Executive summary

Today, the development of animation and video game is getting faster and faster and the market of them is also super large. However, their development cost is huge at the same time. The development cycle could be very long, the consumption of human and hardware resources is huge and the requirement of performance is increasing rapidly.

Architecture, as an essential element of animation and video games, can be found in almost all animation and video games. One feature of the architecture is that they rarely appear alone, so a lot of resource of the entire project is occupied by them. Therefore, a fast, convenient and excellent architectural modeling method is very urgent needed. This project is undertaken for solving this problem.

In this project, a procedural modeling of architecture tool is implemented. This tool is a Maya plug-in which can quickly build up a detailed and relative real British city. The attributes of each building can be controlled by users. The components of each building are produced procedurally which include windows, roofs, doors etc. And the amount of buildings is also decided by users. Multiple different buildings can be generated procedurally and randomly. Users can even design their own city map by drawing the position of each building. All buildings will be located on this map.

I created a new algorithm for producing a whole building with bricks rather than primitives see pages 41

I created a new algorithm for adding tiles on a roof to make it looks more real, see pages 43

I combined C++ MFC with Maya plug-in together to let users draw the City Map in MFC and implement it in Maya, see pages 49

I implemented the city map by using the Voronoi Diagram to create a complex map, see pages 49

I created an algorithm to model and locate the bay windows by separating the ground floor into two parts, see pages 39.

Table of Content:

1 Introduction

1.1 Aims and Objectives

The aims of this project could be:

Develop a tool to help Video Game and Animation developers to produce their architecture faster. These architectures are built procedurally with considerable amount of details.

Dynamic effects can be applied directly to these architectural, such as the effect of explosion.

Create a city map and arrange the buildings on it

To achieve these aims, there are several technical tasks should be finished:

- Individual building controlled

As all the buildings are created with random attributes, some characters of some buildings may not satisfactory, so the attributes should be able to be modified by users.

- Multiple building creations

All buildings should be generated procedurally and randomly to keep them all differently.

- Build architecture by bricks

If the buildings are made by bricks and tails rather than by primitives only, many dynamic effects can be involved directly.

- Assign appropriate texture to architectures

Textures can make buildings more realistic. In addition, texture can replace some details on buildings sometimes. The modeling difficulty can be reduced in this way.

- A city map production

This city map should be used to locate the buildings. We should create a reasonable and real city map.

1.2 Background and Context

Commercial Viability

From the report of ESA(entertainment software association), in 2010, consumer spent

15.90 billion dollars on Video Games content totally. And the 1.4 billion box office of Hollywood animation movie in 2010 was double as in 2009. This amazing development of video games and animation industry is obvious. However, the difficulty and risk of development is increasing, not only because of the requirement of high quality, but also because more and more grand scenes are wanted. The effort paid by a typical Game/Animation developer is in the Table1

	Content Creation	*Admin*	*Music*	*Programming*
Effort(Percentage)	*60%*	*10%*	*10%*	*20%*

Table 1

It is obvious that the majority effort of a game development is content creation which consumes half of the whole effort. Correspondingly, this part is the hardest one in the development process.

Architecture, as an essential element of almost all animation and video games, appears in different scenes in different styles and scales. However, due to the diversity and complexity of the architecture, buildings are always totally different from each other. That means developers have to model them manually with dozens hours to achieve this flexibility. This project provides a method to solve this problem by modeling all the architecture procedurally and randomly.

The project can generate a number of buildings procedurally in few seconds and can save developers` days of working time. Due to the attributes of the elements of each building are assigned randomly, no building is exactly same as others which is just like the real world.

In addition, the buildings in this project are suitable for some particular requirements as the almost all components (e.g. bricks, doors, windows, roofs, tiles and etc.) of one building are independent, they are named systematically and their degree of coupling is low. So if developers want to implement some actions to particular parts of the buildings, these parts are easily to be selected and modified.

The interaction between developers and project is good. The attributes of components are modified with the inputs from user interface, as well as the attributes of city map. These UIs also limit the actions of developers to make the input meaningful.

All the research and development are implemented as a Maya plug-in. Maya plug-in is “A plug-in which enables Maya developers to hide their code and protect their intellectual property.” That means this project has its own clear commercial value besides the academic meaning.

1.3 Language and Environment

This project uses MEL and C++ two programming languages. The part of building modeling uses MEL which is short for Maya Embedded Language. The MEL is a script language and it can execute in Maya directly with a relevant slow running speed. C++ is used in the city map design part. Maya C + + API is a Maya advanced programming method. Compared with the MEL script, the Maya plug-in developed by C++ API has advantages of speedy code execution and high security, and can also use a pointer for memory access and operation conveniently. However, the programming interface of Maya C++ has platform correlation, so the performance of the cross-platform is poor.

2 Framework and Work Flow

Framework

The first two rectangles stand for two Maya C++ API project.
Voronoi City Map Creation can generate the data of Voronoi map.
Maya plug-in Creation translates these data into Maya plug-in.
.mll file will be created then imported in Maya.

