Frontier Sets: A Partitioning Scheme to Enable Scalable Virtual Environments

A. Steed and C. Angus

Department of Computer Science, University College London, Gower St, London, WC1E 6BT, United Kingdom

Abstract

We present a new spatial partitioning scheme called frontier sets. Frontier sets build on the notion of a potentially visible set (PVS) [ARB90, TS91]. In a PVS a world is sub-divided into cells and for each cell all the other cells that can be seen are computed. Frontier sets represents regions of mutual invisibility. One frontier in a frontier set considers pairs of cells, A and B. It lists two sets of cells, F_{AB} and F_{BA}. From no cell in F_{AB} is any cell in F_{BA} visible and vice-versa.

We have used frontier sets to investigate peer-to-peer networking schemes for networked virtual environments. Preliminary investigation of simulations within the Quake II game engine shows that frontiers have significant promise and may allow a new class of scalable peer-to-peer game infrastructures to emerge.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Multi-user online simulations and games pose a number of challenges to systems designers [SZ99]. One challenge is that of scaling the system to large numbers of users. Two problems are key: the simulation or game mechanics may require computation of $\Omega(N^2)$ in the number of participants; and maintaining the game state potentially involves $\Omega(N^2)$ remote interactions across a network. Simulations of more than a few tens of participants have usually required dedicated networks and assumptions about resource availability and interaction capabilities of the simulation client. Alternatively they have had to limit the interactivity of the simulation or use a non-real time simulation.

One strategy that is often used to increase the number of participants are area of interest management schemes that attempt to identify users that do not need to communicate [MBD00]. A common strategy is to allocate each participant to one of a set of pre-determined regions of space and then only enable communication between participants who share the same region.

A computer graphics technique that is related to this problem is visibility culling [CCSD03]. One technique that is often used in networked games is potentially visible sets (PVS) [ARB90, TS91]. A PVS can be used to exclude a pair of entities from consideration for simulation purposes because they are not mutually visible [Fun95]. However this visibility must be evaluated every time the entities concerned move.

In this paper we introduce a new data structure called a frontier set. Frontier sets take a PVS structure and generate a new structure that allows pairs of entities to negotiate a criterion that either can test very rapidly to ensure that no interaction is necessary between them.

In Section 2 we introduce the frontier sets data structure and give examples of how frontiers can be used. In Section 3 we give a simple method for building frontiers. In Section 4 we then describe how we have applied frontier sets to the game Quake II [Id 97]. We then conclude and describe potential avenues for further research.

2. Frontier Sets

2.1. Potentially Visible Sets

A PVS exploits the fact that in many virtual environments, from a specific point of view much, if not most, of the rest of the environment may be occluded [ARB90, TS91]. If the

environment is divided in to regions of space (or cells) then for any cell, it will be possible to identify openings (or portals) through which other cells can be seen. For any cell, it is possible to explicitly compute which other cells are visible from that cell, because if a cell is visible then there must be a line of sight through all the portals between them. Figure 1 shows a simple example of a world comprised of cells and the PVS of one of the cells.

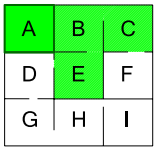

Figure 1: *An example of a potentially visible set. From cell A, only cells B, C and E are visible.*

The calculation of a PVS can be done automatically or by hand for smaller models. For the purposes of this paper we assume that a PVS already exists. We will also assume that the PVS is symmetric; that is, if cell A can see cell B, cell B can see cell A. Assymetries can be dealt with by minor extensions to the algorithms we will describe.

2.2. Frontier Definition

PVSs are typically based on a cell-based sub-division of the world. A frontier is a pair of sets of cells that are mutually invisible. We can find a frontier given two cells A and B. The frontier will be two regions F_{AB} and F_{BA} such that no cell in F_{AB} is visible to a cell in F_{BA} and vice-versa. $F_{AB} \cap F_{BA} = \emptyset$, otherwise any cell in the intersection must be visible from both F_{AB} and F_{BA}.

We will use the term frontiers to refer to a pair of such sets, or to only one of the sets when the meaning is clear. The complete set of frontiers for a whole world will be referred to as a frontier set.

