3D Scene Annotation for Efficient Rendering on Mobile Devices

Siak Chuan Tan, Binh Pham, Jinglan Zhang and On Wong
{A2.tan;b.pham;jinglan.zhang;o.wong}@qut.edu.au

Abstract

This paper presents a new approach for efficient 3D rendering on mobile devices, where selective rendering can be achieved with the help of 3D scene annotation. By taking advantage of first person environments in most 3D applications, we are able to annotate the flooring details of the 3D space. This allows 3D environments to be interfaced using a higher level view of objects. With the higher level of scene understanding, it is possible to determine which 3D objects are not required for loading or rendering based on the viewer's location and its surrounding constraints.

1. Introduction

The capabilities of mobile devices have improved significantly since there were first introduced. There is a growing list of devices which supports hardware 3D acceleration. These devices range from Personal Digital Assistants (PDAs) to Ultra Mobile PCs (UMPCs). However, the limitation of resources in mobile devices has lead to unrelenting research into more efficient methods to render 3D graphical objects.

There are several 3D representations such as X3D, U3D and WPF XAMLs. Although some of these representations are developed with mobile devices in mind, mobile devices are still not capable of handling large amount of 3D rendering. In an attempt to solve this problem, several techniques such as compression and progressive transmission were developed.

3D applications employed in mobile devices are usually based on spatial environments. This is especially true for 3D applications and games such as Second Life mobile [1] and Doom, where highly efficient techniques are required to render walls and objects to reduce the amount of object rendering.

Selective rendering is a common solution for improving rendering efficiency, but there are two perspectives in selective rendering. One can selectively reduce the quality or the quantity of objects to render. Transcoding [2-4] is a common means of selective rendering, where the quality is reduced by distillation or content adaptation to suit the memory limitation of the destination device, thus improving rendering efficiency with an acceptable degraded quality. There are other methods such as progressive transmission and adaptive refinement [5-8] which will not be mentioned in this paper.

Another perspective in selective rendering is based on the quantity of objects to be filtered, where redundant objects out of the viewer's frustum are removed to reduce memory and computation requirement. There are several mechanisms such as Binary Space Partitioning (BSP) [9] and portal rendering [10] which perform such filtering features, and these mechanisms will be mentioned in later chapters.

This paper will investigate problems faced in efficient rendering of first person environments in mobile devices. Scenes definition is employed so that objects can be selectively filtered for efficient rendering. Current techniques which try to improve on its efficiency will also be analysed. We will then present an innovative approach which uses rich annotation of scenes, followed by implementation and analysis of results. This paper will then conclude with the results and future direction of this research.

2. Problems

The common dilemma with 3D object rendering is selectivity, and this paper will address selectivity by filtering away redundant objects. It is always desirable to render or load as few objects as possible, but the process to identify redundant objects is usually very computationally exhaustive. Alternatively, simple techniques can be developed for managing large amounts of data during an interactive walkthrough of an architectural model [11].

Both approaches have their own strength and weaknesses. There is a larger computation requirement to calculate which objects are within the viewer's frustum if rendered objects are to be pre-determined, but the advantage of this method is a lesser strain on the memory. However, there is a larger memory but lower computation requirement if all objects are

0-7695-3067-2/07 $25.00 © 2007 IEEE
DOI 10.1109/DICTA.2007.85

rendered regardless of necessity. Thus it is very difficult to determine which is more efficient. Using a concept that it might be more efficient to just render a subset of the objects without complex calculations, this research takes advantage of the strengths of both techniques by providing an easy means for filtering unnecessary 3D objects without the need for complex computation, thus allowing large scene scenarios to be implemented on mobile devices.

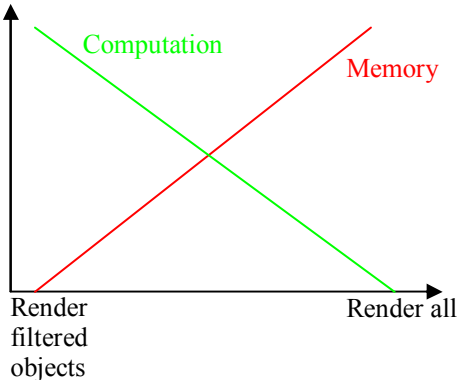

Figure 1: Resource extremes of different rendering methods

In order to achieve quantitative selective rendering, scenes must be managed so that objects can be determined if they are to be rendered based on the scenes. Current scene management approaches such as BSP and portal rendering are either too computationally or memory exhaustive, hence such extremities are not ideal for a mobile environment as mobile devices have limited resources both in memory and CPU. This research will address the issue of resource extremities.

