ithaca, New York, shirley@graphics.cornell.edu

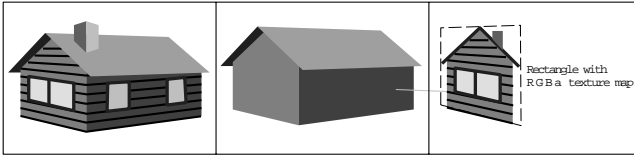


Figure 1: Three representations for a house. The left two are view independent LODs while the right one is a view dependent texture map.

with high benefit/cost value to render until the user-specified frame time is reached. The result is that low-valued visible objects may not be displayed. In environments where too many visible primitives are present at a given point in the simulation, this can result in large “blank” spots on the scene which would cause a distracting effect.

To reduce the number of primitives rendered at each frame, visibility information from a pre-processing phase is used to cull objects that are certainly blocked from view by partitions. This approach works well for models that can be subdivided into cells containing open spaces (such as doors and windows) through which visibility can be determined. In an outdoor environment such cells and portals are not easily identifiable making the pre-processing of such an environment to extract visibility a hard problem.

Our system is also a predictive system and assumes that it will run on a multiprocessor machine with texture mapping capability. We allow for situations where more unoccluded primitives can occur inside the viewing frustum than can be rendered in real-time and do not assume that visibility information can be extracted from the model. This last feature, makes the system suitable for navigation of large outdoor environments.

3 Benefit Calculation

Visual navigation systems use different representations (LODs) of an object to improve the performance of the simulation. As explained in the previous section, each LOD makes a contribution to the quality of the simulation that can be estimated by a benefit heuristic.

In computing these benefits we face two interesting issues: how to compute the benefit of individual representations of objects taking into account their view angle dependent nature (e.g. a roadside billboard has a low benefit when seen from the side), and how a group of objects is perceived (its “semantics”).

3.1 Benefit of Objects

In our approach, an object can have associated with it not only the conventional LODs but also any other drawable representation that resembles the object from given viewpoints. Consider the possible representations we can use to render a house as in Figure 1. In this picture, the first (leftmost) of these representations is the house object at full detail, the second is a low LOD representation and the third is just a single polygon with a texture map representation of the front of the house.

We classify the third representation as *view dependent* and the first two as *view independent* meaning that the *view dependent* would only be considered for a subset of all possible viewing directions, while the *view independent* LODs would be considered for all viewing angles.

We have divided the contribution to the simulation of rendering a given representation associated with an object in two parts. One that is intrinsic to the object, the object’s benefit, and one that is intrinsic to a representation of the object, the accuracy with which it represents the full detail object.

Intrinsic to an object are factors such as its image-space size (since large objects on the screen seem to contribute more than smaller ones), its distance to the line of sight (since assuming that the eye is looking to the center of the screen, objects near the center of view are better resolved by our visual system than objects in the periphery of view), relative speed of the object to the viewpoint, and semantics (role of the object in the simulation). Our per-object benefit is computed as a weighted average of all these factors and it is used to guide the selection of representations to render in Section 4. The weights are empirically determined and can be changed for each run of the simulation.

Intrinsic to a representation of an object is its accuracy with respect to the full detail object, that is, how similar a given representation is to the actual object for a particular view angle.

Note that while the benefit of an object (except for its semantic) can only be determined in real-time and therefore is inherently dynamic, the accuracy of a representation is inherently static and can be determined prior to the walk-through of the model, as described in Section 3.2.

3.2 View Angle Dependent Benefit Calculation

Consider again the house representations in Figure 1. The left most of these representations should have the highest benefit regardless of view angle but we might not want to render it since it is also the most expensive to render. The benefit that should be assigned to the other two will depend upon the user’s view angle (for the texture maps) and view distance (for the low LOD).

A way of incorporating view dependency information into the benefit heuristic is to measure the accuracy of each of the object’s representations according to each viewing direction possible.

Since the space of possible viewpoints and viewing directions is infinite, we approximate it by discretizing this space into a finite set of viewing directions, and assuming that the view distance is infinite (we use an orthographic projection). This seems reasonable because we do not expect to use coarse LODs when the view distance is small. To tabulate directional benefits, we sample the hemisphere of directions (Figure 2) and calculate an image of the object and impostor at each sample point.