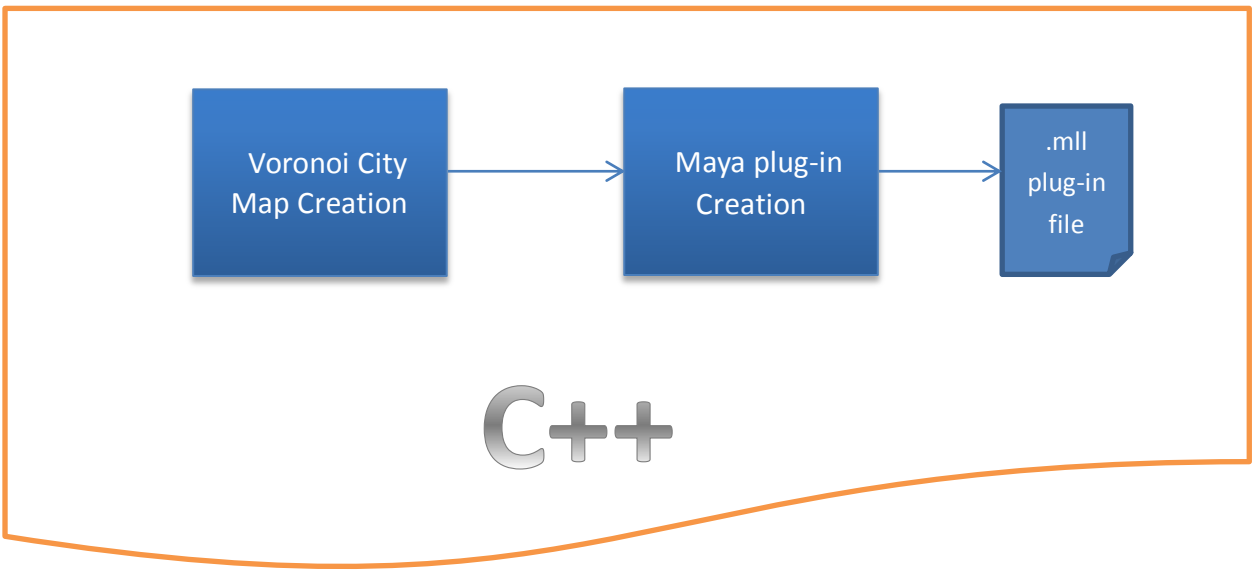


Figure 1 FrameWork C++

Each rectangle stands for a MEL script file. All other scripts are based on the Single Building Modeling:

- Single Building Attribute Control Window calls Single Building Modeling when attributes are modified.
- Multiple Building Numbers Control Window uses a loop to call Single Building Modeling with randomly attributes
- Model Buildings by Bricks replaces primitives of a building basis with bricks.
- Model Roof by Tiles replaces primitives of a building roof with tiles.

