internetworks.

IP broadcast, which is commonly used in DIS environments, cannot be used over the Internet unless it is encapsulated. It also adds an additional burden because it requires that all nodes examine a packet even if the information is not intended for that receiving host, incurring a major performance penalty for that host because it must interrupt operations in order to perform this task at the operating system level.

Point-to-point communication requires the establishment of a connection or path from each node to every other node in the network for a total of N*(N-1) virtual connections in a group. For example, with a 1000 member group each individual host would have to separately address and send 999 identical packets. If a client-server model is used, such as that typically found in networked games and multi-user domains (MUDs), the server manages all the connections and rapidly becomes an input/output bottleneck.

Dead-reckoning

The networking technique used in NPSNET-IV, evolved from SIMNET, and embodied in DIS follows the *players and ghosts* paradigm presented in [1][7]. In this paradigm, each object is controlled on its own host workstation by a software object called a Player. On every other workstation in the network, a version of the Player is dynamically modeled as an object called a Ghost.

The Ghost objects on each workstation update their own position each time through the simulation loop, using a dead-reckoning algorithm. The Player tracks both its actual position and the predicted position calculated with dead-reckoning. An updated Entity State Protocol Data Unit is sent out on the network when the two postures differ by a predetermined error threshold, or when a fixed amount or time has passed since the last update (nominally 5 seconds). When the updated posture (location and orientation) and velocity vectors are received by the Ghost object, the Ghost's is corrected to the updated values, and resumes dead-reckoning from this new posture.

This dead-reckoning technique helps in overcoming a major problem found in a number of networked simulations -- excessive network utilization. For example, each networked participant playing the popular game DOOM generates a packet on every graphics frame. On an SGI, this translates into 30 packets per second, even when an entity is inactive. This not only wastes bandwidth, it also overloads the ability of network devices to process packets. On the other hand, a high performance aircraft in a DIS environment typically produces about 8 packets per second --a dramatic difference.

MBONE

MBONE is a virtual network that originated from an effort to multicast audio and video from the Internet Engineering Task Force (IETF) meetings [2]. MBONE today is used by several hundred researchers for developing protocols and applications for group communication.

We have used MBONE to demonstrate the feasibility of IP Multicast for distributed simulations over a wide area network. In the past, participation with other sites required prior coordination for reserving bandwidth on the Defense Simulations Internet (DSI). DSI, funded by ARPA, is a private line network composed of T-1 (1.5 Mbps) links, BBN switches and gateways using the ST-II network protocol. It had been necessary to use DSI because ARPA sponsored DIS simulations use IP broadcast - requiring a unique wide-area bridged network.

With the inclusion of IP Multicast in NPSNET-IV, sites connected via the MBONE can immediately participate in a simulation. MBONE uses a tool developed by Van Jacobson and Steven McCanne called the Session Directory (*SD*) to display the advertisements by multicast groups. SD is also used for launching multicast applications like NPSNET-IV and for automatically selecting an unused address for a new group session. Furthermore, we can integrate other multicast services such as video with NPSNET-IV. For example, participants are able to view each other's simulation with a video tool, NV, developed by Ron Fredrickson at Xerox Parc [5].

Acknowledgments

We wish to express our thanks to the Air Force Institute of Technology, George Mason University, the Naval Research Labs and all those who participated in the demonstrations of NPSNET-IV.

This work would not have been possible without the support of our research sponsors: USA ARL, DMSO, USA STRICOM, USA HQDA AI Center-Pentagon, USA TRAC, ARPA.

Resources

Many of the references noted below are available via the NPSNET Research Group's WWW home page:

file://taurus.cs.nps.navy.mil/pub/NPSNET_MOSAIC/npsnet_mosaic.html

References

1.Blau, Brian, Hughes, Charles E., Moshell, J. Michael and Lisle, Curtis "Networked Virtual Environments," Computer Graphics, 1992 Symposium on Interactive 3D Graphics (March 1992), pp.157.

2.Casner, Steve. "Frequently Asked Questions on the Multicast Backbone". (6 May 1993). Available at venrera.isi.edu:/mbone/faq.txt.

3.Deering, Stephen. Host Extensions for IP Multicasting. RFC 1112. (August 1989).

4.Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation, (March 1993).

5.Macedonia, Michael R. and Donald P. Brutzman. "MBone Provides Audio and Video Across the Internet". In IEEE Computer. (April 1994). pp. 30-36.

6.Pope, Arthur, BBN Report No. 7102, "The SIMNET Network and Protocols", BBN Systems and Technologies, Cambridge, Massachusetts, (July 1989).