The number and location of these samples will depend on the number of representations that the object has and the possible viewpoints during the walkthrough. For instance, in the case of the 2D house impostor in Figure 1, we will never use it unless we are roughly in front of the house, so only directions around the line perpendicular to the 2D image are sampled.

We sample each of the viewing directions and measure the accuracy of each representation and construct a table that has one entry for each pair (representation, viewing direction). Each of these entries contains a similarity value (accuracy) of the representation measured with respect to the full detail object for the particular viewing direction. During the walkthrough, the accuracy of a given representation and viewing direction can be obtained by accessing this table.

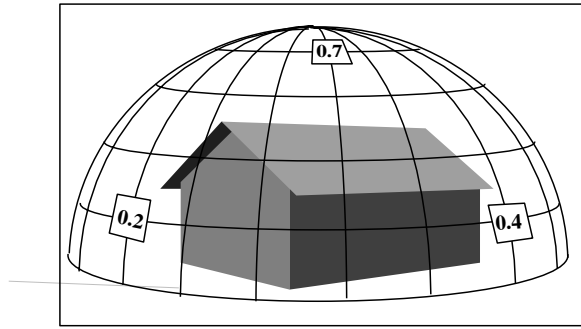


Figure 2: Discretizing the space of viewpoints around an object. Replication accuracies are shown at three of the view angles. The low LOD house looks the “best” from the top.

Ideally the accuracy of an image with respect to the ideal image should be obtained by a perceptual comparison of the two images but since we are in search of automatic ways to determine similarity we resort to computational techniques. In our implementation we use simple image processing techniques to get this similarity value.

We avoid a simple pixel-by-pixel comparison of the two images, since slight differences on the impostor’s image would cause two very similar images to have a similarity close to zero. Because the achromatic channel of vision is the most important for shape recognition, we start by obtaining a gray scale version of the two images by simply averaging the rgb components at each pixel. Since edges are features on an image that are readily identified by the human visual system, an edge operator is applied to the images. The images are convolved with a 5x5 Laplacian operator and its zero crossings are computed. A subsequent blurring step increases the chances of matching of the two images, which we then compare pixel-by-pixel.

This image comparison method is far too simple to mimic human image processing, but does serve as a placeholder in our system that can be replaced later with a module that performs better by using segmentation and high level processing.

3.3 Benefit of Clusters

This section is meant to highlight that much more research needs to be done on how benefit heuristics can draw on perceptual behavior. We argue that a per-object benefit heuristic does not address how humans perceive a collection of objects when seen as a whole. Briefly, if two objects α and β are represented by an impostor γ and have benefits B_α and B_β what should the benefit B_γ of γ be?. B_γ is not simply the sum of B_α and B_β since α and β when viewed as a group might give a different contribution (meaning) to the simulation than the objects alone would, that is, the benefit of all the objects in a scene does not translate into a perceptual measure for the entire scene.

A practical example would be to consider a walkthrough of a battle field containing many soldiers and guns. In this situation the benefit of a gun and a soldier do not add up to form the benefit of a soldier holding a gun, particularly if the soldier is pointing the gun toward the user of the system.

Therefore we conclude that to determine the benefit of an object in some cases is undecidable without knowing what surrounds it. As pointed out by Gestalt Psychologists [7],

the meaning conveyed by an object may be more than merely the “addition” of the meanings conveyed by each one of the objects alone, that is, the whole conveys more information than the sum of its parts.

While realizing that it is extremely difficult to account for how objects interact in a scene we still use a per-object benefit heuristic knowing that it may not be suitable for some groupings of objects.

4 Navigation System Design

The ultimate goal of this work is to design a visual navigation system that is able to keep a user-specified uniform frame rate when displaying a large environment.

We begin by presenting a general framework for visual navigation systems. We then formalize the navigation problem as an NP-complete tree traversal problem and explain in detail the design of our system.

4.1 Framework for Visual Navigation Systems

In many cases, conventional LODs are either not readily available, are expensive, or are time consuming to generate. Since these LODs are simply representations of the “true” objects they do not necessarily need to be versions of the same object with fewer geometric primitives (or drawn with a less accurate rendering algorithm such as flat shading instead of Gouraud shading) but rather representations that can be drawn on the computer screen in less time than the true object and provide the simulation with a feel similar to that obtained by using the full detail object.

