Fractal Block World
Old Stick-Ball Caves

June 8, 2026

1 What is this

This is documentation that was in the creation manual for how to create "stick-ball" caves via the Lua-to-C Chunk Generation API functions. It has been moved into this document.

2 Relevant Parts of the Chunk Generation Lua-to-C API

```
//-----------------------------------------------
//          The Exotic "Cave" Block Functions
//-----------------------------------------------
void caves_start();
void caves_set_5x5x5();
void caves_set_num_nodes(
    float min_nodes, float max_nodes);
void caves_set_nodes(
    float frac_large_node,
    float small_node_min_rad,
    float small_node_max_rad,
    float large_node_min_rad,
    float large_node_max_rad);
void caves_set_edges(
    float max_edge_dist,
```

```
    float frac_large_edge,
    float small_edge_min_rad,
    float small_edge_max_rad,
    float large_edge_min_rad,
    float large_edge_max_rad);
void caves_end();
bool caves_close_to_node(
    int x, int y, int z);
INFO caves_close_to_node2(
    int x, int y, int z);
bool caves_close_to_edge(
    int x, int y, int z);
```

3 Basic Usage

To start creating the caves, you call `caves_start`. There are then a variety of functions you can call to create the stick and ball style caves.

By default, all sticks and balls in the surrounding 3x3x3 chunks will be created. The engine accomplishes this by having a way to create all sticks and balls within any given chunk (using the chunk path of the chunk as input to the pseudo random number generator). So if there is a node in one chunk and a node in an adjacent chunk, and if there is an edge between the two nodes, we can carve out a shaft surrounding the edge that goes between the two nodes.

If you have a node in one chunk A and then a node in a chunk B that is 2 chunks away from B, by default there is no way to have an edge from the first node to the second one. However, if you call `caves_set_5x5x5` after `caves_start`, but before `caves_end`, then all sticks and balls in the surrounding 5x5x5 chunks will be created. (Note: the 5x5x5 mode is slower for the computer than 3x3x3 mode). Nodes that are two chunks away from each other can then be connected by edges.

The function `caves_set_num_nodes` specifies how many nodes should be created in each chunk. You specify the min and the max number, and then for each chunk the actual number will be chosen at random between the two. Specifically, the min and max values are floats, a float is chosen at random between these two, and then the integer floor of that is used.

There are two types of nodes: small ones and large ones. You specify the

radii of these two types of nodes by calling the function `caves_set_nodes`. You specify the min and max radius of a small node, and then each small node will have a radius randomly picked between the min and max. You also specify the min and max radius of a large node. You also specify the fraction of nodes that are large.

Each edge has a radius (so each edge is really a tube, or cylinder). You call `caves_set_edges` to set the radii of these edges. There are two types of edges: small and large. You specify the min and max radius of small edges. You do the same for the large edges. You also specify the fraction of edges that are large versus small. Finally, you specify the max edge distance. If 3x3x3 mode is being used and two nodes are in adjacent chunks (or the same chunk), then they will be connected by an edge iff the distance between them is less than the max edge distance. If 5x5x5 mode is being used, then the same is true but now for nodes that are in chunks that are at most 2 chunks apart.

When you are finished specifying the caves, call `caves_end` to finish creating the maze.

3.1 Querying the Caves: Part 1

```
bool caves_close_to_node(int x, int y, int z)
bool caves_close_to_edge(int x, int y, int z)
```

The next step is iterating through every block position P in the chunk to see if P is inside a node or an edge. You then "carve out" all such block positions. You call "caves_close_to_node" and "caves_close_to_edge" to see if a block position is inside a node or edge.

3.2 Example

Here is the main function of a chunk generation script that creates some caves.

```
function p.__main()
    --The chunk is by default solid to start with.
    set_default_block("block_s")

    --Creating the stick-and-ball data
```

```
--structure for the caves.
caves_start()

--Making the cave connect
--together nodes that are at most 2 chunks apart
--(as opposed to 1 chunk apart).
--Setting the 5x5x5 option makes cave creation slower.
caves_set_5x5x5()

--Between 2 and 3 nodes per chunk (random).
caves_set_num_nodes(2.0,3.99)

--Only 0.01 of nodes are large, the rest are small.
--Small nodes have radius between 3.0 and 4.3, and
--large nodes have radius between 17.5 and 18.0.
caves_set_nodes(0.01, 3.0,4.3, 17.5,18.0)

--Max dist between two nodes that can be connected
--with an edge is 20.0 (to go beyond 16.0 for this
--number, must call caves_set_5x5x5).
--No edges are large.
--Small tubes (around edges) have radius between 1.0 and 2.0.
--Large tubes have radius between 7.0 and 8.0.
caves_set_edges(20.0, 0.0, 1.0,2.0, 7.0,8.0)

caves_end()
--Now, the stick-and-ball data structure
--for the caves has been created.

--****************************************
--****************************************
--****************************************

for x = 0,15 do
for y = 0,15 do
for z = 0,15 do
    local close = caves_close_to_node(x,y,z)
        or caves_close_to_edge(x,y,z)
```

```
        if close then
            --Carving out the position.
            set_pos(x,y,z,"block_e")
        end
    end
    end
    end
end
```

3.3 Querying the Caves: Part 2

```
INFO caves_close_to_node2(int x, int y, int z)
```

A common task it to add one item to the center of each node in the stick and ball caves. To do this, you can use the "case_close_to_node2" function which returns an object data with the following members:

```
data.close      //the result of case_close_to_node
data.which_node //which nodes is closest to
data.dist       //distance to closest node
data.is_big     //whether the nodes is closest to is "big"
```

Each node has a number. You can keep track of this so you only place one item per node. Here is a modification of the cave creation code from this section that only puts one "gold_10" item in the center of each node. This code should occur after the "caves_end()" function.

```
    --A table, whose keys are the IDs of the nodes
    --that have a power up placed in them.
    local filled_nodes = {}

    for x = 0,15 do
    for y = 0,15 do
    for z = 0,15 do
        local close_to_edge = caves_close_to_edge(x,y,z)

        local data = caves_close_to_node2(x,y,z)
        local close_to_node = data.close
```

```
    --Carving the position if need be.
    if close_to_node or close_to_edge then
        set_pos(x,y,z,"block_e")
    end

    --Adding gold in the center (for each node).
    if close_to_node then
        local which_node = data.which_node
        local dist = data.dist
        local is_big = data.is_big

        if (dist < 1.5) and
            filled_nodes[which_node] == nil
        then
            filled_nodes[which_node] = true
            add_bent(x,y,z, "bent_gold_10")
        end
    end
end
end
end
```