Most methods are also based on static scenes and the manipulation of scenes in mobile devices is seldom considered. These scenes will need to be manipulable especially in collaborative applications. A renovator, for example, might need to amend some changes to a house layout remotely due to unforseen circumstances. Computer games however do not suffer from this issue as they simply assume that the scenes should not be changed during playtime. We present a simple definition mechanism which allows easy manipulation of scenes for mobile devices, without the need for exhaustive calculation or computation.

We will now analyse the advantages and disadvantages of BSP and Portal Rendering in greater detail.

2.1 BSP

Binary Space Partitioning (BSP) is a very common approach in 3D scenes management where a space is subdivided recursively by planes. These planes are represented by walls in a 3D scene. BSP planes are rendered from back to front in a similar way to a typical painter's algorithm, but do not suffer inter-overlapping problems the painter's algorithm faces due to its subdivided planes. BSP is also used in collision detection, ray tracing and shadow casting. An example of how BSP are converted into a BSP tree can be found in figure 2 [12].

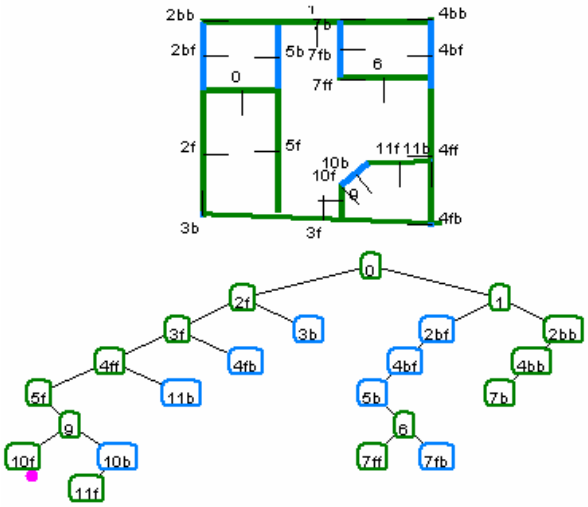

Figure 2: Complexity of BSP trees [12]

BSP's efficiency is based largely on how well it is balanced. A poorly balanced tree will have a very poor order of efficiency, and thus it usually requires a longer time to produce. BSP trees must also be static, which means that if the map is changed, the whole tree will be needed to be recalculated. There are several ways to address these issues [13] [14], but they tend to be overly complex and are unsuitable for mobile devices [15]. Lastly, due to the complexity of the tree, it is almost impossible for any human intervention via simple text editing programs.

In rendering terms, BSP renders its walls by painting through the whole tree from back to front. This means that there will be many segment of walls which are redundant but painted nonetheless. More importantly, the introduction of Z-buffering has solved a major part of BSP, though at a slighter higher computation cost.

Lastly, BSP has no ideal way of linking 3D objects contents to a spatial space. Additional calculations and rules must be made to determine what 3D objects content are considered redundant. The Doom game, for

example, needs to maintain a separate list of 3D sprite objects (called 'things') linking to particular sectors, and these objects are needed to be configured separately.

2.2 Portal Rendering

Neither BSP trees nor z-buffering, by themselves, avoid rendering surfaces that are completely hidden by other surfaces. Portal Rendering and Potentially Visible Sets (PVS) [10, 16] were first introduced by Jones in an attempt to solve the 'hidden line' problem. Portal rendering uses the concepts of '*cells/sector*' and '*portals*'. A cell is a region of space and a portal is a transparent region which connects cells. Cells also contain PVS which are essentially arrays of portals of neighbouring cells. Calculation of rendering objects are determined if the portal of the cell is 'stabbing' the portal of the neighbouring cell. This is performed recursively until there are no more potentially visible cells.