With this in mind, our design allows an object to be associated to many different representations that resembles it, possibly from different view angles.

4.1.1 Object-Oriented Design

The main abstraction for a single object, is the “conceptual object” abstraction. It corresponds to any object in the model that has a well defined meaning in the simulation, such as, a car or a building. Associated with the conceptual object is a set of “drawable representations”, which have characteristics similar to the actual object it represents.

The “drawable representation” abstraction represents a variety of hardware drawable representation or impostors for a given conceptual object. The abstractions for drawables encapsulate hardware defined primitives such as meshes of triangles, splines, list of polygons, etc., as well as the impostor representations presented in Section 4.1.2. This encapsulation of both hardware primitives and impostors allows the design of very efficient rendering routines that extract the most performance of the graphics subsystem. Other impostor abstractions may be added to this design as deemed necessary to solve a particular problem or to add a particular feature to the walk-through program.

The conceptual object’s interface is defined by virtual functions to compute the object’s benefit, visibility, and a “draw” function that is redefined for each specific drawable representation. The drawable representation’s interface is defined by functions to compute the drawable’s rendering cost, accuracy, and by customized “draw” functions.

4.1.2 Types of Impostors

As mentioned in Section 3.1, we allow an object to be represented by both view dependent and view independent im-

postors.

Examples of *view dependent* impostors are:

- A texture map that is pasted onto the appropriate face of an object’s bounding box. This texture map is called a textured cluster when it corresponds to an image of a group of objects.
- Another view dependent texture map is also known as billboard in [6] and is obtained in the same way as texture maps. A billboard is centered at an object’s center and made to rotate in such a way that it always face the observer. Since one billboard is computed for each face of the object’s bounding box as the observer moves around the object a different billboard is selected to display according to the viewpoint. This impostor is useful to represent objects that are approximately rotationally symmetric such as pine trees.
- Another variant of the texture map described above is a pseudo-texture map¹. A pseudo-texture map is a triangular mesh (or a quadrilateral strip) onto which a texture map is pasted in such a way that each pixel in the image is associated to a pair of triangles (or quadrilateral) in the mesh.

Examples of *view independent* impostors are:

- The conventional levels-of-detail, i.e., geometrically coarse versions of a given object².
- Boxes whose faces have the average areas and colors as the corresponding sides of the object’s bounding box.
- Texture mapped boxes. This representation uses texture maps that are pasted onto each face of the object’s bounding box and is useful to represent box like objects such as the Standard Oil Building in Chicago.

4.2 Impostor Selection

There are certain cases where specific impostors are more suitable than others and we can usually “suggest” to the walkthrough program which representation to display at a given point in the simulation.

For example, if the image-space size N of an object is less than a few pixels then the representation that should be used is the average box above. If N is greater than a pre-fixed maximum size then the full detail object might be required. If different LODs are present in the model, then different image space size thresholds may be used to select the appropriate LOD to be displayed.

Box-like and symmetric objects can be displayed using a texture mapped box and a billboard, respectively. Texture maps can be selected according to the observer’s viewpoint. For example, if four texture maps are used for each face of an object’s bounding box, then the appropriate texture map for a given viewpoint can be selected as follows:

1. In a pre-processing phase, associate to each texture map a number corresponding to the region it belongs as in Figure 3.

¹It can be used in machines that do not have texture mapping hardware.

²Some toolkits such as Performer[6] provide routines to automatically generate coarse versions of a given full-detail object.

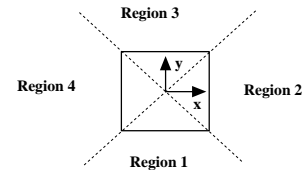


Figure 3: Possible viewpoint regions in object coordinates.

2. During the walkthrough we determine the viewpoint with respect to the object’s coordinate system and therefore the region it is in.

In some situations, both a view dependent and a view independent representation are suitable. When this is the case, we can decide upon these two representations by obtaining the accuracy of each representation for the particular observer view angle using the table described in Section 3.2 and then select the representation with the highest accuracy/cost ratio. This heuristic is particularly useful in cases where the observer’s line of sight is approaching a 45 degree angle with the line perpendicular to the texture map. In such a case although the texture map may have a low rendering cost, its accuracy will also have a low value which will favor the selection of a possibly more costly view dependent representation.