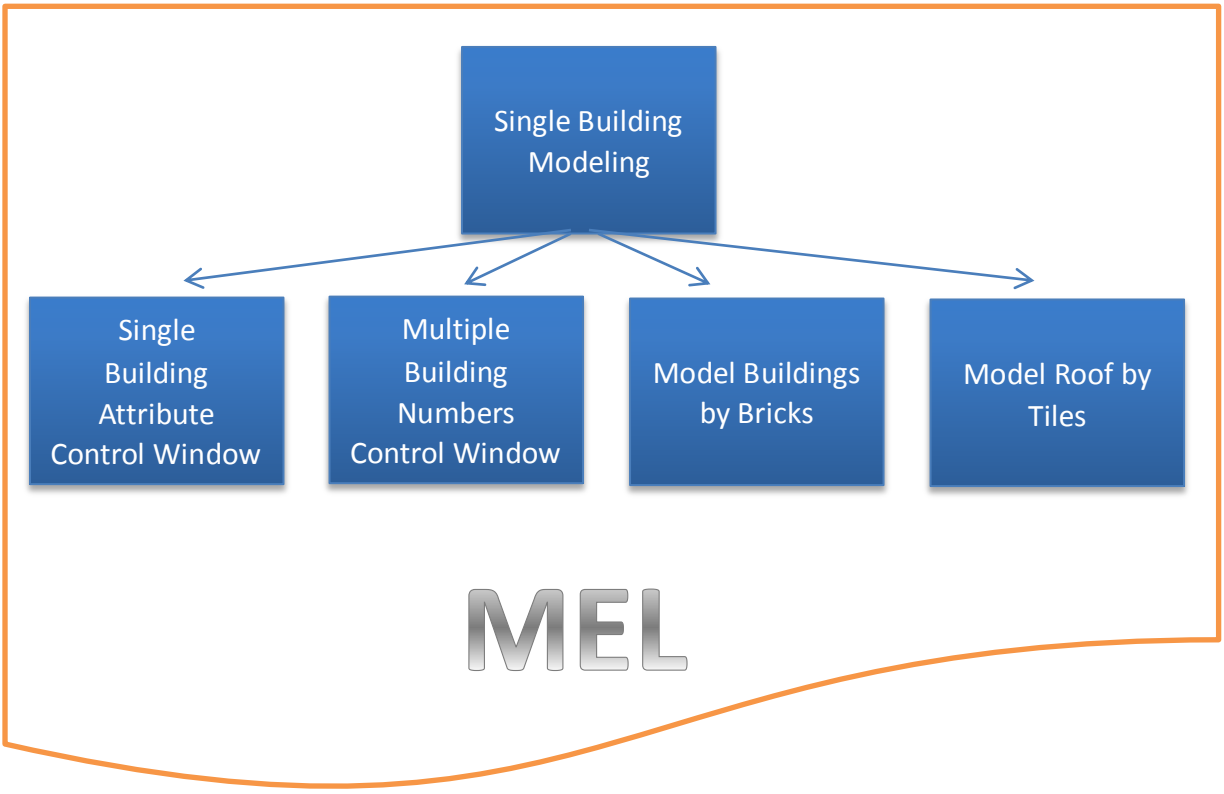


Figure 2 Framework MEL

Work Flow

Figure 3 shows the Flow Chart of this project. Firstly, users input the number of buildings they want to create. Then there is a loop to create single building several times. The attributes of each building can be modified by the input from users on the Single Building Attribute Window. On the other side, in a MFC application, users can draw some points which stand for the positions of buildings. These points then are used to create Voronoi Diagram which is treated as a map. The data of this map will

be recorded and translated to a .mll plug-in file. Finally, this plug-in will be imported and buildings will be located according to the map.

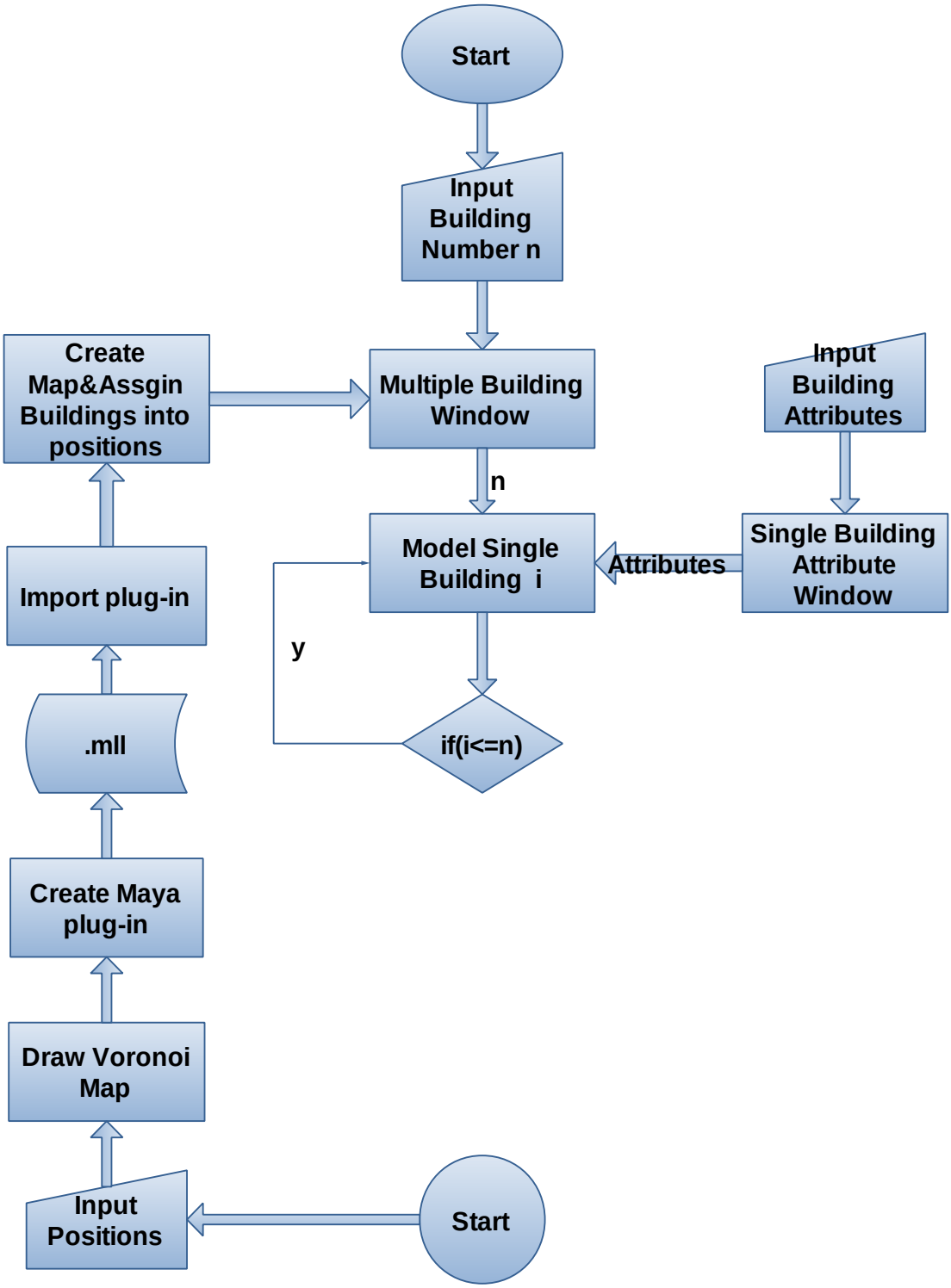


Figure 3 Work Flow(Flow Chart)

3 Architecture Research

“Roofs and walls, doors and windows are the essential features of buildings.” (D.K. Ching, 1995). Peter Wonka (2006) also states that whenever, whatever and wherever the architectures are, they always constituted by similar elements. These elements can be: windows, doors, ledges, quoins, window gratings, awnings, window ornaments, pilasters, and stairs.