PVS calculations can be a very time consuming process, but it becomes practically free after calculations are made. We aim to cater for mobile applications which should allow scenes to be manipulable without costing too much resources. As such, the computational and memory requirements to regenerate PVS constantly do not suit mobile environments. There are also several constraints which might not be feasible in this paper's targeted mobile applications, including mandatory concave portals.

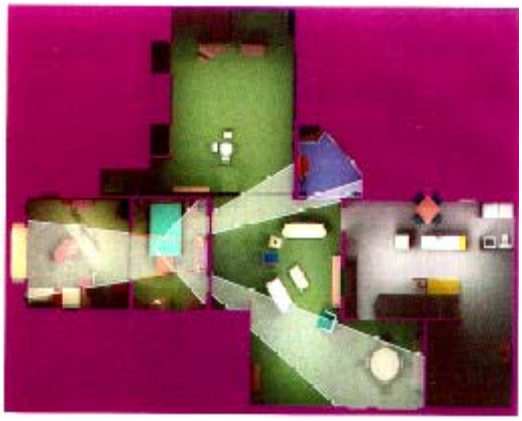

Figure 3: Overhead view of a house showing portal culling frustums.

3. Enclosed Space Definition (ESD)

We now present a simple but innovative approach to improve rendering efficiency, by using a simplified but rich definition and concepts of portal rendering. This can be achieved by creating an annotation mechanism for better understanding of the 3D scene and by doing so, filtering off redundant objects. The decision making process can then be significantly improved based on new knowledge obtained from the annotation of 3D scenes. Further filtering techniques for remaining objects can then be applied or every object can be rendered without filtering using the z-buffer techniques.

Enclosed Spaced Definition (ESD) improve rendering by means of a higher level of 3D annotation. Instead of partitioning space in a 3D scene by walls, we introduce rooms over walls. By using a room-based approach, we are able to determine redundant objects immediately by filtration. Instead of calculating the user's viewable frustum every time on every object to determine if they should be rendered or loaded, we can just check for the rooms and determine by enclosed room based logic.

The goal of ESD is to be slim but enable rich annotation of the scenes, and it targets mobile devices which have higher constraints in computing and memory resources. It is important that there is no complex pre-computation or calculation of metadata every time the structure is changed, but the information stored is yet rich enough to provide useful information for efficient rendering.

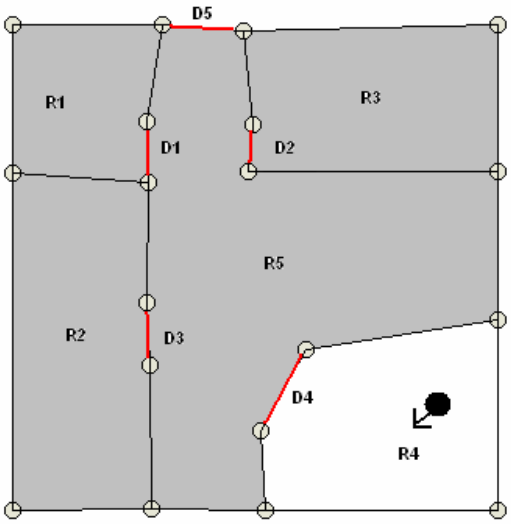

Figure 4: Enclosed Room Concept, 3D objects in grey regions are filtered.

This mechanism immediately reduces the resource requirement for loading or rendering a large number of objects. Only objects which are in the same room as the user are needed to perform more CPU intensive operations.

3.1 3D Annotation

ESD is annotated into an xml format. The tags and content of the annotation should be human-readable and easily understood. This ensures that ESD can be easily manipulated on different devices using simple notepad editors even when there are no specialised applications to edit it.

Although ESD probably has a larger memory requirement in best case scenarios, it has a constant performance in its annotation size and searching efficiency in any conditions. Consistency is a better trait when different devices with different computational constraints are involved.

Table 1: Efficiency of ESD vs. BSP

Definition size	*Best*	*Worst*
BSP	N	N^2
ESD	N + N/M	N + N/M
Searching efficiency	*Best*	*Worst*
BSP	N	N^2
ESD	N/M	N/M
*N = no of walls M = average no walls per rooms N/M = no of rooms		

There are 2 types of annotation in ESD. The first type consists of informative and redundant information for easy computation, and it is used mainly during the editing phrases of the ESD. Upon completion of editing, this annotation can be compacted into the second type in the form of a slim version for reduced memory upkeep. The two versions of annotation are inter-convertible. These two versions will be mentioned in the next sections.