4.3 Formalization of the Problem

We begin by defining a meta-object abstraction to be an entity with one or more hardware drawable representations as in the framework described in Section 4.1. It is an abstraction for both conceptual objects and groups of objects.

As before, a hardware drawable representation is an entity that can be rendered by the graphics hardware to represent objects and has associated to it a rendering cost and a measure of its “contribution” to the simulation.

The model is then defined as a collection of conceptual objects at specific positions and orientations in space that forms the environment in which the user navigates.

The model hierarchy is a tree structure whose nodes are meta-objects that provide multiple representations of the model, each representing it at a given rendering time and providing the user with a given perception of it. In this hierarchy each node is a meta-object that contains drawable representations of its children. The root contains the coarsest representations for the entire model with the lowest possible rendering cost while the leaves form the perceptually best representation of the model with the highest rendering cost.

More formally, the model hierarchy M is a tree structure that can recursively be defined by the following rules:

1. A meta-object that has no children is a model hierarchy with just one node, the root node.
2. Let $M_1, M_2 \dots M_n$ be model hierarchies whose root nodes are the meta-objects $m_1, m_2 \dots m_n$, respectively, that represent sets of conceptual objects and have associated with each of them the sets $r_1, r_2 \dots r_n$ of drawable representations. Let m be a meta-object that represents the union of m_i and has associated to it a set r of drawable representations such that $Cost(Max(r)) < \sum_{i=1}^n Cost(Min(r_i))$, where $Max(r)$ is the representation that has the highest cost among those in r , $Min(r_i)$ is the representation that has the lowest cost among

those in r_i and $Cost(x)$ is the rendering cost of representation x . M is then defined to be a model hierarchy if m is the parent of m_i for $i = 1 \dots n$.

Figure A shows how the model of a city would be organized to form a hierarchy in which each node has a set of impostors to represent the objects it subsumes.

Given these definitions, we state the walk-through problem as a tree traversal problem:

“Select a set of nodes in the model hierarchy that provides the user with a perceptually good representation of the model”, according to the following constraints:

1. The sum of the rendering cost of all selected nodes is less than the user specified frame time.
2. Only one node can be selected for each path from the root node to a leaf node, since each node already contains drawable representations that represent all its descendant nodes.

The problem as described here is an NP-complete tree traversal problem and is a variant of the “Knapsack problem”, which is not surprising since we are generalizing the system that Funkhouser and Sequin showed to be a knapsack problem. The candidate sets from which only one element will be selected to be put in the knapsack are the set of representations associated to each meta-object. The knapsack size is the frame time per frame that the selected representations must not exceed. The cost of each element is the rendering cost associated to a representation. The profit of an element is the accuracy of the representation plus the benefit of the object with which it is associated.

To solve this problem we use the framework described in Section 4.1 and develop a model hierarchy building algorithm and a heuristic representation selection algorithm.

4.4 Design of the Model Hierarchy

We partition the entire model according to our formalization of the problem, and form a tree structure in which each node contains low-cost representations for the nodes it subsumes.

The structure that we use is a variation of an octree that is a bounding volume hierarchy, that can be used to cull objects against the viewing frustum and also serves as a rendering aid, since we can draw its nodes.

This tree is constructed in a bottom-up fashion instead of the traditional top-down recursive way, so that we can see which objects are being “clustered”³ together as described in Section 5.

The criteria used to group objects takes into account only the proximity of objects and our model hierarchy building program is designed to cluster together nearby objects first in the way illustrated in the 2D example of Figure 4.

According to a user-supplied number of divisions in x, y, and z axis of the bounding box of the entire model an initial octree cell size and therefore tree depth is specified. We start by creating a “child list” that contains all the conceptual objects in the model with their bounding boxes. This initial list will correspond to the leaves of the tree. The child list is used to generate the next level up of the tree. For each

³What is meant by clustering is basically the generation of impostors for the group of objects.