7.Pratt, David R. "A Software Architecture for the Construction and Management of Real Time Virtual Environments". Dissertation, Naval Postgraduate School, Monterey, California (June 1993).

8.Zyda, Michael J., Pratt, David R., John S. Falby, Chuck Lombardo, Kelleher, Kristen M. "The Software Required for the Computer Generation of Virtual Environments". In Presence. 2, 2. (Spring 1993). pp. 130-140.

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 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 concept of choosing a representation from among those associated with an object that accounts for the direction from which the object is viewed is also supported. The level-of-detail concept is extended to the whole model and 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 representation 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 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 with high benefit/cost value to render until the user-specified

*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

1995 Symposium on Interactive 3D Graphics, Monterey CA USA
© 1995 ACM 0-89791-736-7/95/0004...$3.50

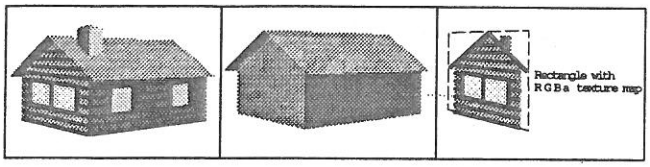

Figure 1: Three representations for a house. The left two are view independent LODs while the right one is a view dependent texture map.

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 walkthrough 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.

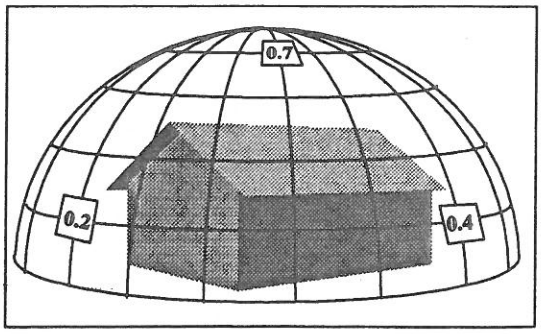

Figure 2: Discretizing the space of viewpoints around an object. Replication accuracies are shown at three of the view angles. The low LOD house looks the "best" from the top.

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 then 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 then 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 map[1]. 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 object[2].
- 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 then a few pixels then the representation that should be used is the average box above. If N is greater then 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 obeserver'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.

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

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

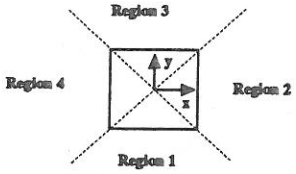

Figure 3: Possible viewpoint regions in object coordinates.

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 defined to be 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 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 $M_1, M_2 ... M_n$ be model hierarchies whose root nodes are the meta-objects $m_1, m_2 ... m_n$, respectively, that represent sets of conceptual objects and have associated with each of them the sets $r_1, r_2 ... r_n$ 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 $i = 1 \ldots n$.

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

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

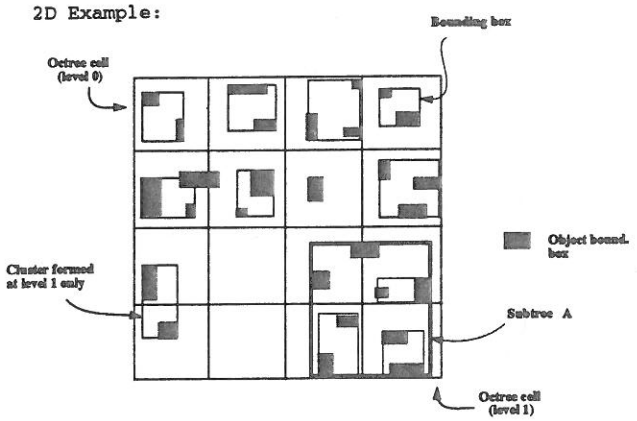

Figure 4: Generating the model hierarchy octree. Representations are generated for cells with more than one object.

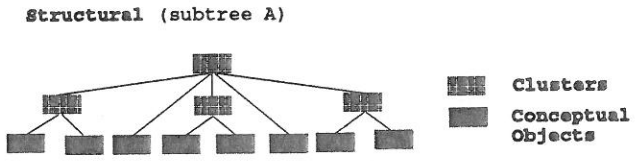

Figure 5: Subtree A as depicted on Figure 4.

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[4] 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 cluster representations are generated only if there is more then 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,

[4]Actually, representations only need to obey the cost requirement stated in Section 4.3.

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.

