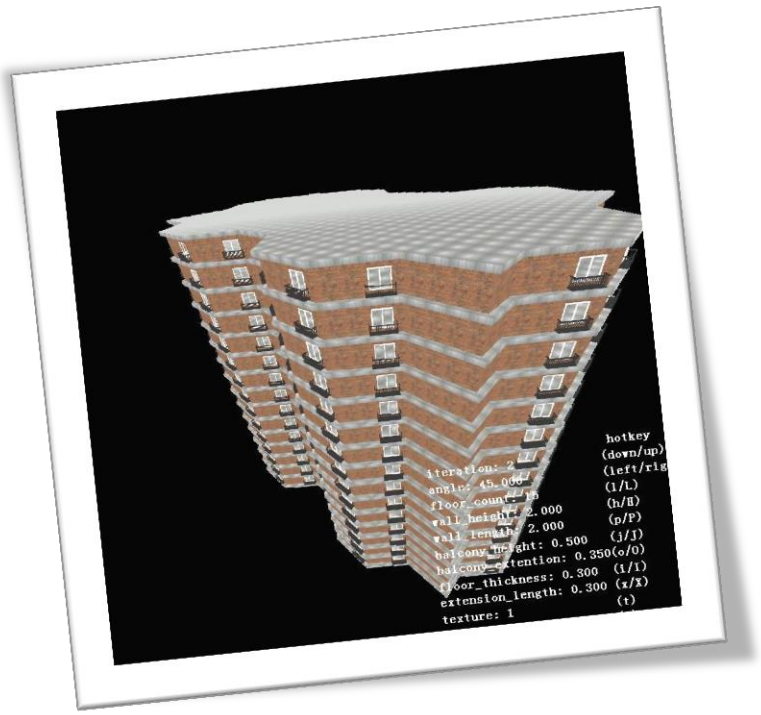
MSc CGE – MathsGfx1 – Coursework2:
Building Generation with 3D L-System

Team Members: Madina Berkaliyeva, Wanganning Wu.

Number of pages: 15

Date of Submission: 31/01/2014



Introduction

A Lindenmayer System (L-System) is a parallel rewriting system that can be used for procedural buildings generation. The most basic L-System must consist of symbols that are used in string creation, a set of production rules and an initial axiom from which the construction string begins. In order to make use of this string, some operations for translating the constructed string into a geometric form (such as angle and branching symbols) are required. Given the axiom and based on the production rules, the system repetitively replaces the symbols with the respective string of symbols.

There are different techniques used for procedural L-System building generation, some of them generate the building mass using volumes (boxes, cylinders, pyramids and etc.) as the base shapes and then they generate façade texture to be applied onto the buildings (Figure 1). The other way of

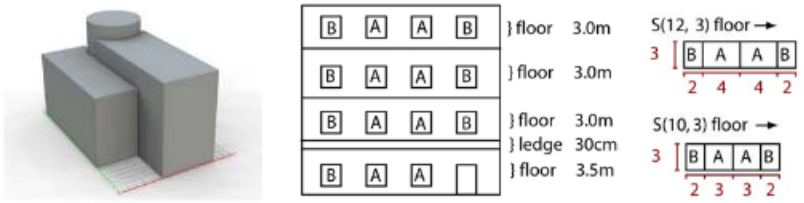


Figure1. Building geometry (left) and façade (right)

generating buildings using L-Systems is to use meshes that represent walls as the base shapes; this method is used in Houdini FX software. Figure 2 demonstrates the building generation process using walls and angles:

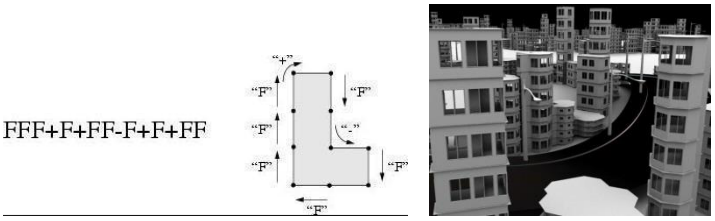


Figure2. Procedural wall generation (left) and produced building example (right)

The important feature of using this way of building generation is that it is possible to specify in the rules the windows position, window size and the placement of the balcony. Having these features in the production rules will make buildings different from each other.

Objectives

The goal of this work is to produce a program that procedurally generates 3D Lindenmayer System Buildings based on Octet Framework. The main tasks of this work are:

- Load a configuration files from the text file and add the hotkeys to control the produced buildings;
- Implement wall generation, windows and balconies based on L-System rules;
- Triangulate the concave polygon of the floor board;
- Manually create various rules that will create visually appealing buildings.

Team Work

As the team we have first researched on the topic in order to find the existing techniques of applying L-Systems to procedurally generate buildings. Once we found the idea we want our work to be based on, we split the work into smaller tasks and each of the team members implemented one of the task at a time. Since a lot of tasks were dependent on other tasks, we held regular meetings and discussed the problems and progress so far and helped each other. Below are the lists of tasks for each team member:

Tasks Assigned: Wanganning Wu

- Implementing the basic 3D stochastic L-System class to support the building generation;
- Implementing wall generation based on L-System rules;
- Implementing the Ear-Clipping algorithm to triangulate building floor whose shape can be a concave polygon;
- Implementing a doubly-linked and circular list as a data container to optimize the Ear-Clipping algorithm;
- Implementing a font-helper class to output text information of hotkeys onto the screen;
- Implementing a camera control class to enable viewing buildings in the scene with mouse or keyboard.