Basis: support the full load of the building, and transmit these loads to the foundation.

Wall: as load-bearing elements, bear the load of the building which came from the roof or floor layers, and then pass these loads to the basis. As building envelope components, exterior wall plays to resist invasion by the nature of various factors on the indoor; interior wall plays a role of separating room, creating the indoor comfort.

Floor layer: under furniture, equipment and human load, as well as its own dead weight, and then pass the load to the wall. Besides that, it also gives horizontal support to the walls.

Floor: bear the load in the ground floor room.

Stairs: vertical transport facilities of buildings, used for people up and down the floor and the emergency evacuation.

Door: entrances and exits of the building.

Roof: to resist the influence of nature rain, snow and sun heat radiation on the top floor rooms; bear loads of top of the building and pass loads to the vertical load-bearing elements.

Quoins: On the brick or walls, quoins are the stones of the corners. Quoins can be used as structural or decorative. The feeling of strength will be felt by quoins on buildings.

Pilasters: are built-in wall or attached to the wall surface. They are slightly prominent pillars. Normally, the pilaster is in a flat or rectangular shape, but sometimes also in the semicircular columnar shape or other arbitrary shape of the true column, including a helical column.

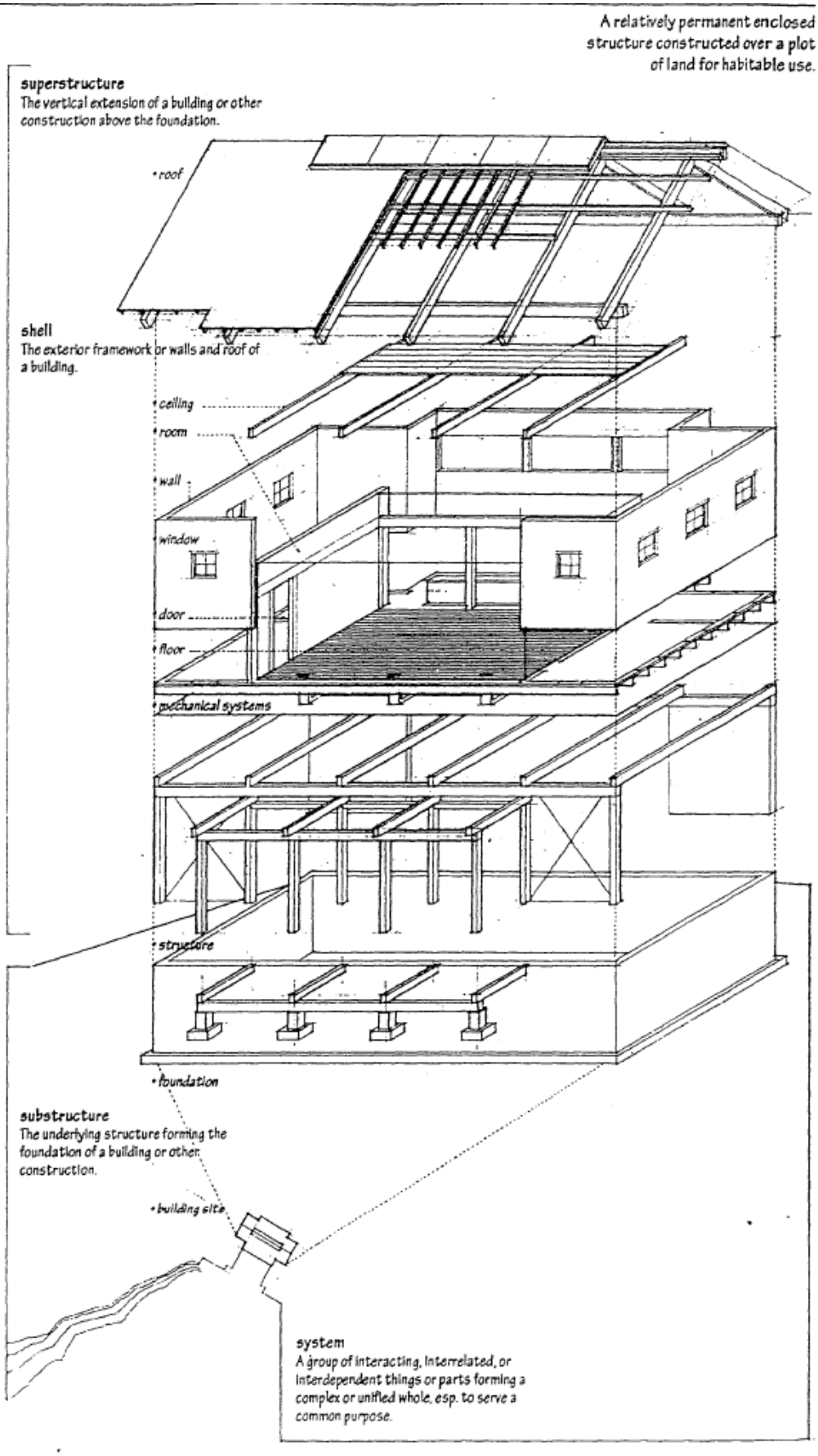


Figure 4 Architecture Structure

D.K. Ching states that a roof can be flat, sloping or curved. Lean-to is a kind of roof which only has one slope. A hip can be defined as roofs intersect in an inclined line when their two walls make a projecting angle. A valley is the inclined line of intersection if the walls meet in a reentering angle. Domical or conical roofs should be carried by circular walls.

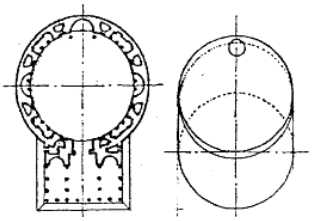