The 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 4.2. 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. 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

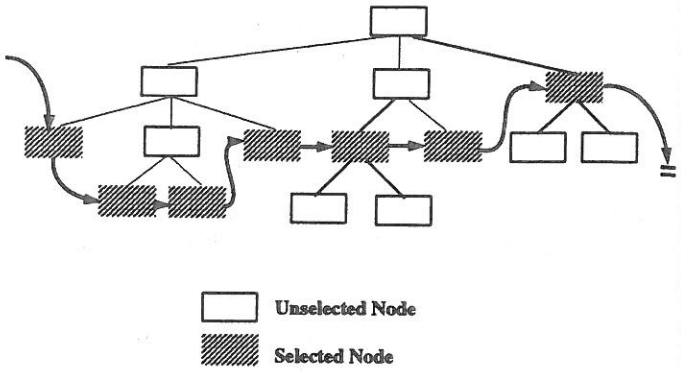

Figure 6: Tree representing the model hierarchy and the set of nodes to be rendered as a linked list.

it is empty. We remove the element in the front of the list and discard it if it is not visible.

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 ones 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 mininimize 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 resolution[5] 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 preprocessing of the original image because if there are too many pixels on the image the rendering of the texture would require 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 this table.

[5] What 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.

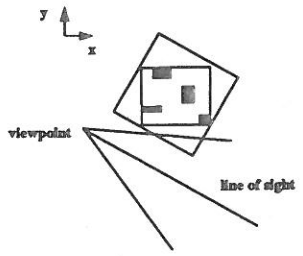

Figure 7: Checking the visibility of a set of objects against the viewing frustum.

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.

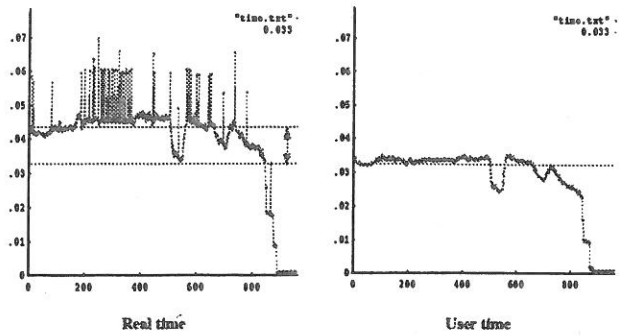

Figure 8: Plot frame versus frame time with (left) and without (right) smooth LOD switching.

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

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 tree traversal requires that a cluster know something about the benefits of its children, so all primitives are visited once per frame in the first pass of the algorithm, and therefore it is $O(n)$, 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 traversal algorithm is used.

7 Conclusion

We have presented a 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 precomputed 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, respectivelly, 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 environmnets. *Computer Graphics*, 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.

Guided Navigation of Virtual Environments

Tinsley A. Galyean

MIT Media Lab
Cambridge, MA. 02139
tag@media.mit.edu

ABSTRACT

This paper presents a new method for navigating virtual environments called "The River Analogy." This analogy provides a new way of thinking about the user's relationship to the virtual environment; guiding the user's continuous and direct input within both space and time allowing a more narrative presentation. The paper then presents the details of how this analogy was applied to a VR experience that is now part of the permanent collection at the Chicago Museum of Science and Industry.

1. INTRODUCTION

Today's Virtual Reality (VR) technology provides us with an opportunity to have experiences that would otherwise be impossible. We can smoothly and continuously interact while immersed in environments that would be inaccessible or impossible to experience. In these environments, we are free to roam and explore. architectural walk throughs for example, scientific visualization, and even games like DOOM place us in alternative worlds while giving us methods for navigating these virtual spaces. These methods allow smooth and continuous interaction that can immediately influence the constantly changing presentation, but they rely on the user's actions and thoughts to bring structure to the experience. If any narrative structure (or story) emerges it is a product of our interactions and goals as we navigate the experience. I call this emergent narrative. In some applications this complete freedom to explore is appropriate. However, there is an alternative. This is the process of empowering the author to bring structure to the experience, which makes this medium more appropriate for applications such as teaching, storytelling, advertising and information presentation. To do this, we will need to balance the interaction (exploration) with an ability to guide the user, while at the same time maintaining a sense of pacing or flow through the experience. This type of guidance is the process of a providing narrative structure. Like a narrative presentation any solution must guide the user both temporally and spatially.

2. PREVIOUS WORK

Virtual environment navigation has mainly consisted of building new methods and technologies that allow the users to control the position and orientation of the virtual camera, through which they see the virtual world. Early work in camera control (even before the advent of VR technology) focused on specifying camera movements over a path. [1, 4] In an effort to address the needs of animation production and does not address the issue of interactive camera control.