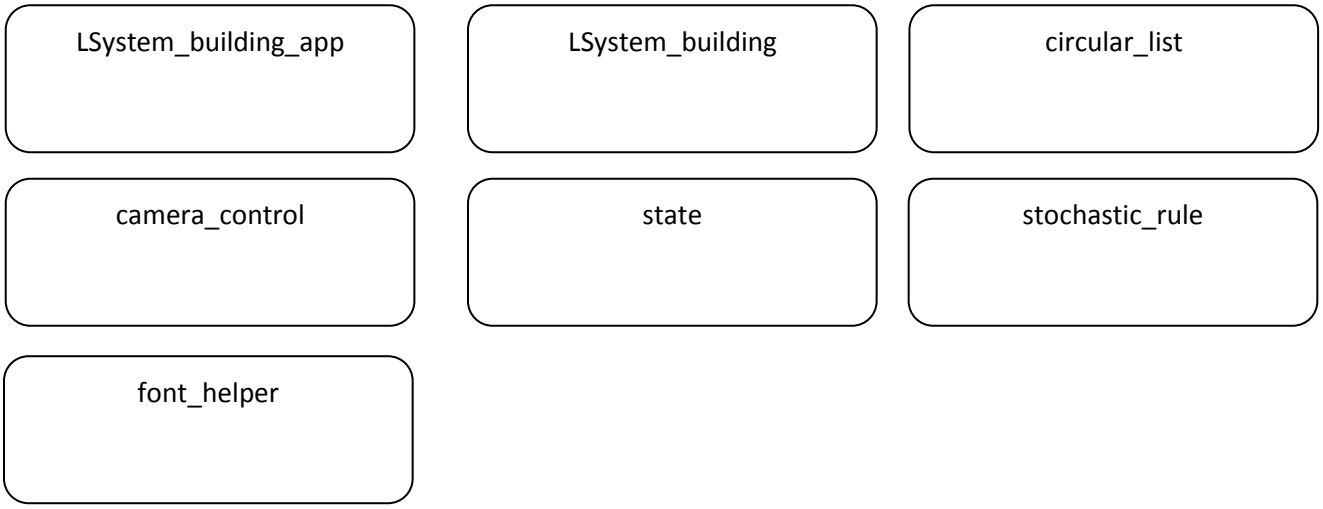
Tasks Assigned: Madina Berkaliyeva

- Implementing a window generation method in the specified wall given the size by recreating the wall meshes;
- Implementing a window frame generation method around each window by generating additional meshes;
- Implementing balcony generation function;
- Implementing hotkeys functionality that allows user interaction with the building parameters;
- Creating different rules that are generating visually appealing buildings.

Implementation

Class

This section describes the classes that have been implemented and their main functionalities. Below are the classes implemented:



- **class LSystem_building_app**

  This is a custom app class derived from class app that maintains shader instance, font helper instance, LSystem_building instance, camera_control instance, responds keyboard and mouse messages, prepares rendering states and renders meshes in the LSystem_building instance.

- **class LSystem_building**

  This class is the core class for generating the building according to the rules and parameters loaded from files or changed after the loading operation.

  It mainly holds two groups of data. The first group of data is used to hold the L-System parameter, resulting string generated by the iterative process of L-System. The second group of data is to hold the vertex information of different part of the building such as floor board, window, balcony etc. for rendering.

  Except for offering the member functions to create the basic L-system functions such as loading the configuration file containing the rules and parameters, iterating a certain times to generate the final string with axiom and rules, and parsing each letter the final string, it also includes member functions that help generate the building while the final string is being parsed. These member functions can be classified as several main processes such as extending, triangulating the polygon of the floor board, cutting window out of walls, and generating the mesh of floor boards, walls, window frames, and balcony.

- **class circular_list**

  This class is the auxiliary data container used in Ear-Clipping algorithm.

It is a doubly-linked and circular list which improves the efficiency during the Ear-Clipping operation. In Ear-Clipping algorithm, one of the basic operations is to remove a vertex from the polygon and connect its two neighbors up after an 'ear' is found and cut. As it is fast for an array to position rather than insert or remove data (Figure 3), we instead use the doubly-

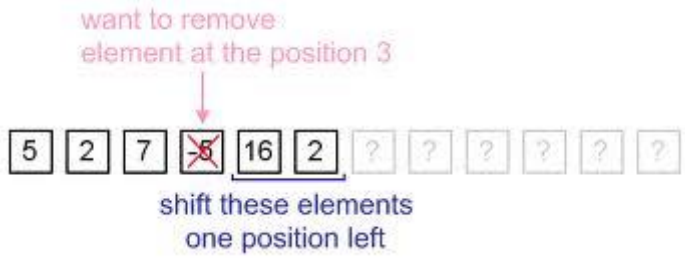


Figure3.

linked list to remove vertex and get its predecessor and successor conveniently. In addition, as the removing operation can happen on each node of the doubly-linked list, a doubly-linked and circular list is made to avoid checking if the node to be removed is the head or the tail node (Figure 4).