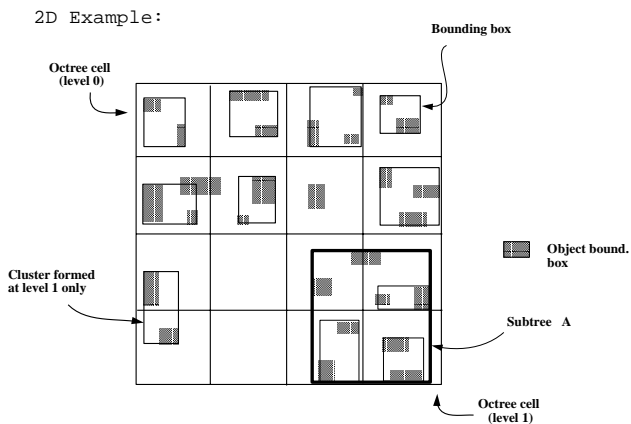


Figure 4: Generating the model hierarchy octree. Representations are generated for cells with more than one object.

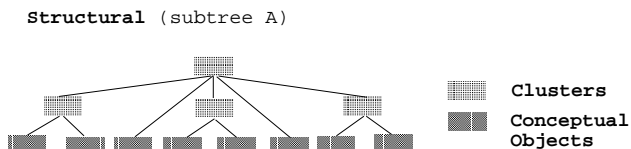


Figure 5: Subtree A as depicted on Figure 4.

level of the tree and for each cell in that level, we get the set of objects that are completely inside the cell. If this set is empty we move on to the next cell. Otherwise we compute the bounding box of the objects in the cell and discard it if the bounding box is already in the child list, since impostor representations for that set of objects had already been created. If it is not in the list we create impostor representations for the cluster inside the cell.

In our implementation clusters are generated by creating texture maps footnoteIn the algorithm, unlike our implementation, representations only need to obey the cost requirement stated in the formalization Section 4.3. of the objects from given view angles and their generation is described in Section 5. After the impostor representations have been created, we make the cell point to its children and remove them from the child list. We then add the new cell to the end of the child list and repeat the process until we obtain a single cell with impostor representations for the entire model.

It is important to note that at each time we cluster objects we always take into account the actual objects that the cell subtends instead of previously computed clusters.

Note that representations are generated only if there are more than one object totally inside each cell. Single objects inside a cell as well as objects on cell boundaries will be grouped in the next levels up in the hierarchy. Figure 5 shows the structure of subtree A depicted in Figure 4.

4.5 Traversal of the Model Hierarchy

Due to the NP-complete nature of selecting representations to render from the model hierarchy, we have devised a heuristic algorithm that quickly (in less than the frame time) traverses the model hierarchy. This algorithm selects representations to be rendered, accumulating rendering cost until the user-specified frame time is reached. When this occurs, the algorithm stops and sends a list of representations to the

graphics pipeline.

The tree traversal is top-down from the root node and first traverses the branches that contain the most “beneficial” nodes according to the benefit heuristic presented in Section 3.1.

A problem is that our per-object benefit heuristic associates benefit not to cluster representations but to representations for conceptual objects that are at the very bottom of the tree. High up in the hierarchy we do not know to which branches of the tree the most beneficial objects belong. Because of this, we have decided to break the selection of nodes to render in two phases as described below.

4.5.1 First Pass: Assign Initial Representation, Benefit, Visibility, and Cost.

In this first phase of the selection process, we recursively descend the model hierarchy in a depth-first manner and associate a benefit and visibility value with each node in the tree, and an initial drawable representation.

Since the leaves represent single objects, their benefits are computed as a weighted average of the factors intrinsic to objects as described in Section 3.1. The benefit value associated to a tree node is taken to be the maximum value of all the benefits of its children.

The visibility of nodes are computed by checking if the bounding box in eye-coordinates of the bounding box of the object intersects the viewing frustum. A node is said to be visible if at least one of its children is visible.

At a given point in the simulation a view dependent and a view independent representation for an object is selected using the criteria specified in Section 4.1.2. The rendering cost and accuracy of drawable representations that are stored with each representation in the model are used to select which of these two representations will be assigned to be the initial representation of the node. In this case, the representation that has the highest accuracy/cost ratio is selected to be the initial representation. In the next phase (described below), if there is still frame time left we try to improve on this initial choice.

After initial representations are selected to each of a node's children, the children's cost is stored with the node to be used in the next phase.