There are potentially many frontiers for a pair of cells. No frontier will exist between two cells if the two cells are visible to each other. If B is not in the PVS of A, then frontiers can be initialized with $F_{AB} = A, F_{BA} = B$. In Figure 2a we see the frontiers for two cells A, I. No frontier would exist between cells D and H because they are mutually visible.

2.3. Example Usage

Consider two users moving around the environment depicted in Figure 2a. If Anne is in cell A and Bob is in cell I at time t_0 then a frontier can be established. $F_{AI} = A,B,C,F, F_{IA} = G,H,I$ as shown in Figure 2a. If Anne remains in F_{AI} and Bob remains in F_{IA} then they can never see each other. If this were a networked virtual environment this would mean that if Anne and Bob both exchanged location information at t_0 they would not have to send any further updates.

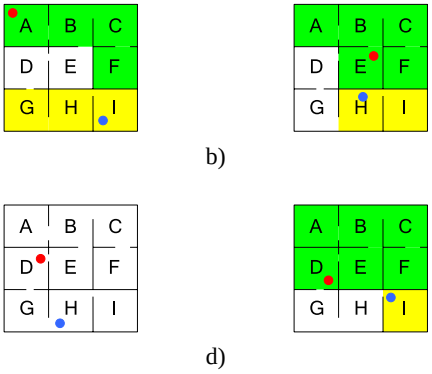

Figure 2: *An example of frontiers in use. a) Users Anne and Bob are in cells A, I respectively. A frontier exists $F_{AI} = \{A,B,C,F\}, F_{IA} = \{G,H,I\}$. b) A frontier exists $F_{EH} = \{A,B,C,E,F\}, F_{HE} = \{H,I\}$. c) No frontier exists because cell D can see cell H. d) A frontier exists $F_{DI} = \{A,B,C,D,E,F\}$ and $F_{ID} = \{I\}$.*

If either Anne or Bob leaves their frontier, then one of two situations arises: either a new frontier can be set up between the two cells Anne and Bob are currently in, or they can see each other. In Figure 2b, at time t_1, Anne leaves the F_{AI} established at t_0 and enters cell E. At this time Bob is in cell H. There is a new frontier $F_{EH} = \{A,B,C,E,F\}, F_{HE} = \{H,I\}$.

At time t_2, Figure 2c, Anne moves to cell D. There is no frontier between cells D and H because the cells are mutually visible. At time t_3, Figure 2d, Bob moves to cell I. A frontier can be established $F_{DI} = \{A,B,C,D,E,F\}$ and $F_{ID} = \{I\}$.

Pseudo-code for an algorithm to implement this within a networked simulation is given in Figure 3. A client calls SendNetworkUpdate each frame. Independently of this, they can receive an update from the network which triggers ReceiveNetworkUpdate. The main job of SendNetworkUpdate is to establish if, since the last frame, this client or the other client have left the agreed frontier. If they have, they get rid of the current frontier. If there is no current frontier then it sends an update to the other client. Finally it tries to establish a new frontier with the current cells for this client and the other client.

3. Creating Frontier Sets

There are many different potential ways of constructing frontiers. In this section we give a simple algorithm that attempts to grow regions that are balanced in size and as large as possible. In our scenario the best frontiers will be those

```
AttemptEstablishFrontierWithOther(cellThis, cellOther)
    /* frontiers[x][y] is the set F_xy and vice versa. */
    IF frontiers[cellThis][cellOther] ≠ ∅ {
        frontierThis = frontiers[cellThis][cellOther]
        frontierOther = frontiers[cellOther][cellThis]
    }
}
ReceiveNetworkUpdate() {
    cellOther = GetOtherParticipantCurrentCell()
}
SendNetworkUpdate() {
    cellThis = GetCurrentCell()
    /* If we or other have moved out of frontier,
       remove it. */
    IF (cellThis ∈ frontierThis) OR
        (cellOther ∈ frontierOther)
        frontierThis = frontierOther = ∅
    /* If no current frontier, send a packet and try to
       establish one. */
    IF frontierThis = ∅ {
        SendNetworkUpdateToOther()
        AttemptEstablishFrontierWithOther(cellThis, cellOther)
    }
}
```