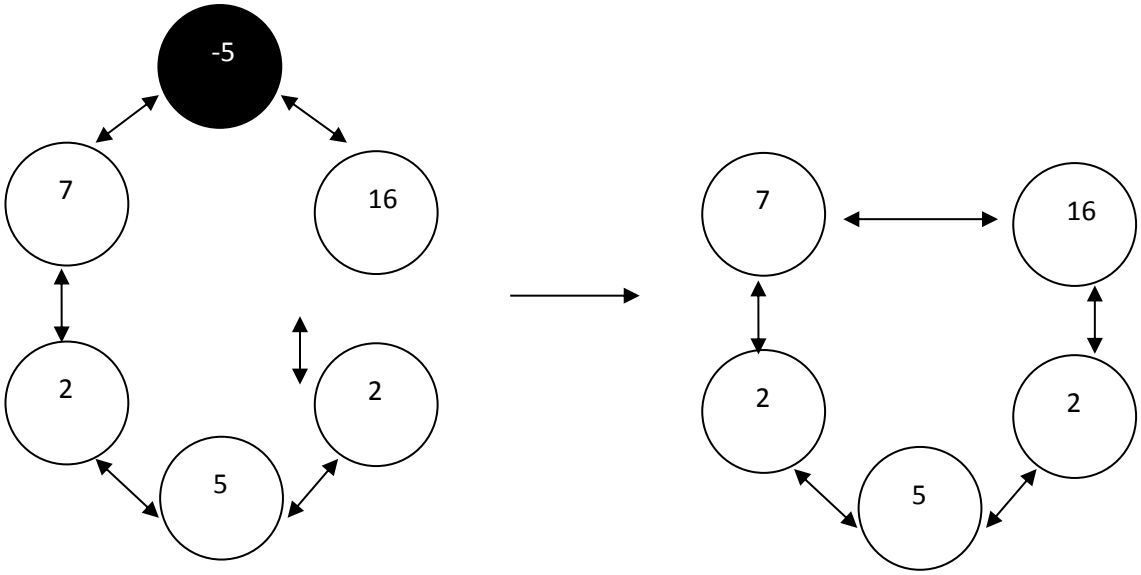


Figure4.

- **class camera_control**

  This is an auxiliary class that maintains camera_to_world matrix. It offers member functions to change the distance between the camera position and the object being viewed, and rotate the camera around the object. All the changes will be applied to the matrix straight away rather than construct the matrix each frame as the frequency of the change of camera parameters is relatively low compared to the frame rate.

- **struct state**

This is a structure that is used to be pushed in and popped out through stack in LSystem_building. It holds the current operational position, rotation matrix, wall length and the u coordinate for texture sampling which makes it possible to ensure all walls have continuous u coordinate.

- **class stochastic_rule**

  This class is mainly designed to store stochastic rules(degenerate as normal rule when there is only one rule stored) from which it picks rule randomly when it is asked to offer a rule. The probability of each rule is equal which means the probability of each rule is 1 / rules count

- **class font_helper**

  This is a singleton helper class to output text in OpenGL based project. It is mainly implemented using wglUseFontBitmaps and glCallLists. The basic features it offers are configuring the font properties such as the font type, font size and etc., and formatting drawing a string with different colors onto any position of the screen.

Implementation details

This section describes the main functions of LSystem_building and LSystem_building_app classes in details.

- **void load()**

  Load configuration file containing rules and parameters which are used to initialize the L-System instance. The table below lists parameters and their meanings respectively.

Semantic String	Meaning
angle	angle for each turn
branch length	the width of each wall
extension length	the width of the outside part of the floor
floor thickness	the floor thickness
wall height	the wall height
floor count	the floor count
branch length decrement	decrement of wall length for each ‘F’ operation
world position	the world position of the building
iteration	iteration count
axiom	initial string
letters	letters to be replaced by strings during each iteration
rules	strings to replace letters

texture	Index of the textures to be applied on to the building
balcony extension	the width of the balcony
balcony height	the height of the balcony

- **void render()**

  Combine the world, view, and projection matrix and render the generated building mesh which consists of wall mesh, floor mesh, balcony mesh, and window frame mesh. Four different textures are applied to the meshes respectively. These textures are specified by the parameter.

- **translate_building_to_origin()**

  Translate the building and attach the approximate center of the ground floor to the origin. The idea is to get the center point of the bounding rectangle.

- **void generate_output_str()**

  Generate the final string by iteratively replacing symbols with respective strings according to the current rules and letters. Letters which occur more than once corresponds to different rules, which makes them have probability to be chosen for the next substitution. This is the basic idea how the stochasticity is implemented.

- **void generate_mesh()**

  Generate wall mesh, floor mesh, balcony mesh, and window frame mesh according to the output string. This is the function to parse the final L-System string and construct a complete floor of a building. It will invoke several functions to finish the construction of each component of the building.

Letter	Meaning
+	Rotate around axis y by angle δ
-	Rotate around axis y by angle $-\delta$
K	Cut window out of a wall
(	Set the window size
F	Make a branch
B	Add a balcony to the wall
[	push current state onto the stack
]	pop out state from the stack, and make it current state

- **void extend_floor_polygon()**

  Extend the floor polygon outward by a certain distance to form the extension part of the floor coming out from the inside of the building (Figure 5).

