jMonkey Engine 3.0 – PVS

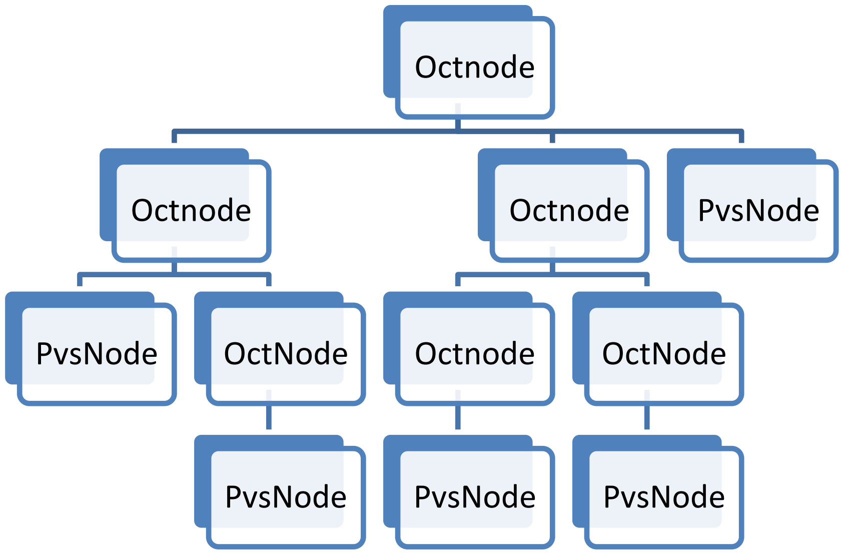

Figure 1-1: All PvsNodes are leaf nodes.

Every node in the tree has a bounding box, the camera's PvsNode is determined by going down the tree and finding which bounding box contains the camera.

Its possible camera is inside more than one PvsNode. In that case, the intersection of the PVSets of all nodes the camera is in is used.

If camera changed PvsNode (exited from its current PvsNode)

1. Find PvsNode where camera resides (likely to be close to the parent of the old one...)
2. Let PVSet = PvsNode.getPvs()
3. Let DynSet = all dynamic octnodes at depth X that intersect with PVSet
4. Do steps below

Camera stayed same PvsNode

1. Let PVSetR = cam.intersectsAll(PVSet)
2. Render PVSetR
3. Let Cont_F = cam.containsAll(DynSet)
4. Let Int_F = cam.intersectsAll(DynSet)
5. Render Cont_F
6. For each node under Int_F, check intersection with camera, if yes, render it.

Dynamic Object Octree (DOO)

This particular variant of the Octree is used for storing “moveable” and animated objects. The Boolean “Moveable” must be set on the object in Blender3D for it to be added to DOO.

The DOO exhibits the following properties:

It is Loose; nodes can expand or retract as objects get added or removed.

It is generated on the fly as objects are added.

Empty nodes (nodes without any objects) are removed automatically

Once a node in the DOO has been found to be intersecting with the camera, intersection with the current PVS set is also checked. This provides occlusion culling for all dynamic objects for free.

All objects inside the DOO contain a BIH tree generated for the mesh; this is used for ray casting and collision detection.

Static Triangle Octree (STO)

The STO is the primary variant of Octree used in jME3.0. The STO stores PVS information.

This variant of the Octree is triangle based; no triangle clipping is done so the Octree must be loose.

There cannot be an area inside the STO that is not covered by a PvsNode. All leaf nodes must be PvsNodes. To support this requirement, the STO nodes can expand, but cannot contract, thus the STO is only a half-loose Octree.

The STO will contain (many) empty nodes; these nodes are PvsNodes so they contain visibility information for empty space. Note that empty nodes are NOT subdivided into more empty nodes. This reduces the number of empty nodes significantly.

A specific criterion is used for adding a triangle to an STO octnode:

- If the triangle is contained entirely by a certain child of the octnode, the triangle is added into that child.
- If the triangle intersects several octnodes, an algorithm is used to determine the addition cost into each child octnode, the child that has the lowest addition cost is chosen. A triangle can only exist in a single octnode. If the triangle intersects the octnode it is added to, the bounding box of the octnode is expanded to contain the triangle. See Figure 1-2.

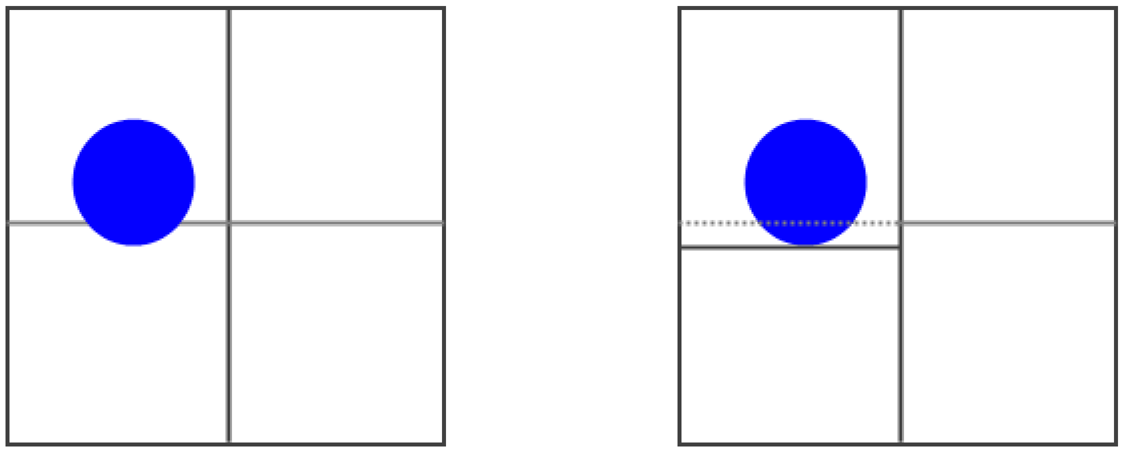

Figure 1-2: (a) on the left, the Octree has been split. (2) On the right, the bounding box of the top left child has been expanded downwards to contain the blue sphere. The dashed line indicates the split of the bottom left node which has been left as-is.

Generation of PVS for STO requires use of conventional PVS algorithms. jME3 will use hardware occlusion queries to generate PVS data.

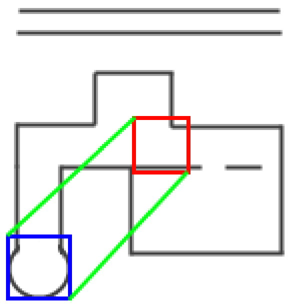

Figure 1-3.

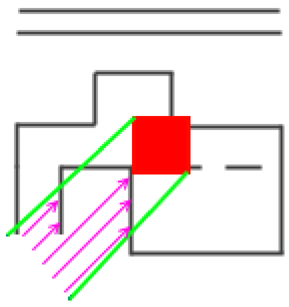

Figure 1-4.

To compute visibility, we create a camera frustum that will contain both the current node and test node (figure 1-3). The entire scene is rendered, except for the contents of the blue and red node (figure 1-4).

After the entire scene has been rendered, the test node is rendered as a bounding box, with occlusion query turned on. This tells us how many pixels of the test node are visible in the camera. If no pixels are visible, that means the test node cannot be seen from the current node, and therefore the test node is not added to the Potentially Visible Set (PVS). In Figure 1-4, we see that none of the purple rays hit the test node.