When there is more than one floor, the flat roof of the lower floor becomes the floor of the floor above. The eaves are the projecting parts when roof extends beyond the wall that supports it. If the wall also projects to support the extension of the roof, the projection is called a Cornice.

In generally, walls are made wider at the bottom and that part is called the Base. A similar projection at the top is called a Cap. A wall lower it is called a Parapet. A post is a short thick piece of wall and if it supports something, it is a Pedestal. The Die is the part between Pedestal`s cap and base. A tall post is called a Pier when it is square or a Column when it is round. Capitals are the caps of piers and columns and the shaft are the part between the cap and the base. The flat upper member of a capital is called the Abacus.

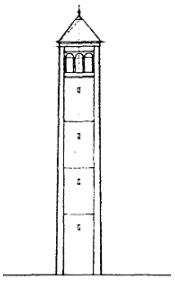
Instead of a floor or roof, a sort of continuous arch, which is called a Vault, covers the space between two parallel walls sometimes.

D.K. Ching also provides some definitions of familiar architecture structure beyond the basic elements:

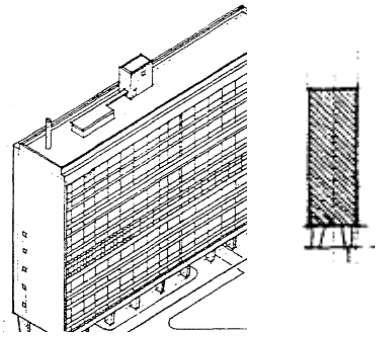
Rotunda: A round, domed building, or a large and high circular space in such a building



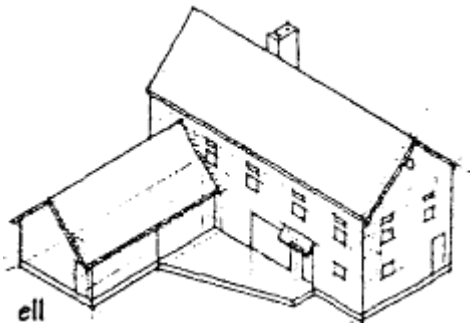
Tower: A building or structure high in proportion to its lateral dimensions, either standing alone or forming part of a larger building.



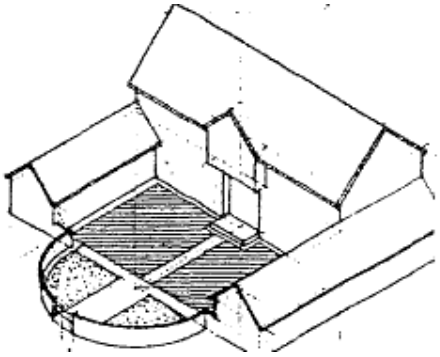
Piloti: Any of a series of columns supporting a building above an open ground level



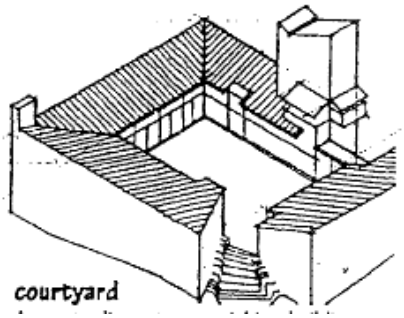
Ell: A wing at right angles to the length of a building.
Wing: A part of a building projecting from and subordinate to a central or main part



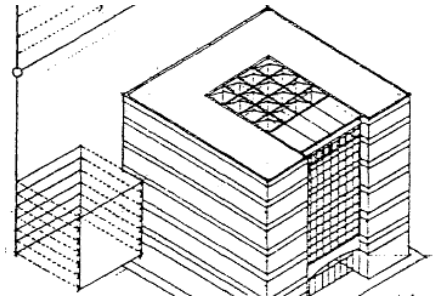
Court: An area opens to the sky and mostly or entirely surrounded by walls or buildings



Courtyard: A court adjacent to or within a building



Atrium: a sky it, central court in a building.



In the project, the basic elements should be included in every building model, for example, windows, doors, roofs, bay windows, chimneys. All attributes of these elements can be controlled by users.