4.5.2 Second Pass: Best-First Tree Traversal.

In this phase, we use the information obtained in the previous phase for each node of the model hierarchy to implement an efficient ‘best-first’ tree traversal. The result of this traversal is a rendering list of drawable representations that is sent to the graphics hardware for rendering as shown in Figure 6.

To implement this strategy, we make use of a list of meta-objects organized in decreasing order of benefit (computed in the previous phase). We keep accumulating frame time as we select representations to render and whenever the time required to render the children of a node plus the total accumulated time so far exceeds the frame time we insert the representation for the node in the rendering list and move on to the next node.

The algorithm first explores the branches of the tree connected to the most beneficial nodes as follows: Start by inserting the root node in the list and setting the total rendering cost to be the cost of rendering the initial representation associated to the root node. We then process this list until it is empty. We remove the element in the front of the list and discard it if it is not visible.

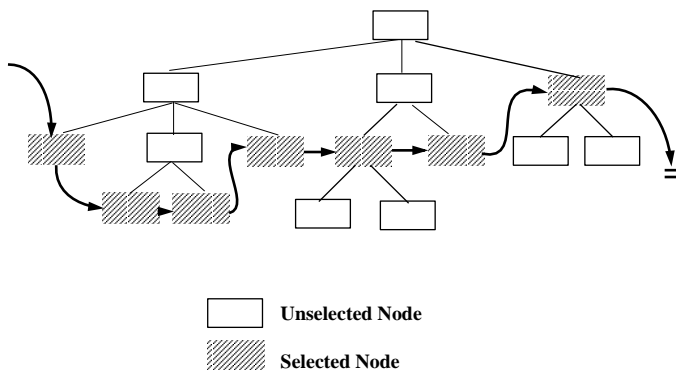


Figure 6: Tree representing the model hierarchy and the set of nodes to be rendered as a linked list.

If the node is a leaf node (containing a conceptual object) we check if there is still rendering time left to select a better representation for the object. In the positive case we select to render (insert in the rendering list) the next higher accuracy representation for the node and add its rendering time to the total accumulated rendering time.

In the case where the node contains representations for a cluster of objects, we check if instead of rendering the cluster representation we still have time to render all of its children, i.e., the total accumulated time plus the cost of rendering the node's children does not exceed the frame time. In the positive case, we insert each of its visible children in the list ordered by each one's benefit and add their cost to the total accumulated rendering time. Otherwise we insert the cluster's representation into the rendering list.

Note that at each point in this traversal, a complete representation of the scene is stored in the list of meta-objects and whenever there is frame time left to render the children of a node, before adding the cost of the children to the total accumulated cost we subtract the cost of the initial representation for the node.

4.6 Temporal Coherence

While navigating through the model the viewpoint can randomly get close or far away from “important” objects that require most of the frame time. This sometimes causes a seemingly random switch from a cluster representation to the representations of the actual objects (or vice-versa). The idea of using frame-to-frame temporal coherence as used by Funkhouser and Sequin, is used here to minimize this effect by discouraging switching from representations for parent nodes to representations for children nodes. We keep a counter of the number of times the walkthrough program decided to switch from parent to children. The actual switching is only allowed if this counter exceeds a pre-fixed threshold. The delayed switching from children representations to cluster representations is not implemented since it would occur in a situation that most of the frame time has already been allocated and this delay would greatly reduce the frame rate.

5 Implementation

This research has resulted in the implementation of three programs on a four processor SGI Onix workstation with a RealityEngine board: the model hierarchy building and

representation generation program, the cost and accuracy of representations measurement program, and the walkthrough program.

These programs are implemented in C++, use GL[8] for rendering, and have an X-Motif GUI to facilitate parameter changes for system evaluation.

5.1 Model Hierarchy Building and Representation Generation

The program that builds the model hierarchy implements the hierarchy building algorithm described in section 4.4 and opens two windows, as shown in Figure B. The right window displays the objects/clusters and compute texture maps for each of the sides of their bounding boxes while the left illustrates the process of building the hierarchy. In this image, the dots represents objects that were not “clustered” yet. The purple square with green dots is the bounding box of the objects in green that completely fit inside it and the “red” band is showing groups of objects already “clustered”.