A number of different researchers have addressed the issues behind camera control for manipulation and/or exploration applications. [2, 5] All of these methods focus on providing better ways for the user to roam free, exploring within the virtual world. It is this ability for the user to directly control his/her place in the virtual world that is so often synonymous with the words "virtual reality." While these methods couple smooth, continuous interaction with the smooth and continuous presentation available in real-time computer graphics environments, they do nothing to guide the user. There is no room for an author's intentions to influence the experience. Therefore there is no narrative structure.

Researchers in interactive narrative working with linear material like digital video have worked to unfold it in order to provide a non-linear environment for the user [3]. Shots are interactively laced together into sequences and these sequences tell a story. Because shots are the smallest building blocks available, the interaction intermittently guides how these shots are laced together.

The traditional analogy of how these types of interactive experiences are structured is often referred to as the branching analogy. Each branch represents a linear segment traversing part of the narrative space. A linear segment is played until the next node is reached. It is at these nodes where options are provided. The advantage of branching is that the experience does have a narrative structure, the interaction is guided. The disadvantage is that one can interact only at the nodes thereby chopping up both the interaction and the presentation.

The goal set forth is to find a way to marry the advantages of immersive VR experience with the advantages of narrative structure. How do we allow the smooth, continuous interaction and presentation, to coexist-exist with the structural and temporal qualities of narrative (plot and pacing)? In other words, how do we balance the notion of interaction with guidance (telling).

3. THE RIVER ANALOGY

Here I propose an alternative analogy for navigating virtual spaces. Instead of linking a sequence of branches and nodes, or giving the user free rein, I suggest that the navigation paths be more like a river flowing through a landscape. The user is a boat floating down this river with some latitude and control while also being pushed and pulled by the pre-defined current of the water. Like the branching structure this approach constrains the audience's movement through the space to interesting and compelling paths. But there are some unique advantages to this approach: the flow of the experience, the continuous input of the rudder, and multiple levels of structure.

The river analogy assures an uninterrupted flow. When in a boat you float down the river even when you are not steering. The pre-

1995 Symposium on Interactive 3D Graphics, Monterey CA USA
© 1995 ACM 0-89791-736-7/95/0004...$3.50

sentation is continuous regardless of whether or not there is input. The amount of control you have over the boat varies with the properties of the river. If the rapids increase, you move faster with less room for swinging from side to side. Alternatively, the pace can slow and the river can widen giving room to steer from one bank to the other.

In the river analogy the boat's rudder can be likened to audience input. A rudder takes input continuously. The amount of influence may vary depending on the water conditions but you can always provide the input. It is also the case that the rudder may have both an immediate and a long term impact on the navigation. How the rudders are used can determine both your local position within the river, but also a more global position, such as which fork in the river your boat might take.

The river provides two levels of guiding structure. First is the local structure of the river including the water flow, rocks in the river, the width between the banks, etc. Second, is the global structure, including both the path the river flows and the forks that separate and/or rejoin. The audience input has influence on how both levels of this representation are navigated. The rudder or input can steer between the banks while the position of the boat when a fork is reached will dictate which part of the fork the audience will travel.

Like a river, a guiding navigation method should guide without interruption of the presentation. This creates a sense of interaction by constantly accepting user input and guiding it with a higher level, longer term structure.

4. APPLICATION OF THE RIVER ANALOGY

A highlight of the Chicago Museum of Science and Industry's new exhibit, *Imaging the Tools of Science*, is the virtual reality experience. The primary goal of this exhibit was to expose and educate the visitor to what VR technology is and can do. Any experience that was going to be successful, was going to be highly constrained by the issues inherent in bringing an immersive experience to a public place like the museum. In a museum setting it is necessary to limit the amount of time a person spends, provide an interface that keeps people from getting lost and frustrated, while at the same time making them aware that they have some direct and immediate control over how they move through the environment. To meet these demands it was decided that the experience would be between 2 and 3 minutes long with a clear beginning, middle, and end. This allowed the user to feel they had a complete experience while allowing the museum to predict how quickly they could move people through the exhibit. These constraints required the user's navigation to be guided through the virtual world, and the river analogy helped us address these issues.

In this application, the analogy of the river was taken quite literally. We defined a path through the virtual space as the river. The user was then guided through the space much like a water-skier behind an invisible boat. The boat or anchor moves along the path at a rate that varies as specified by the creator of the experience (the author). The user is then tethered to the anchor by a spring that constantly pulls them along. Meanwhile the user is free to look in any direction he or she chooses. Figure 1 shows the model we used. This model gives the user direct control over where they are looking while at the same time giving them indirect control over their local position. Looking in a given direction will impart some force in that direction and allow the user to swing over in that direction moving closer to the object they are watching. At the same time the boat continues to pull them along the journey, maintaining a sense of pacing and flow.