4 Basic theoretical research of procedural modeling of architecture

4.1 Segment Definition

Segment Linking:
A number of segments are linked to each other through a series of nodes at their ends to achieve the desired range of shapes. At each end of each segment, there are only two nodes attached together, but there can be any number segments attached to each node.

When deleting a node, we should make sure there are no cascade effects that cause the removal of additional segments.

Quad Splines:
Users need only specify at least a quarter and the other three quarters will be mirrored when a quarter is created or modified.

The Quad Splines can be used as a constraint in the single building modeling part of the project. It will cut design time for the creation of symmetrical objects.

4.2 Vertex Definition

A list of vertices is held by each spline. A curve is defined by four vertices, one point is origin and one another is the destination. The other two are used to define the shape of the curve.

To calculate the x and y coordinates of a point t which is between 0 and 1, we use two equations:

$$x(t) = axt3 + bxt2 + cxt + x0$$
$$x1 = x0 + cx / 3$$
$$x2 = x1 + (cx + bx) / 3$$
$$x3 = x0 + cx + bx + ax$$

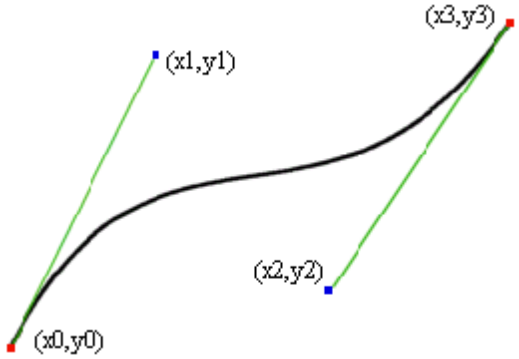


Figure 5 Example Bezier Curve

$$y(t) = ayt3 + byt2 + cyt + y0$$
$$y1 = y0 + cy / 3$$
$$y2 = y1 + (cy + by) / 3$$
$$y3 = y0 + cy + by + ay$$

4.3 Grammars

H.Focillon and W. Kandinsky give us the reason why using grammars:
“Style consists of vocabulary and syntax“ (H. Focillon)
“It‘s the content that creates the form“ (W. Kandinsky)
Peter Wonka (2006) explains that vocabulary is the formal elements which make up the vocabulary repertory. Syntax is a system of relationships and iteratively generates a design by creating more and more details.

If we want to build models with grammars, two aspect problems should be recognized. Firstly, encoding architectural design knowledge needs a powerful framework. Secondly, the architectural design knowledge should be presented appropriately by

the rules.

- Framework: e.g. syntax and semantic of a grammar
  - rule ::= id : predecessor : cond → successor : probability
  - cond ::= …
- Actual design knowledge: e.g. rules

He introduces C++ is an extremely powerful framework. In C++ classes, we could encode all architectural concepts somehow.

Maya C++ API is the first choice for the city map development because the algorithm can be executed much faster than in Maya.

Shape:

Muller(2006) describes that the shape grammar is used to configure shapes." A shape consists of a symbol (string), geometry (geometric attributes) and numeric attributes. "(Muller, 2006)Shapes are identified by their symbols which is either a terminal symbol $\in$ S, or a non-terminal symbol $\in$ V. The corresponding shapes are called terminal shapes and non-terminal shapes. The most important geometric attributes are the position P, three orthogonal vectors X, Y, and Z, describing a coordinate system, and a size vector S. These attributes define an oriented bounding box in space called scope.

Peter Wonka(2006) states that shape is one of grammar elements. One essential idea of the grammar is using a set of other shapes to replace one shape iteratively. Therefore a shape tree of the derivation of the shape grammar will be built. The tree doesn`t need to be stored explicitly and working on a set of shapes would be another implementation.

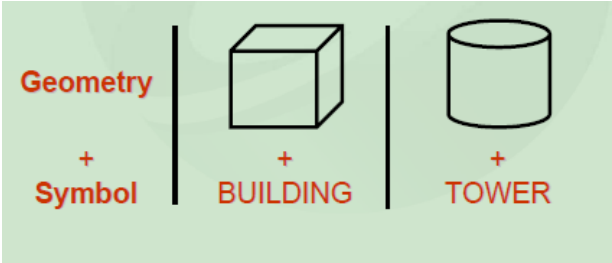


Figure 6. Shape grammars

In this part, the shape grammars are appropriate to most cases that I have met in the project, however, some polygonal buildings are hard to build only in this way. The later part like generate mesh modeling must be combined flexibility.

Production process:

Muller(2006) describes the production process as the configuration of a finite set of basic shapes. The production process can start with an arbitrary configuration of shapes axiom (A) with follow steps:

(1) Select an active shape with symbol B in the set
(2) Choose a production rule with B on the left hand side, in order to compute a successor for B, a new set of shapes BNEW
(3) Mark the shape B as inactive and add the shapes BNEW to the configuration and continue with step (1).