View dependent impostors such as texture maps are automatically obtained in the following way with the help of the graphics hardware:

1. Set up a viewpoint, a viewing direction, and an orthographic projection matrix.
2. Draw the object(s) in a completely black background and adjust the texture resolution⁴ by scaling the object(s) inside the orthographic viewing volume.
3. Grab the resulting image from the window (right window in Figure B) and set the alpha component of black pixels to zero, so that if the objects have holes we can see through when they are rendered.

Average color boxes are also obtained in a similar fashion. The average color for each face is just the average of the rgb colors of all non-black pixels and the average area is the number of all non-black pixels in the face’s image that is converted to an area in object coordinates.

The generation of a pseudo-texture map involves a pre-processing of the original image because if there are too many pixels on the image the rendering of the texture would require rendering too many meshed triangles. Therefore, we successively shrink the original image by convolving it with a Gaussian filter that averages the RGB components of the pixels.

5.2 Cost and Accuracy of Representations Measurement

The cost of each representation is measured by selecting a specific representation and drawing it a number of times in order to get an average rendering time as shown in Figure C.

The accuracy of an impostor is measured using the procedure described in Section 3.2 and a table that describes how similar each of the representations is compared to the original image of the object for five directions around the object is generated. One of the most immediate improvements we need to make is the use of more directions in the table.

⁴What ultimately determines the resolution of the texture map is the complexity (or granularity of details) that the object(s) exhibit(s) from a particular direction.

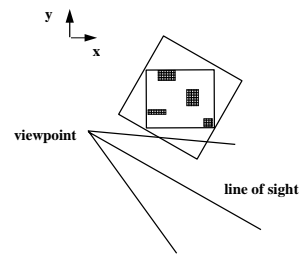


Figure 7: Checking the visibility of a set of objects against the viewing frustum. The test returns true although no object in the cluster is inside the viewing frustum.

5.3 Visual Navigation

The walkthrough program implements the framework described in Section 4.1 and the traversal algorithms described in Section 4.5. The computation of the representation to be rendered in the next frame is done in one processor while another one holds the graphics pipeline to render the current frame. Semaphores are used to synchronize the two processes.

The traversal algorithm assumes that visibility of bounding boxes can be determined quickly. This can be done by first computing the bounding box in eye-coordinates of the object’s bounding box. We then compute its intersection with a box formed by extending the slice of the viewing frustum corresponding to the farthest z-value of this box to its nearest z-value. This visibility test can return true even though no object inside the cluster is also inside the viewing frustum as shown in Figure 7.

This problem is solved by the first phase of the traversal algorithm since it marks a cluster as visible if and only if at least one of the objects that it represents is inside the viewing frustum. If computing the visibility of individual objects are taking too much time we can use a faster test and check if spheres enclosing groups of objects intersect the viewing frustum.

5.4 Performance

Our test model has around 1.6 million polygons and during our tests we have constrained the number and size of texture maps generated by the hierarchy building program to the available texture memory of one megatexel (one million texture pixels) by selecting appropriate octree cell sizes and adjusting the resolution of the texture representation for objects and clusters.

For this model we were able to keep a frame rate of around 16 frames per second (fps) for a target frame rate of 30 fps throughout the simulation without too much degradation in image quality. Figure D shows the image seen by the observer (left) and a top view of the the same scene showing where clusters are being displayed (right).

Figure 8 shows the user mode (right) and real time (left) throughout a simulation path of the model. The user time graph shows that our estimation of cost and rendering algorithm is achieving the goal of keeping a uniform and high frame rate. The real time graph show spikes due to random interrupts and a gap with respect to the 1/30 line due to smooth LOD switching using transparency blending.

These interrupts can be minimized by mechanisms such as processor isolation, interrupts redirection, processor locking and so on as described in [9].

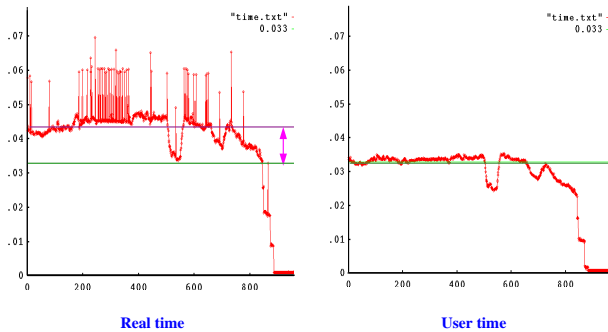


Figure 8: Plot frame versus frame time with (left) and without (right) smooth LOD switching.