3.2 Manipulation

Because ESD is compiled to ensure human readability, it is possible to edit ESD information with any text editors or software can be written to interface with the ESD files. Due to the way ESD is formatted, manipulation can be done with simple 2D oriented actions.

The editing versions consist of the following information in XML format:

- Nodes : The points which form the walls
- Walls: Connecting 2 nodes together to form a wall.
- Doors: Connecting 2 nodes together to create a door.
- Rooms: Consisting of Walls and Doors

There is redundant information retained with regards to the walls and doors which allows better control over how rooms are created. This additional information can also be used by other additional techniques if developed in future. For example, metadata may be created to pre-calculate rendering procedures before being compiled for deployment.

As ESD is based on nodes and complete walls, it is possible to determine rooms automatically based on the user's input. Thus there is no need for human intervention with regards to setting of room information. Instead of flooding, ESD uses a faster and innovative way of room detection. Wall tracing is employed in determination of rooms.

A random node is selected, and it is assumed that there are *x* rooms where *x* is number of links the node have. The first neighbour of the node is then selected, and rooms are created recursively. If 2 rooms happen to meet, the information on these rooms is combined and fused into a single room. Figure 5 illustrates the first two steps of the room detection algorithm. It will keep looping through all its nodes till every room is formed.

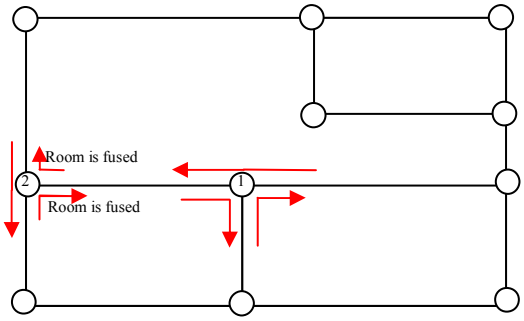

Figure 5: Step 1 and 2 of room detection technique.

3.3 Selective Rendering

ESD is intended to be deployed in multiple types of devices including mobile ones, and thus the definition for deployed files are intended to be streamlined and enhanced for rendering. Redundant information is removed or fused into the room tags. Rendering objects can then be filtered based on which rooms the user is in. No recalculation is needed every time the user moves as the ESD definition is rich enough to know where the user is and which rooms are they moving to every time they pass a door. (It is necessary to transverse along the tree every time for BSP).

There are a few different strategies which can be employed to improve efficiency in 3D objects in the same room. Rays can be shot to determine if objects are in the viewer's frustum for example. Determination of these viewable 3D objects sometimes requires too much computation, and is often ignored.

3D objects in adjacent rooms which are exposed by open doors can be tackled with two strategies. As 3D objects in adjacent rooms are not important to the user's view, they can either be rendered with poorer quality or a 2D image of the adjacent room can be pre-generated and textured over the door space. Some applications assume a closed door concept where the doors are assumed to be closed at all times.

Figure 6 illustrates a simple example of how 3D objects can be rendered in pseudo-code form. It is a very simple and clean process as the information of rooms is being retained in ESD.

```
Determine the room user is in
FOREACH 3D objects in rooms
        Render 3D object*
FOREACH room in room collection
        If room is joined to user's room
                IF user's view includes door
                    Render adjacent 3D objects**
```

* *Different strategies can be employed to filter desired object 3D objects if desired*

** *Different strategies for rendering of adjacent rooms can be employed if desired.*

Figure 6: Pseudo code for rendering.

Because the integrity of spatial information is maintained, it is possible to reverse the compacted definition back to the manipulation stage.

4. Implementation and Analysis

An implementation of ESD has been made both on desktops and mobile devices to show the practicability and efficiency of room identification and rendering with ESD. ESD is firstly created with simple mouse actions to form a room structure (Figure 7) before being parsed and allocated rooms automatically (Figure 8). Doors can be set up at this step as well. A XML definition is then compiled which is loadable in both mobile and desktop applications.