Figure 3: *Pseudo-code for the use of frontiers to limit packet update information for two moving participants.*

```
GenerateFrontiers(cells) {
    FOR EACH PAIR (a, b) FROM cells {
        frontierA = frontierB = ∅
        /* A frontier can only be set up if a and b
           cannot see each other. */
        IF b ∈ PVS(a) {
            frontierA = {a}; frontierB = {b}
            pvsFrontierA = PVS(a); pvsFrontierB = PVS(b)
            /* GetLeaves() returns all cells adjacent to x. */
            growA = a.GetLeaves(); growB = b.GetLeaves()
            WHILE NOT (growA.Empty() AND
                growB.Empty()) {
                IF NOT growA.Empty() {
                    nextCell = growA.Pop()
                    IF nextCell ∋ pvsFrontierB) {
                        frontierA = frontierA ∪ nextCell
                        pvsFrontierA = pvsFrontierA ∪ PVS(nextCell)
                        FOR EACH c ∈ nextCell.GetLeaves()
                            IF (c ∋ frontierA) AND (c ∋ growA)
                                growA.Push(c)
                    }
                IF NOT growB.Empty()
                    /* Repeat block above, swapping A and B. */
                }
            frontiers[a][b] = frontierA; frontiers[b][a] = frontierB
        }
    }
}
```

Figure 4: *Pseudo-code for the construction of frontiers with symmetric PVS.*

that will be expected to last as long as possible when used to cull packets on the network.

Figure 4 gives pseudo code for the region growing approach to the creation of frontiers. It is based on a standard cells and portals data structure. Cells are linked in an adjacency graph, with cells being nodes and portals being the links between nodes. If the PVS is symmetric, to add a cell C to a frontier F_{AB}, C must not be visible to any cell in F_{BA}. Likewise, a cell C can only be added to F_{BA} if it is not visible to any cell in F_{AB}. Key to this is maintaining an aggregate PVS of a set of cells. As a cell is added to a frontier, the PVS of the frontier is merged with the PVS of the cell being added.

The algorithm alternately adds a cell to F_{AB} and F_{BA}. It proceeds by following the adjacency graph in a breadth first manner from A and B. Figure 5 shows how this works for the creation of the frontiers given in the example from Figure 2. The final frontiers are $F_{AI} = \{A, B, C, F\}$ and $F_{IA} = \{G, H, I\}$.

4. Testing Frontiers in QuakeII

To test the frontier concept we have implemented a simulation of frontiers within the game QuakeII [Id 97]. This platform was chosen because it is a very popular example of an online, fast-paced action game, and also because the source code is available under the GNU Public License. Quake II uses a client-server model for distributing game state amongst the players.

Each game level within the Quake II game has a PVS. This is primarily for speeding up rendering, though in in multi-user games it is also used at the server to cull packets that do not need to be relayed to a client because that client can not see the entity concerned.

Typical game levels in Quake II have between 1000 and 3000 cells. In our initial tests we have pre-computed the complete frontier set. Because this requires $O(N^3)$ space in the number of cells (i.e. for each pair of cells, we classify all cells into in one half of the frontier or in neither), we first compress the PVS using the scheme of [vdPS99]. We find that by reducing typical game levels to just 256 cells we do not lose much of the visibility information.

Figure 6 shows an example frontier pair. We can use frontiers as the basis of a peer-to-peer networking scheme as suggested in Section 2.3. A peer-to-peer scheme is desirable because it is lower latency than a client-server system. A *perfect peer-to-peer* scheme would only send information between two players if and only if they could see each other. Of course, such a scheme is impossible to implement because one must know whether the other user is to know not