There are a series of parameters that can change the nature of this interface: the current and desired speed of the anchor, the amount of thrust the user is imparting, and the spring and damping constants. In this implementation, all of these values are free to change throughout the experience. The changes are encoded at

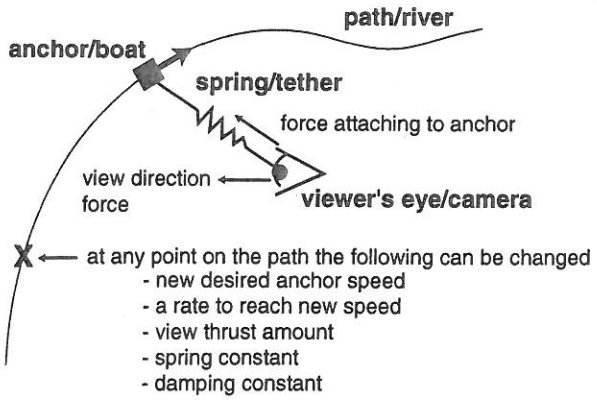

Figure 1: An application of the River Analogy, consisting of a number of different parts: the anchor moving along the path, a spring attaching the user position to the anchor, a thrust imparted by the user dictated by the direction the user is looking, and a general viscous damping to prevent the user from oscillating about the anchor position.

locations along the path, allowing the author to specify over which areas of the journey the user is more or less free to roam. For example, as the user approaches a larger open space the author many choose to slow down the anchor, decrease the spring and damping constants, and increase the viewer thrust allowing the user more latitude and time to explore. Alternatively, the author might focus the experience by increasing the spring constant, speeding up the anchor, and reducing the thrust.

5. CONCLUSION

It is clear that there are VR application for which the current methods of navigation are not sufficient. Some of these applications suggest the need for a method to guide the user as s/he navigates the virtual landscape. The River Analogy provides a way of thinking about how the author's intentions can steer the interaction given by the user to create a guided navigation. This paper has presented this analogy and one particular application of this analogy to an existing public VR exhibit. This work only begins to touch on the potential of guided interaction for virtual environments.

ACKNOWLEDGEMENTS

The Art Technology Group including: Martin Friedmann, Stephen Clark, Andy Schwarz, Keith Baccki, Andy Hong, David Rose, Joe Chung, Jeet Singh. Martin in particular for outstanding work on implementation. Glorianna Davenport (Media Lab) for her help and support. Steven Drucker for the many inspiring conversations.

REFERENCES

1. Bartels, R., J. Beatty, and B. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgen Kaufmann, Los Angeles, 1987.

2. Brooks, F. P. Grasping Reality Through Illusion -- Interactive Graphics Serving Science. *CHI '88 Proceedings*, Special Issue of SICHI Bulletin,1988, 1-11.

3. Davenport, G., T. Aguierre-Smith, and N. Pincever, Cinematic Primitives for Multimedia, Computer Graphics & Applications, (July 1991), 67-73.

4. Shoemake, K. Animating Rotation with Quaternion Curves. Proceeding of SIGGRAPH '85. (San Francisco, California, July 22-26, 1985). In Computer Graphics 19, 3 (July 1985), 245-254

5. Ware, C. and S. Osborne. "Exploration and Virtual Camera Control in Virtual Three Dimensional Environments," Proceedings of the 1990 Symposium on Interactive 3D Graphics (Snowbird, Utah, March 25-28, 1990),special issue of Computer Graphics, ACM SIGGRAPH, New York, 1990, 175-184

Portals and Mirrors:

Simple, Fast Evaluation of Potentially Visible Sets

David Luebke and Chris Georges
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

We describe an approach for determining potentially visible sets in dynamic architectural models. Our scheme divides the models into cells and portals, computing a conservative estimate of which cells are visible at render time. The technique is simple to implement and can be easily integrated into existing systems, providing increased interactive performance on large architectural models.

Introduction

Architectural models typically exhibit high depth complexity paired with heavy occlusion. The ratio of objects actually visible to the viewer (not occluded by walls) to objects theoretically visible to the viewer (intersecting the view frustum) will usually be small in a walkthrough situation. A visibility algorithm aimed at reducing the number of primitives rendered can exploit this property. Following prior work [1,2,3], we make use of a subdivision that divides such models along the occluding primitives into "cells" and "portals". A cell is a polyhedral volume of space; a portal is a transparent 2D region upon a cell boundary that connects adjacent cells. Cells can only "see" other cells through the portals. In an architectural model, the cell boundaries should follow the walls and partitions, so that cells roughly correspond to the rooms of the building. The portals likewise correspond to the doors and windows through which neighboring rooms can view each other.