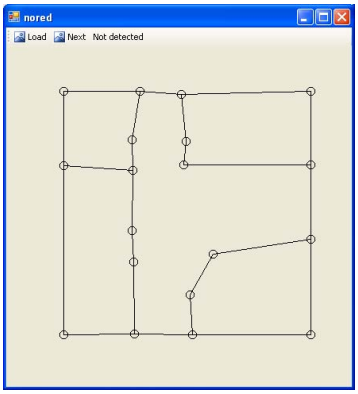

Figure 7: ESD being created with a desktop App.

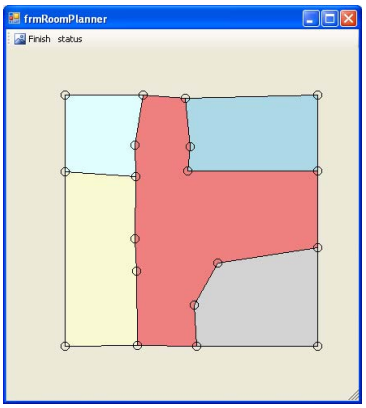

Figure 8: Rooms are automatically generated.

```
<ESD>
  <Nodes>
    <Node NodeID="..." Location="..." />
  </Nodes>
  <Walls>
    <Wall WallID="..." NodeID="..." ConnectedNodeID="..."
ConnectsToRoomID="..." />
  </Walls>
  <Doors>
    <Door doorID="..." FirstNodeID="..." WallID="..." />
  </Doors>
  <Rooms>
    <Room roomID="...">
      <Nodes>
        <Node NodeID="..." />
      </Nodes>
      <Doors>
        <Door DoorID="..." />
      </Doors>
    </Room>
  </Rooms>
</ESD>
```

Figure 9: ESD saved in XML format

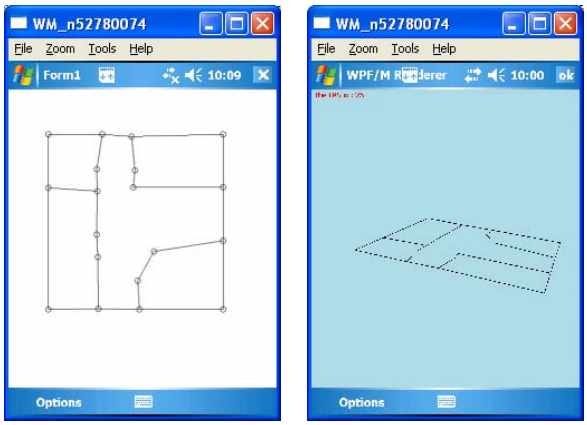

Figure 10: Mobile version of ESD which is capable of the same features of desktops due to its sleekness.

Using the samples in Figure 7-10 for analysis, a similar scene structure is obtained using BSP. BSP requires creating 22 Nodes, excluding door information. ESD on the other hand, requires 44 nodes with door information. A quick test is made to determine different requirements of BSP and ESD, worst case scenario implies 'unlucky' situations when searching have to be done through every nodes or node levels. From the figures in Table 2, one can say that ESD requires more memory in its definitions. In rendering efficiency, however, the ESD is able to filter numerous redundant rendering. Assuming that that there is an average of 5 objects in each of the 5 rooms, ESD is able to filter out the redundant objects due to the ability to tag objects to rooms in the definition.

Table 2: Analysis Comparison

	BSP	ESD
Nodes Required	22	44
Node Searching (Worst case)	8 levels	6 levels
Average Rendered Nodes	22	4
Rendered Objects* (Worst case)	20	5
*See assumption above		

5. Conclusion and Future Work

This paper has shown the usefulness of annotation for 3D scenes and how it can help in rendering efficiency. Using enclosed spaces as a quick guide for determination of rendering relevance, redundant 3D objects are filtered away very cheaply. Further filtering if desired can then be performed more efficiently as the test set becomes much smaller.

ESD can also be scaled to a larger perspective, as it can be used in a wider spatial scenario instead of just room-based scenes. Although some refinements on the annotation techniques must be made to allow multiple enclosed spaces in a larger scene, it is possible to allow map-like annotations using multiple definitions or even a database.