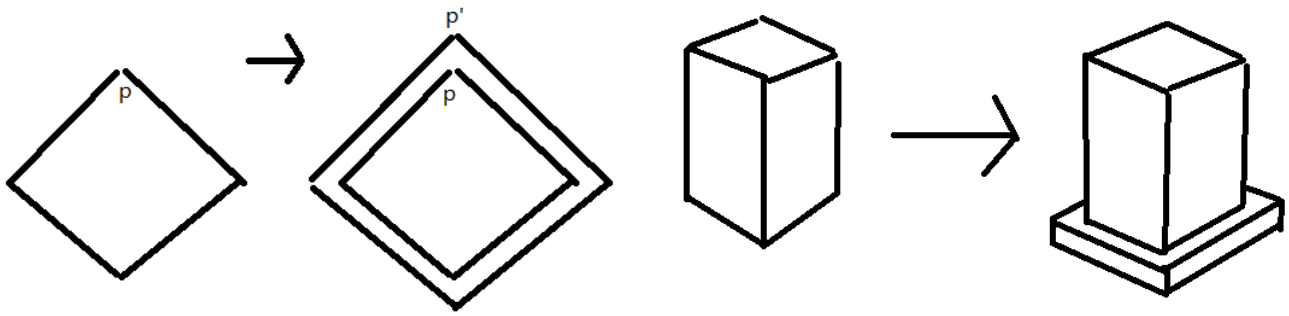


Figure5.

- **void extend()**

  Generate one wall whose width and height is obtained from the member variable. Instead of generating a line segment like how it does in using 2D L-System to draw a tree, it generate a rectangle surface(Figure 6).

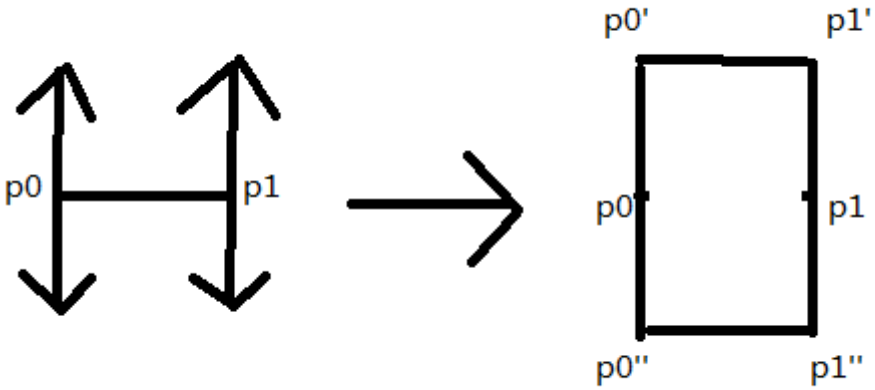


Figure6.

- **void enclose()**

  Connect the end point and the starting point and create a wall in between. The operation is executed only if the final rule does not close the wall itself. This is simply implemented by checking if the accumulated angle has exceeded 360 degrees.

- **void ear_clipping_triangulation()**

  Triangulate the floor board and roof which can be concave polygon using Ear-Clipping algorithm. One way to triangulate a simple polygon is based on the fact that any simple polygon with at least 4 vertices without holes has at least two 'ears', which are triangles with two sides being the edges of the polygon and the third one completely inside it. The algorithm then consists of finding such an ear, removing it from the polygon (which results in a new polygon that still meets the conditions) and repeating until there is only one triangle left.

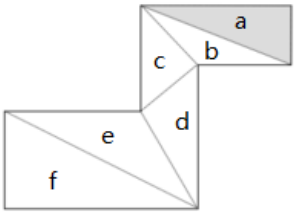


Figure7.

  The algorithm starts scanning all vertices of the polygon by sweeping a line from any side of the polygon. In Figure 3, it starts from the top side. For each vertex it meets, a triangle will be constructed from the vertex, and its two neighbor vertices on the polygon. Other vertices on the polygon will be checked to see if they are inside the triangle. If any of them is inside the triangle, then the triangle isn't the ear, the sweep line keeps going and do the same check for the next vertex it meets. If all points on the polygon except for the three making up the triangle are not inside triangle, it means the ear is found, and it will be removed from the polygon as a triangulated triangle. The corresponding point on the polygon will also be removed. The algorithm keeps repeating the operation until all points on the polygon have been removed.

  In Figure 7, First, the vertex on the top right corner will be met, the triangle which it belongs to is triangle a. Then other points on the polygon which don't belong to triangle a will be checked to see if any of them is inside triangle a. The result is that no other points are inside the triangle, so this is one of the ears which can be cut down. The same goes for triangle b, c, d, e and f. So all these triangles in Figure 7 are the final triangles after the Ear-Clipping triangulation.