Depending on the selection algorithm in step one, the derivation tree (Sipser 1996) can be explored either depth-first or breadth-first. However, both of these concepts do not allow enough control over the derivation. Therefore, we assign a priority to all rules according to the detail represented by the shape to obtain a (modified) breadth-first derivation: we simply select the shape with the rule of highest priority in step one. This strategy guarantees that the derivation proceeds from low detail to high detail in a controlled manner. We do not delete shapes, but rather mark them as inactive, after they have been replaced. This enables us to query the shape hierarchy, instead of only the active configuration

Notation:
Peter Wonka (2006) gives the format of the rule as well:
“

id : pred : cond -> successor

Example:

1: fac(h) : h > 9 -> floor(h/3) floor(h/3) floor(h/3)

id – an integer identifying the rule
pred – text string = symbol of the shape to be replaced
cond: condition on the parameters of the shape
successor: shapes to replace the predecessor
“

This grammar is very suitable for producing the floors and windows in the project. Actually, a considerable part of procedural modeling work is implemented under this grammar.

Basic Split rules:

Muller (2006) provides an explanation of the basic split rule. The basic split rule splits the current scope along one axis.

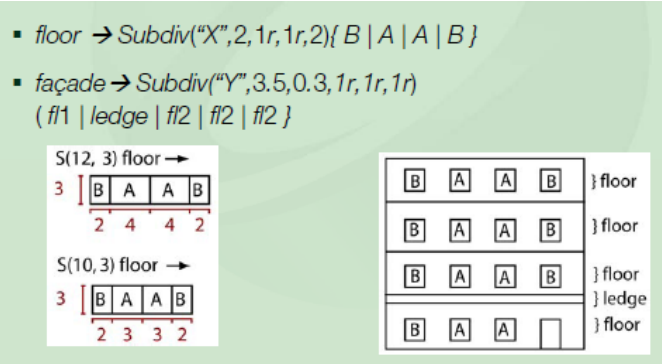


Figure 7. Scaling Split Rules

In the Subdiv, the first parameter is used to split axis x, y or z. Split sizes are described by the remaining parameters. In the braces, a list of components is given. The modifier "r" is taken into the rule to scale the split rule. "r" means relative while "a" or an absolute length for absolute. After subtract all absolute length from the splitting axis size, the remaining length is split according proportional to the r values.

Scope Rules:

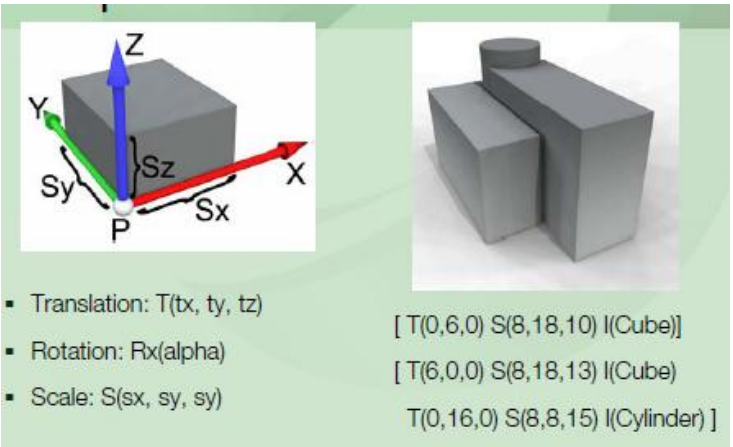


Figure 8. Scope Rules

T(tx,ty,tz) would be added to the scope position P as a translation vector. Rx, Ry and Rz rotate the axis. S(sx, sy, sz) sets the size. In a stack, the current state of the scope is pushed and pop by the modules [ ]. I(Objld) adds a geometric primitive "Objld" at point P to create geometry. According to the current state of the scope, the primitive is scaled and rotated. Besides typical objects like a cube, a quad, a cylinder, any three-dimensional model can be used.

Before use this scope grammar in the project, how to generate a couple of buildings should be considered, because the translation is the absolute position of coordination but the distances between each other are relative to their scales.

Repeat:
To allow for larger scale changes in the split rules, we often want to tile a specified element. For example:

1: floor;Repeat("X",2){ B }

The floor will be tiled into as many elements of type B along the x-axis of the scope as there is space. The number of repetitions is computed as repetitions = ⌈Scope.sx/2⌉ and we adjust the actual size of the element accordingly. Component split: Up until this point all shapes (scopes) have been three-dimensional. The following command allows it to be split into shapes of lesser dimensions:

1: a;Comp(type, param){ A | B | ... | Z }

Where type identifies the type of the component split with associated parameters param (if any). For example we write Comp(" faces"){A} to create a shape with symbol A for each face of the original three-dimensional shape. Similarly we use Comp("edges"){B} and Comp("vertices"){C} to split into edges and vertices respectively. We use commands such as Comp("edge",3){A} to create a shape A aligned with the third edge of the model or Comp("side faces"){B} to access the side faces of a cube or polygonal cylinder. To encode shapes of lesser dimension, we use scopes where one or multiple axis is zero. To go back to higher dimensions we can simply use the size command S with a non-zero value in the corresponding dimension (e.g. to extrude a face shape along its normal and therefore transforming it into a volumetric shape).

To separate the wall from the doors or windows, we also use a splitting grammar.