Given such a spatial partitioning of the model, we can determine each frame what cells may be visible to the viewer. By traversing only the cells in this potentially visible set (PVS), we can avoid submitting occluded portions of the model to the graphics pipeline. What cells comprise the PVS? Certainly the cell containing the viewpoint is potentially visible. Those neighboring cells which share a portal with the initial cell must also be counted as potentially visible, since the viewer could see those cells through the portal. To this we add those cells visible through the portals of these neighbors, and so on. In this manner the problem of determining what cells are potentially visible to the viewer reduces to the problem of determining what portals are visible through the portals of the viewer's cell.

luebke@cs.unc.edu (919) 962-1825
georges@cs.unc.edu (919) 962-1789
CB# 3175 Sitterson Hall; UNC, Chapel Hill, NC 27599-3175

1995 Symposium on Interactive 3D Graphics, Monterey CA USA
© 1995 ACM 0-89791-736-7/95/0004...$3.50

Our system makes this determination dynamically at render time. Rather than finding the exact PVS for each cell as a preprocess, we postpone the visibility computation as long as possible. This type of dynamic evaluation of portal-portal visibility is not new. Earlier efforts have centered on precisely determining sightlines through portals; our method offers a less exact but much simpler alternative. The algorithm has been implemented on the Pixel-Planes 5 graphics computer at the University of North Carolina and provides a substantial speedup on a sample model of 50,000 polygons.

Previous Work

Jones [1] explored the subdivision of geometry into cells and portals as a technique for hidden line removal. In his algorithm, models are manually subdivided into convex polyhedral cells and convex polygonal portals. The subdivision is complete in the sense that every polygon in the dataset is embedded in the face of one or more cells. Rendering begins by drawing the walls and portals of the cell containing the viewer. As each portal is drawn, the cell on the opposite side of the portal is recursively rendered. In this way the cell adjacency graph defined by the partitioning is traversed in depth-first fashion. The portal sequence through which the current cell is being rendered comprises a convex "mask" to which the contents of the cell are clipped. If the intersection of a portal with the current mask is empty, the portal is invisible and the attached cell need not be traversed.

More recent work has abandoned the attempt to compute exact visibility information, focusing instead on computing a conservative PVS of objects that *may* be visible from the viewer's cell. The graphics pipeline then uses standard Z-buffer techniques to resolve exact visibility. Airey [2] was the first to use a portal-based approach effective in architectural environments. He described multiple ways to approach the problem of determining cell-to-cell visibility, including ray-casting and shadow volumes. Teller [3] has taken the concept further and found a closed-form, analytic solution to the portal-portal visibility problem. Using 2D linear programming to test portal sequences against arbitrary visibility beams, Teller computes a complete set of cell-to-cell and cell-to-object visibilities in a preprocess. At render time this PVS is further restricted according to which portals are actually visible. Teller's approach is mathematically and computationally complex, requiring hours of preprocess time for large models [3].

Motivation

Such a large preprocessing cost may be inappropriate for interactive applications. For example, architectural walkthroughs are often used for revision purposes. A visualization of a building under design is more valuable to an architect if inquiries of the type "What if I move this wall out ten feet?" can be answered immediately. Adding portals, moving portals, and redistributing

cells boundaries will all be common operations in an interactive architectural design application. To take full advantage of the static visibility schemes mentioned above, each of these would require a potentially lengthy PVS recalculation best done off-line.

Envisioning such an application as our final goal, we decided to focus on improving the dynamic visibility determination. Jones' algorithm finds the exact intersection of 2D convex regions, requiring $O(n \lg n)$ time for portal sequences with n edges. Teller's linear programming approach computes only the *existence* of an intersection, and runs in time linear in the number of edges. We sought a dynamic solution that would also run in linear time and would integrate easily into existing systems.

Faster Dynamic PVS Evaluation

We use a variation of Jones' approach that employs bounding boxes instead of general convex regions. Our scheme first projects the vertices of each portal into screen-space and takes the axial 2D bounding box of the resulting points. This 2D box, called the *cull box*, represents a conservative bound for the portal; that is, objects whose screenspace projection falls entirely outside the cull box are guaranteed not to be visible through the portal and may be safely culled away. As each successive portal is traversed, its box is intersected with the aggregate cull box using only a few comparisons.

During traversal the contents of each cell are tested for visibility through the current portal sequence by comparing the screenspace projection of each object's bounding box against the intersected cull box of all portals in the sequence. If the projected bounding box intersects the aggregate cull box, the object is potentially visible through the portals and must be rendered. Since a single object may be visible through multiple portal sequences, we tag each object as we render it. This *object-level culling* lets us avoid rendering objects more than once per frame.