Pseudo code:

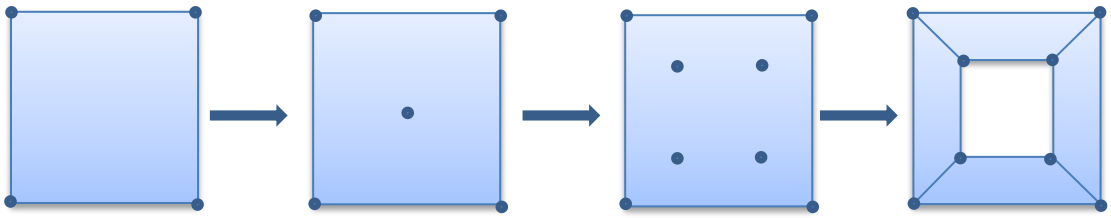
```
void ear_clipping(polygon, output_triangle_list)
{
	while(polygon.size() > 3)
	{

		for(i = 0; i < polygon.size(); i++)
		{
			triangle tri = triangle(polygon[i]->next, polygon[i], polygon[i]->prev);
			point = polygon[i]->next->next;
			for(j = 0; j < polygon.size() - 3; j++)
			{
				if(is_point_inside(point, tri))
				{
					break;
				}
				point = point->next;
			}
			if(j == polygon.size() - 3)
			{
				output_triangle_list.push_back(tri);
				polygon.remove(i);
				break;
			}
		}
	}
	output_triangle_list.push_back(triangle(polygon[0], polygon[1], polygon[2]));
}
```

- **void create_horizontal_surface_for_each_floor()**
  void create_side_surface_for_each_floor()

  Create horizontal and side surface for each floor board. The horizontal surface of the floor board will be triangulated by using Ear-Clipping algorithm, and the side surface of the floor board will be generated by creating new vertices along y coordinate of vertices on floor polygon, and combining old and new vertices. Once the first floor board is generated, all other floors will be produced by simply translating the first floor board along y coordinate, which optimizes the efficiency of the code.

- **void cutWindow()**
  This function is called when the K letter appears in the final string. It uses the four vertices that represent the wall mesh to cut the window. Firstly the middle point on the wall is found, and then 4 vertices that represent the window are calculated given the window size. Next step is to remove old wall vertices from the list and substitute them with new 16 vertices that represent new 4 meshes. The uv coordinates are also recalculated in the similar manner and stored in the list.

The diagram below demonstrates the process:



In order to make the code more optimized, this process is not calculated for each floor, but instead it is done once and then the window points for the next floors are just translated along the y-axis.

- **void makeWindowFrame(vec3 &wp0, vec3 &wp1, vec3 &wp2, vec3 &wp3)**
  This function takes in four window points and generates the meshes of the window frame. Firstly four new points around the window wp0 point (red point on Figure 8) are calculated, and then the other twelve points are found by either adding outer distance or inner distance to respective points. To add the middle frame, the middle point of the inner bottom frame

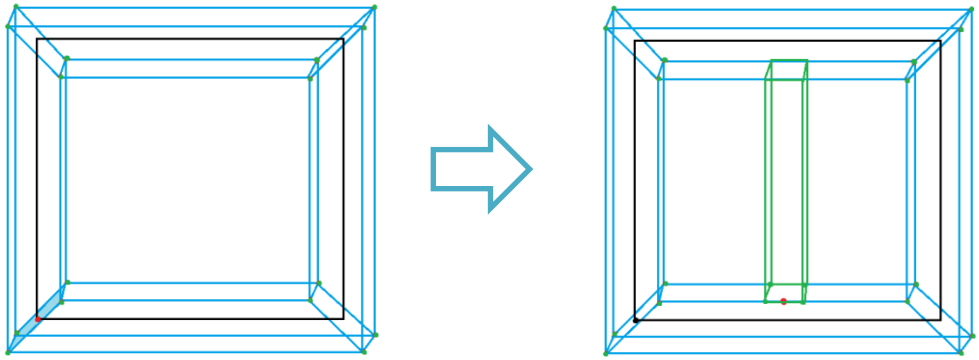
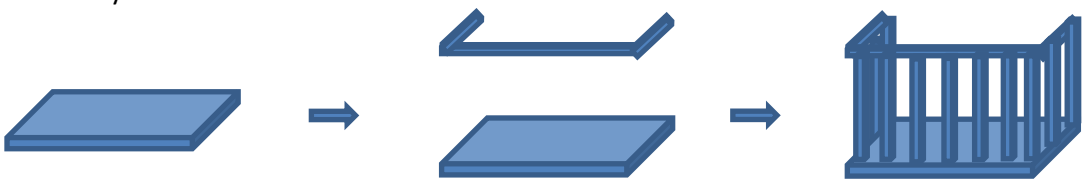


Figure 8

was found and from that point the rest of the middle frame points were found. Those points are then used to create frame meshes. Then similar to the window, frame points for the next floors are just translated along the y-axis.

- **void addBalcony(vec3 &p0, vec3 &p1, vec3 &p3, vec3 &vec)**
  void make_vertical_rack(vec3 &p_bot, vec3 &p_top, vec3 &vec)
  The balcony mesh consists of three main parts: balcony floor, balcony top racks and vertical racks. The balcony floor and top racks are calculated given the size of the wall and the balcony extension width.



Then do calculate the vertical balcony racks, we need to partition each side (left, front and right) according to the spacing between racks. Then for each partition we find the pair of points (one on the balcony floor, one on the balcony top rack) and these points are then used to calculate the meshes of individual racks in make_vertical_rack(vec3 &p_bot, vec3 &p_top, vec3 &vec) function. The following pseudo code demonstrates the partitioning process for one of the sides:

```
spacing = 0.07f;
partitions = length/spacing;
//p1, p2
for(i = 0; i < partitions; i++)
{
	t = i/partitions_v;
	bot_point = (1-t)*m_b_p1 + t*m_b_p2;
	top_point = bot_point + rack_height;
	make_vertical_rack(bot_point, top_point);
}
```

Similar to windows, this process is done only for one floor, and then all points are translated along y axis for other floors.