The preliminary results on this approach for selective rendering in mobile devices are promising, and further work can be done to introduce a 'fuzzy' visible set with improved definitions. Corresponding mechanisms can also be discovered for ray tracing and collision detection using the ESD approach.

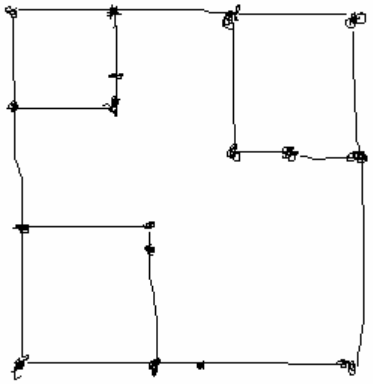

Figure 11: ESD can recognise and generate annotation with hand drawn maps by dots.

Considerations are also being made for manipulation efficiency. For example, hand drawn walls and doors can be recognised and automatically converted to ESD annotation (Figure 10). As ESD annotation consist of nodes and walls, it is possible to recognise large dots as nodes, link them up and convert such information into ESD. In addition, because ESD provides the framework for automatic room identification, the whole process from scanning to creation of ESD can also be automated.

7. References

1. White, D., *Second Life Spills Over into Mobile World*, in *Mobile Magazine*. 2007.
2. Fox, A., et al. *Adapting to Network and Client Variability via On-Demand Dynamic Distillation*. in *Seventh International Conference on Arch. Support for Programing Language and Operating System (ASPLOS-VII)*. 1996. Cambridge, MA.
3. Martin, I.M. *Hybrid Transcoding for Adaptive Transmission of 3D Content*. in *IEEE*

International Conference on Multimedia and Expo (ICME). 2002. Lausanne, Switzerland.

4. D'Amora, B. and F. Bernardini, *Pervasive 3D viewing for product data management.* Computer Graphics and Applications, IEEE, 2003. **23**(2): p. 14-19.
5. Dürst, M.J. and T.L. Kunii. *Progressive transmission increasing both spatial and gray scale resolution*. in *International Conference on Multimedia Information Systems '91*. 1991. Singapore: MacGraw-Hill.
6. Chande, V. and N. Farvardin, *Progressive transmission of images over memoryless noisy channels.* Selected Areas in Communications, 2000. **18**(6): p. 850-860.
7. Stollnitz, E.J., T.D. DeRose, and D.H. Salesin, *Wavelets for computer graphics: A primer, part 2.* IEEE Computer Graphics and Applications, 1995. **15**(4): p. 75-85.
8. To, D., *An Adaptive Multiresolution Method for Progressive Model Transmission.* Presence, 2001. **10**(1): p. 62-74.
9. Fuchs, H., Z.M. Kedem, and B.F. Naylor, *On Visible Surface Generation by A Priori Tree Structures.* ACM Computer Graphics, 1980. **14**(3): p. 124-133.
10. Jones, C.B., *A new approach to the 'hidden line' problem* The Computer Journal 1970. **14**(3): p. 232-237.
11. Funkhouser, T.A., C.H. S´equin, and S.J. Teller., *Management of large amounts of data in interactive building walkthroughs.* 1992 Symposium on Interactive 3D Graphics, 1992. **25**(2): p. 11 - 20.
12. Lewis., P.J. *BSP Trees: An interactive demonstration of Binary Space Partitioning Trees*. [World Wide Web] 2007 [cited 2007 23 July]; Available from: http://www.symbolcraft.com/graphics/bsp/index.php.
13. Chrysanthou, Y., *Shadows for 3D Interaction and Animation*, in *Computer Graphics*. 1996, University of London. p. 125.
14. Torres, E. *Optimization of the binary space partition algorithm (BSP) for visualization of dynamic scenes*. in *EuroGraphics 1990*. 1990. Montreaux, Switzerland.
15. Naylor, B.F. *Interactive solid geometry via partioning trees.* in *Graphics Interface'92*. 1992. Vancouver, Canada.
16. Luebke, D.P. and C. Georges. *Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets*. in *1995 Symposium on Interactive 3D Graphics*. 1995. Monterey, CA: ACM Press.