Alternatively, we can render each object once for every portal sequence which admits a view of the object, but clip the actual primitives to the aggregate cull box of each sequence. Under this *primitive-level clipping* scheme objects may be visited more than once, but since the portal boundaries do not overlap, no portion of any primitive will be rendered twice. Typically object-level culling will prove more efficient, but for objects whose per-primitive rendering cost far exceeds their clipping cost, primitive-level clipping provides a viable option.

Implementation

We have implemented this approach on Pixel-Planes 5, the custom graphics multicomputer developed at the University of North Carolina. The traversal mechanism treats portals as primitives to be rendered. Each portal consists of a polygonal boundary and a pointer to the adjacent cell; when a portal is encountered during traversal we test its axial screenspace bounding box against the current aggregate cull box. If the intersection is non-empty, we use it as the new aggregate cull box and recursively traverse the connected cell.

We feel that modeler integration is crucial to this problem of interactive model revision. If an architect wishes to move a wall or broaden a doorway, the modeling system should be able to make the change quickly and broadcast that change to the graphics system. In our system the spatial partitioning of the model into cells and portals is directly embedded in the modeler's representation. Portals are treated as augmented polygons, each tagged with the name of the attached cell. Cells are simply logical groupings in the modeler's hierarchy and need not necessarily be convex. We have found this quite convenient when constructing models; each room typically corresponds to a cell and it takes only seconds to add and move a portal, or to reshape a cell. We have already adapted two commercial modelers to our system, which speaks to the simplicity of the integration process.

Results

We have tested our system on a subset of the UNC Walkthrough project's model of Professor Fred Brooks' house, comprised of 367,000 radiositized triangles. The speedup obtained by this visibility algorithm, like the speedup obtained by similar schemes, is extremely view-and model-dependent. Over a 500-frame test path through the model, the frame rate using PVS evaluation ranged from just over 1 to almost 10 times the frame rate of the entire unculled model. For typical views the dynamic PVS evaluation culled away 20% to 50% of the model. It should be emphasized again that these numbers are specific to the model and view path, but they certainly indicate the promise of the algorithm as a simple, effective acceleration technique.

Ongoing and Future Work

Efficiency could be further increased by applying *obscuration culling* to portals [4]. This scheme tests potentially visible items against an "almost complete" Z-buffer before rendering. This would allow the 'detail' objects in each cell as well as the occluding cell walls to block portals, potentially reducing the PVS. The Pixel-Planes architecture makes obscuration culling of portals feasible, and we are currently exploring this possibility.

Teller mentions that the concept of portals may be extended to mirrors [3]. Under this scheme mirrors are treated as portals which transform the attached cell about the plane of the mirror; this has the advantage of automatically restricting the PVS seen through the mirror. Though conceptually simple, mirrors introduce many practical difficulties which require additional clipping by the rendering engine to resolve. For example, geometry behind the mirror must not appear in its reflected "world," and reflected geometry must not appear in front or to the side of the mirror.

A special case that avoids these problems can be constructed by embedding the mirror in an opaque cell boundary (for example, a wall-mounted mirror in a bathroom), and we have implemented such mirrors (Plate 1). The concept of an immovable mirror fits poorly with our goal of interactive, dynamic environments, however, so we have focused on the more general case. Clipping is complicated further by mirrors that overlap in screenspace, and further still by mirrors which recursively reflect other mirrors. At present our system allows static mirrors, which can reflect each other to arbitrary levels of recursion, or more general "hand-held" mirrors, (an example of free-moving portals), which permit one-bounce reflections. We are currently working on the dynamic, fully recursive case.

Acknowledgments

The authors would like to extend their sincere thanks to Mike Goslin, Hans Weber, Power P. Ponamgi, Peggy Wetzel, and Stump Brady. This work was supported by ARPA Contract DABT63-93-C-C048.

References

[1] Jones, C.B. A New Approach to the 'Hidden Line' Problem. The Computer Journal, vol. 14 no. 3 (August 1971), 232..

[2] Airey, John. Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. Ph.D. thesis, UNC-CH CS Department TR #90-027 (July 1990).

[3] Teller, Seth. Visibility Computation in Densely Occluded Polyhedral Environments. Ph.D. thesis, UC Berkeley CS Department, TR #92/708 (1992).

[4] Greene, Ned, Kass, Michael, and Miller, Gavin. Hierarchical Z-Buffer Visibility. Proceedings of SIGGRAPH '93 (Anaheim, California 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York 1993, pp. 59-66.