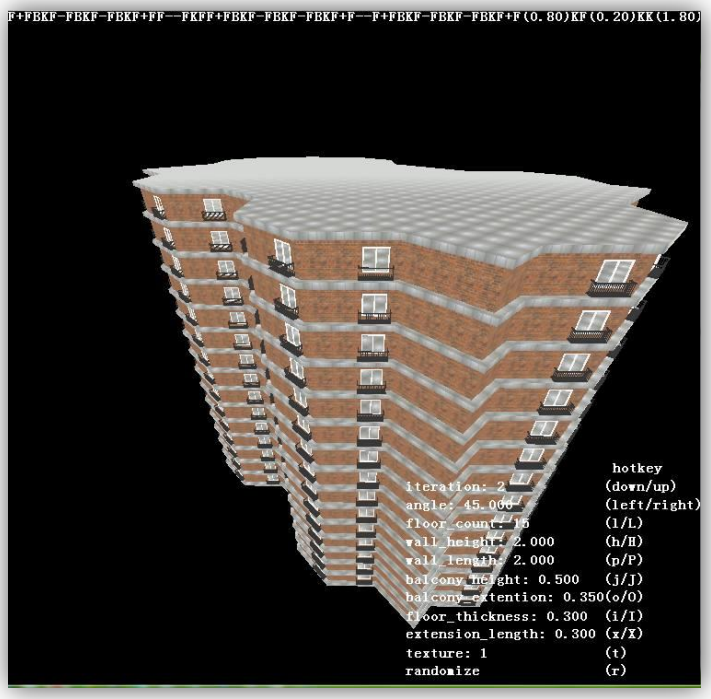
- **void hotkeys()**

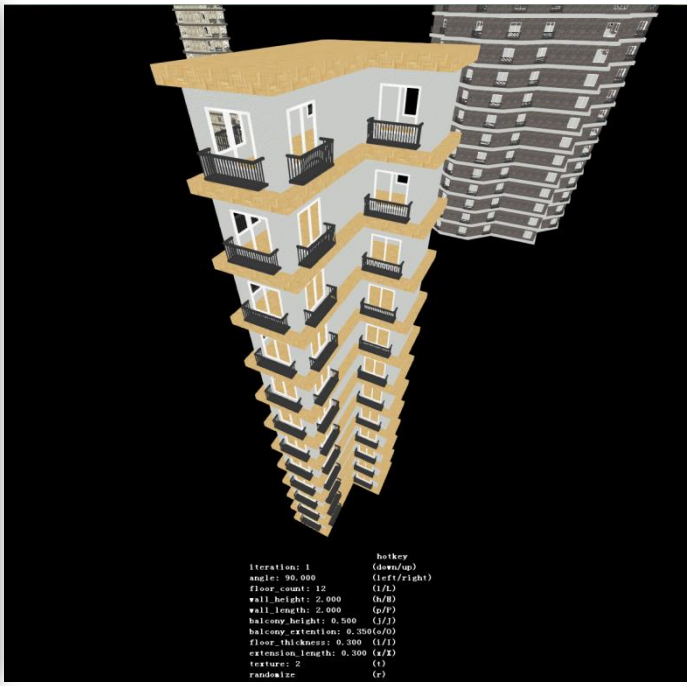
  This function is responsible for reading the input from the user and proceeds with the respective actions:

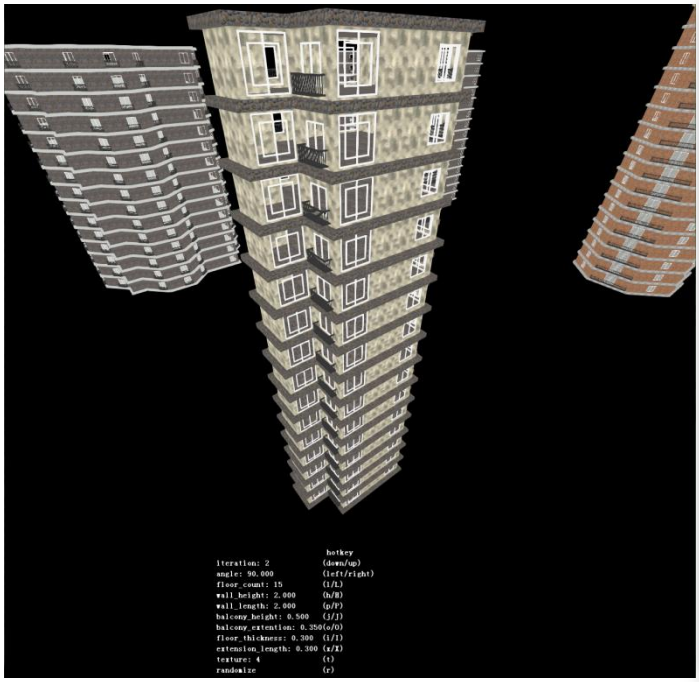
Hotkey	Action
Left mouse button	Hold and drag to rotate the camera
Mouse wheel	Scroll to zoom in and zoom out
W	Rotate camera up
S	Rotate camera down
A	Rotate camera to the right
D	Rotate camera to the left
Q	Zoom in
E	Zoom out
Key up	Increase iteration number
Key down	Decrease iteration number
Key right	Increase angle
Key left	Decrease angle
p	Increase branch length
P	Decrease branch length
h	Increase wall height
H	Decrease wall height
m	Increase window size
M	Decrease window size
l	Increase floor count
L	Decrease floor count
x	Increase floor extension length
X	Decrease floor extension length
i	Increase floor thickness
I	Decrease floor thickness
o	Increase balcony extension
O	Decrease balcony extension
j	Increase balcony height