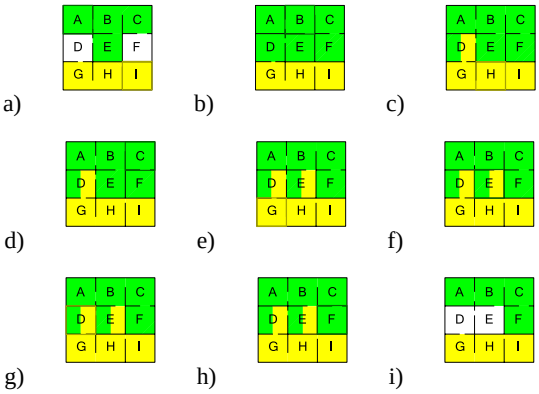

Figure 5: *An example creation of a frontier. Solid coloring shows the frontiers and hatching the corresponding PVS. a) Since A and I are not visible to each other, F_{AI} and F_{IA} can be initialized with A and I respectively. PVS F_{AI} is set to PVS A, PVS F_{IA} is set to PVS I. b) We add B to F_{AI} because B is not in PVS F_{IA}. We add D and F to PVS F_{AI}. c) We extend F_{IA} with cell H. d) We extend F_{AI} with C. e) We add G to F_{IA} . f) We attempt to extend F_{AI} with E, but can not because it is in PVS F_{IA}. g) Likewise we cannot add cell D to F_{IA}. h) We add cell F to F_{AI}. i) The final frontiers.*

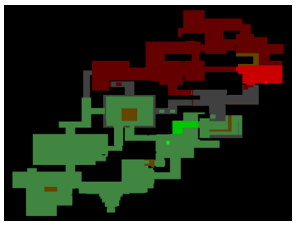

Figure 6: *Example of the use of frontiers in the Quake II level q2dm4. The players represented by red and green dots started in the bright red and green cells. As long as they stay in the dark red and dark green regions they do not need to communicate.*

to send a packet. Frontiers allow us to use a peer-to-peer scheme and still retain scalability.

We recorded several QuakeII game sessions with an average of 16 users online. In simulations of peer-to-peer networking schemes on one session on the QuakeII game level q2dm4, we found that a frontier-based system would require each client to send 3.1 packets per client per game frame on average, whereas a perfect peer-to-peer system sent 2.8 packets per client per game frame. This is compared with 15.6 for a naive peer-to-peer scheme where each user sends to every other user. Surprisingly frontiers were also more efficient than a simple client-server scheme without packet aggregation at the server which produced 3.8 packets per client per frame.

5. Conclusions

We have introduced frontier sets, a new data structure that represents mutually invisible regions of space. We have shown a simple algorithm for constructing frontiers and how these frontiers can be used to enable scalability in peer-to-peer networking systems. We have then given some brief analysis of frontiers using data from QuakeII sessions.

Our immediate work is in implementing a peer-to-peer system on the frontier algorithm. There are also several extensions that can be made to the processes for construction of frontiers. We note that we have tried to make the frontiers as large as possible, but simpler descriptions might require less storage space. We also note that we have described frontiers for every pair of cells, but the frontiers themselves will be highly compressible because if a cell C is in a frontier F_{AB}, then it is likely that cell A would be in F_{CB}.

References

[ARB90] AIREY E. J. M., ROHLF J. H., BROOKS JR. F.: Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics 24*, 2 (1990), 41–50. (Proc. ACM Symposium on Interactive 3D Graphics).

[CCSD03] COHEN-OR D., CHRYSANTHOU Y., SILVA C., DURANT F.: Survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (2003), 412–431.

[Fun95] FUNKHOUSER T. A.: Ring: A client-server system for multi-user virtual environments. In *1995 Symposium on Interactive 3D Graphics* (Apr. 1995), pp. 85–92.

[Id 97] ID SOFTWARE: Quake II, 1997. www.idsoftware.com/games/quake/quake2/.

[MBD00] MORSE K. L., BIC L., DILLENCOURT M.: Interest management in large-scale virtual environments. *Presence: Teleoperators and Virtual Environments 9*, 1 (2000), 52–68.

[SZ99] SINGHAL S., ZYDA M.: *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.

[TS91] TELLER S. J., SEQUIN C. H.: Visibility preprocessing for interactive walkthroughs. *Computer Graphics 25*, 4 (1991), 61–90. (Proc. SIGGRAPH 91).

[vdPS99] VAN DE PANNE M., STEWART A. J.: Effective compression techniques for precomputed visibility. In *Proc. Eurographics Rendering Workshop* (June 1999), pp. 305–316.