Interactive Playing with Large Synthetic Environments

Bruce F. Naylor
AT&T Bell Laboratories
Murray Hill, NJ 07974

Introduction

Until recently, opportunities to experience large synthetic environments have been limited primarily to expensive training simulators. However, with the advent of "location based entertainment" at theme parks and even CD-ROM based games for PC's, these kinds of experiences are beginning to be made available to the general public as well. The constraints on the possibilities for appealing "content" arises from the technological capabilities that are possible for a given performance level on a given platform. Currently, for 3D graphics, performance is closely tied to the number of texture mapped polygons that can be rendered for each frame as well as the rate at which collisions of various kinds can be computed.

Large synthetic environments require at least tens of thousands of polygons, and could easily entail millions. However, for each image, only a small subset of these polygons are typically required to synthesize the image. Similarly, collisions between two objects, or between a viewer and the environment, involve an even smaller subset. The task then for efficient geometric computations is to, if possible, quickly identify the relevant subset. The principal methodology for finding the minimal subset of polygons is to use spatial search structures, such as regular grids, octrees, or binary space partitioning trees. In this paper, we describe briefly the current status of our efforts at using binary space partitioning trees for navigating through and playing with large environments, including rendering and collision detection, as well as permitting interactive modifications of the environment using set operations that should prove appealing for entertainment applications.

Partitioning Trees (or BSP Trees) [Fuchs, Kedem, and Naylor 80] differ from regular grids in that they are hierarchical (multi-resolution), and from octrees in that the method of space partitioning requires not only determining when to partition, but where, as well. The absence of a restriction on the planes used in partitioning trees obviates the need for a distinction between the spatial search structure and the representation of polyhedra by using the planes containing faces to partition space. A single tree, representing some rigid object for example, can be transformed with affine and perspective transformations by only transforming the plane equations; thereby not changing the tree structure. The tree provides a visibility order for rendering objects with any mix of transparent and opaque surfaces, and the near-to-far ordering that can be used for pruning away fully occluded subtrees. In addition, it can be used for efficient solid clipping with a view-volume, for computing shadows and/or global illumination, for intersecting rays with an object, and for determining the location of points (representing, for example, sprites in computer games) in an environment. Spatial relations between two objects can be computed efficiently by merging their respective trees into a single tree [Naylor, Amanatides and Thibault 90]. This provides, on the modeling side, set operations and collision/interference detection. For rendering, tree merging determines inter-object visibility, analogous to merging sorted lists in Merge Sort, which provides the proper ordering required for transparent objects whose convex hulls interpenetrate. It can also be used to discover that a moving object has become totally occluded by another object and so need not be drawn. In addition, tree merging can be used to cast shadows from one object onto another.

Visibility Culling of Large Environments

The most important computation for efficiently navigating through large environments is conceptually a rather simple and familiar one: clipping to the view-volume. However, approaching this using solely the traditional graphics pipeline for polygon clipping in an O(n) process. While any spatial search structure can be used to accelerate this computation, our use of partitioning trees has several consequences. The first is that partitioning trees provide a representation of space as a hierarchy of convex bounding volumes that is highly adaptive to the contents of the space. Thus, unlike a regular grid, the subdivision can be very fine in areas of high level of detail without compromising the representation of large open areas by a few large cells. Our method of building trees [Naylor 93], based on minimizing the expected cost of search operations, produces such trees, since large open regions are treated as being highly probable and will have short paths. This is completely analogous to Huffman codes, in which the number of bits assigned a code, i.e. the path length in the Huffman code binary tree, is inversely related to the probability of that code being used. Here, largeness is treated as being positively correlated to the probability of intersecting a region.

Given a tree representation of the environment, constructed off-line, together with a particular view, we find the subset of the environment within the view-volume by first constructing a Partitioning Tree representation of the view-volume [Naylor 92a]. Since we are using a tree for this, the view-volume can have any polyhedral geometry, and so need not be limited to a truncated pyramid. We also provide solid clipping; that is, the intersection of the view-volume with the solid environment will be displayed by polygons having the attributes of the material with which the view-volume is intersecting. We use the general tree merging machinery for view-volume clipping/culling. However, we do not produce a new tree corresponding to intersection of the environment with the view-volume. Rather we combine the view-volume intersection with the visibility priority traversal so that the polygons are transmitted to the polygon drawing stage as the intersection operation proceeds, thus obviating the need for creating an intermediate clipped tree.

Another very effective but very simple method of reducing the computational requirements is to combine simulation of fog with placement of the far clipping plane.