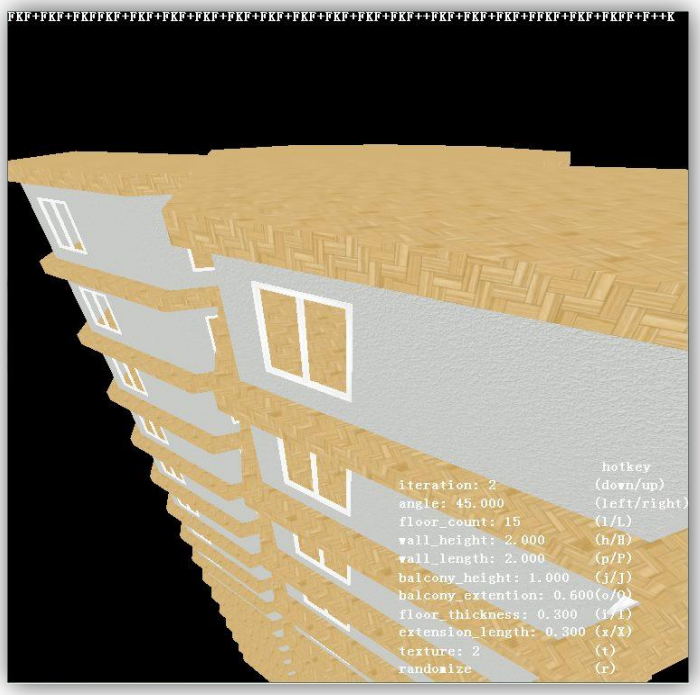
J	Decrease balcony height
T	Toggle between textures
R	Randomize building
k	Increase the view distance of the camera
K	Decrease the view distance of the camera
b	Reload configuration files