The same model, without the model hierarchy, takes around 1 frame per second for certain viewpoints in our test path.

6 Limitations

One limitation of this system is the number of texture maps that can be used to represent objects and clusters. As the system uses more texture maps to represent clusters and individual objects, the chance of a texture-cache miss increases. A cache miss results in an unpredictable interrupt that will invariably defeat the purpose of a predictive system. Future methods of intelligent prefetch of textures that are likely to be needed could make texture cache misses much less likely, and thus allow the use of many more textured impostors.

We have not addressed the illumination of the environment. Although the illumination of a complex environment can be computed using the radiosity method in a view independent fashion the shading attributes of objects (adding specular highlights) and clusters would need to be incorporated to their representations. Instancing of objects would not be practical since two identical objects in different locations in the model would have different shading attributes. As the size of texture memory increases these problems will become less serious, but they will not go away.

The most serious limitation in our current implementation is that our benefit calculation requires that a cluster know something about the benefits of its children, so all primitives are visited once per frame, and our program is thus $O(N)$, where N is the number of objects. Our traversal algorithm is top-down, so there is no reason it could not be $O(\log N)$ if a more intelligent benefit algorithm for representation selection is used.

7 Conclusion

We have presented way of using clusters of objects to improve the performance of an LOD-based visual navigation system. When there are too many visible LODs to render in real-time, we render single texture-mapped cluster primitives in place of groups of individual LODs. The techniques used to generate clusters can also be used to generate a particular type of textured LODs for single primitives. We have also discussed some limitations of the object-based benefit heuristic, and extended it to account for the variability of an LOD's quality as the view angle changes.

The main lessons to be drawn from this work are that the predictive framework of Funkhouser and Sequin extends well to a hierarchical version of the LOD concept, and that pre-computed visibility information is not essential for efficient visual navigation programs.

8 Acknowledgments

Thanks to Ken Chiu and Aaron Yonas for their suggestions on the draft of this paper. Thanks to Ken Chiu and Paul Bourke for the model of a tree and a town house, respectively, used in the color plates. Thanks to the Brazilian government agency CAPES, for providing the first author the financial support to conduct this research. Thanks to the Research and University Graduate Schools (RUGS) Research Facility Fund (RFF) and the NSF CDA-92-23008 grants that provided the graphics workstations that were used in this research. The second author was also supported by NSF RIA grant CCR-92-09457.

References

- [1] John M. Airey, John H. Rohlf, and Jr Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics*, pages 41–50, 1990.
- [2] Kurt Akeley. Reality engine graphics. *Proceedings of SIGGRAPH'93 (Anaheim, California, August 1-6, 1993)*. In *Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH*, pages 109–116.
- [3] Thomas A. Funkhouser and Carlo H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Proceedings of SIGGRAPH'93 (Anaheim, California, August 1-6, 1993)*. In *Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH*, pages 247–254.
- [4] Thomas A. Funkhouser, Carlo H. Sequin, and Seth Teller. Management of large amounts of data in interactive building walkthroughs. *Proceedings of the 1992 Symposium on Interactive 3D Graphics (Cambridge, Massachusetts, March 29 - April 1, 1992)*, special issue of *Computer Graphics, ACM SIGGRAPH*, pages 11–20, 1992.
- [5] Paulo Maciel. Visual navigation of largely unoccluded environments using textured clusters. *Ph.D. Thesis*, January 1995. Indiana University, Bloomington.
- [6] John Rohlf and James Helman. Iris performer: A high performance multiprocessing toolkit for real-time 3D graphics. *Proceedings of SIGGRAPH'94 (Orlando, Florida, July 24-29, 1994)*. In *Computer Graphics Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH*, pages 381–394.
- [7] Harvey R. Schiffman. *Sensation and Perception an Integrated Approach*. John Wiley & Sons, New York, 1990.
- [8] Inc. Silicon Graphics. *Graphics Library Programming Guide, Volumes I and II*, 1992.
- [9] Inc. Silicon Graphics. *React In Irix: A description of real-time capabilities of Irix v5.3 on Onyx/Challenge multiprocessor systems.*, 1994.