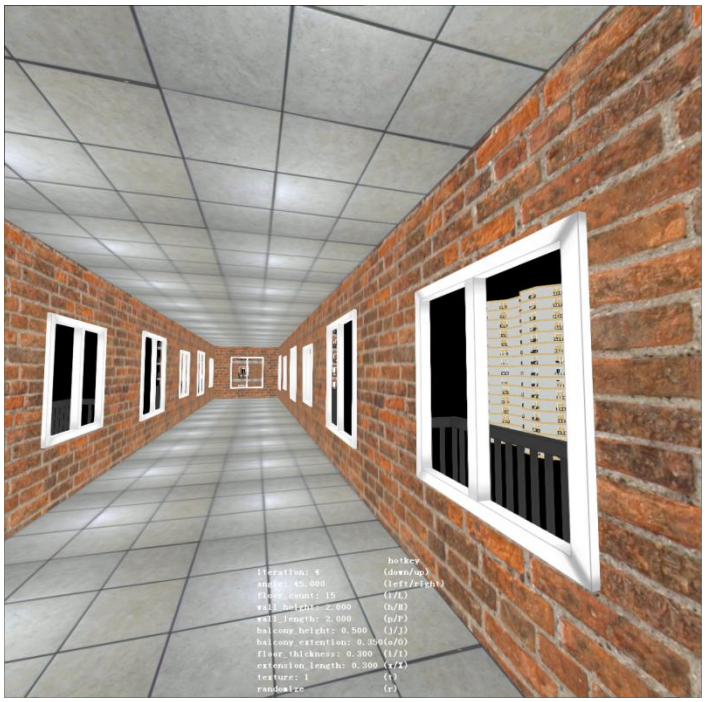
Results



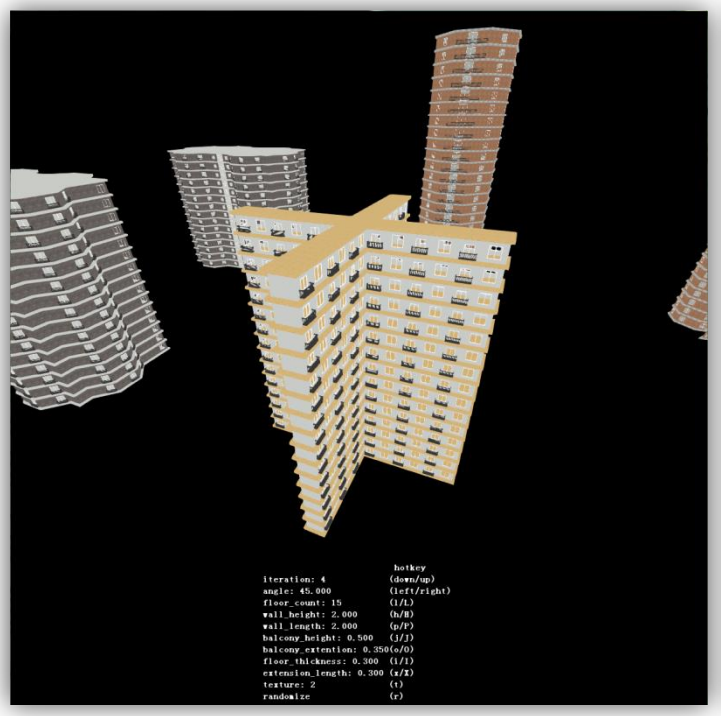


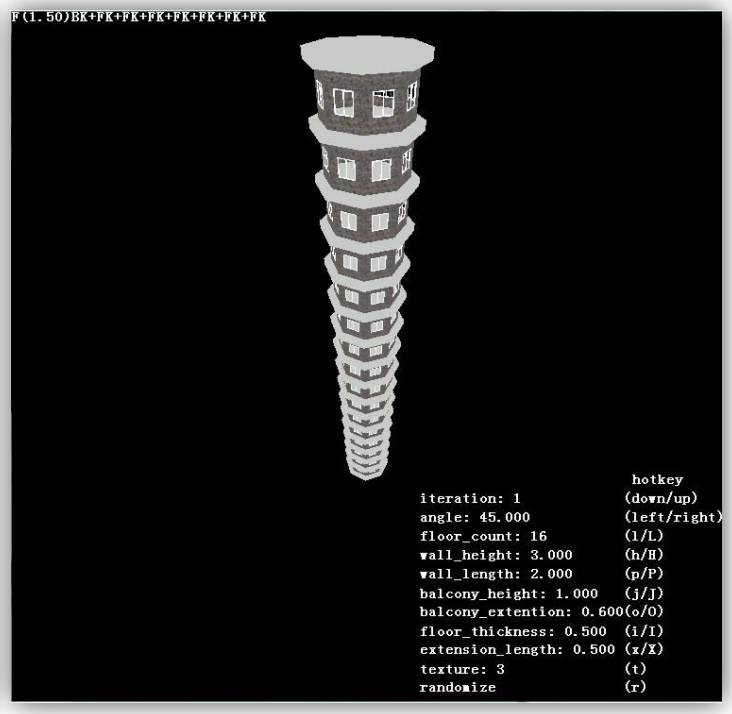





hotkey
iteration: 4 (down/up)
angle: 45.000 (left/right)
floor_count: 15 (l/L)
wall_height: 2.000 (h/H)
wall_length: 2.000 (p/P)
balcony_height: 0.500 (j/J)
balcony_extention: 0.350(o/O)
floor_thickness: 0.300 (i/I)
extension_length: 0.300 (x/X)
texture: 1 (t)
randomize (r)


hotkey
iteration: 2 (down/up)
angle: 45.000 (left/right)
floor_count: 5 (l/L)
wall_height: 5.000 (h/H)
wall_length: 3.500 (p/P)
balcony_height: 1.000 (j/J)
balcony_extention: 0.700(o/O)
floor_thickness: 0.500 (i/I)
extension_length: 0.500 (x/X)
texture: 4 (t)
randomize (r)


hotkey
iteration: 4 (down/up)
angle: 45.000 (left/right)
floor_count: 15 (l/L)
wall_height: 2.000 (h/H)
wall_length: 2.000 (p/P)
balcony_height: 0.500 (j/J)
balcony_extention: 0.350(o/O)
floor_thickness: 0.300 (i/I)
extension_length: 0.300 (x/X)
texture: 2 (t)
randomize (r)


F(1.50)BK+FK+FK+FK+FK+FK+FK+FK
hotkey
iteration: 1 (down/up)
angle: 45.000 (left/right)
floor_count: 16 (l/L)
wall_height: 3.000 (h/H)
wall_length: 2.000 (p/P)
balcony_height: 1.000 (j/J)
balcony_extention: 0.600(o/O)
floor_thickness: 0.500 (i/I)
extension_length: 0.500 (x/X)
texture: 3 (t)
randomize (r)

Bibliography

- 2014. Procedural Modelling of Buildings [ONLINE] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.112.183. [Accessed 6 January 2014];
- PARISH, Y. I. H., ΜULLER, P. 2001. Procedural modelling of cities. In Proceedings of ACM IGGRAPH 2001, ACM Press, E. Fiume, Ed.,301–308;
- 3Daet.com | L-system City . 2014. *3Daet.com | L-system City* . [ONLINE] Available at: http://www.3daet.com/pages/805/l-system_city/. [Accessed 6 January 2014];
- Polygon triangulation - Wikipedia, the free encyclopedia. 2014. *Polygon triangulation - Wikipedia, the free encyclopedia*. [ONLINE] Available at: http://en.wikipedia.org/wiki/Polygon_triangulation. [Accessed 5 January 2014];
- Polygon Triangulation. 2014. *Polygon Triangulation*. [ONLINE] Available at: http://www.cs.unc.edu/~dm/CODE/GEM/chapter.html#FIG. [Accessed 5 January 2014];
- 2014. [ONLINE] Available at: http://www.cs.ucsb.edu/~suri/cs235/Triangulation.pdf. [Accessed 29 December 2013] ;
- Polygon Partitioning. 2014. Polygon Partitioning. [ONLINE] Available at: http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/PolyPart/polyPartition.htm. [Accessed 5 January 2014].