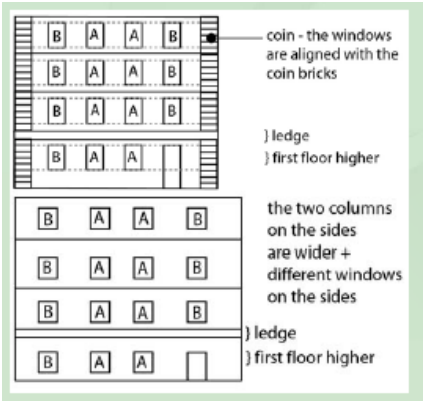


Figure 9. Split walls from windows and doors

To summary the flow about how the grammars work, Peter Wonka (2006) gives a chart

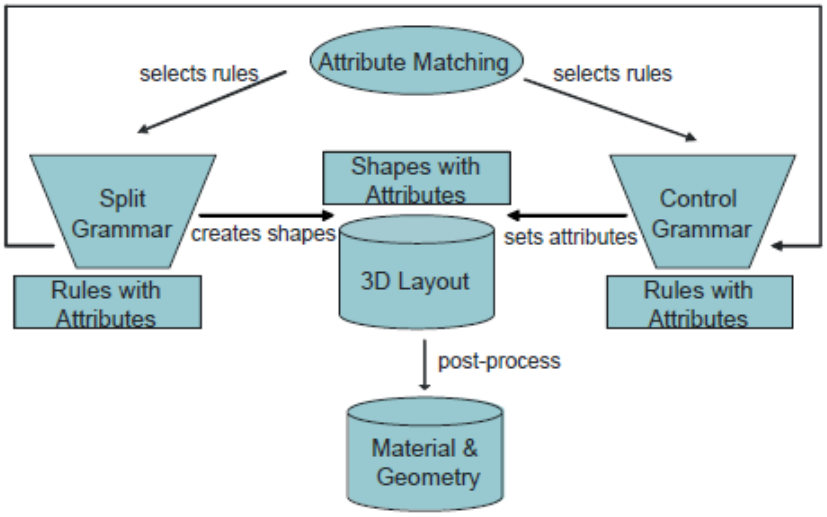


Figure 10. Grammar Work Flow

In this project, the chart could be a very important reference to build basic meshes. How to make appropriate rules and constraints is the most difficult part of it.

Limitations: The split grammars are hard to work on complex geometry.

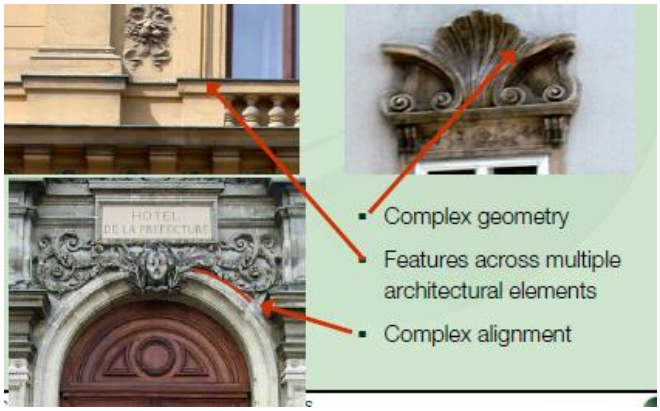


Figure 11. Limitations of Split Grammars

4.4 Generative Mesh Modeling:

Generative Modeling Language:

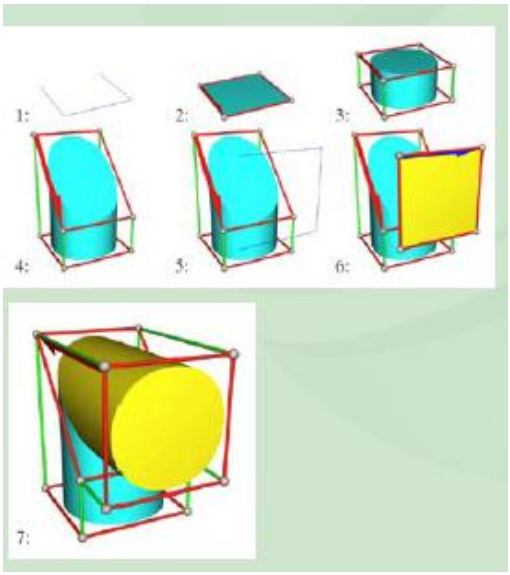


Figure 12. Generative Modeling Language

1: Parameters of the quad operator are the midpoint (0,0,-2) and extension (1,1,0) of the quadrangle and 2 as a mode flag. The quadrangle is put on the stack as an array of four points.
2: The polygon is converted to a mesh face, pushing a halfedge.
3: The extrude operator expects a halfedge and extension vector, and pushes the halfedge of the resulting face.
4: The face is moved by projecting it in z-direction (0,0,1) onto a plane.
5,6: A second quad face is created.
7: The quad faces are bridged with smooth edges.

Generate Details:

To create details on basic meshes, Peter Wonka (2006) introduces the Generative Mesh Modeling which can generate meshes by a grammar. Much additional detail can be provided by Generative Mesh Modeling.

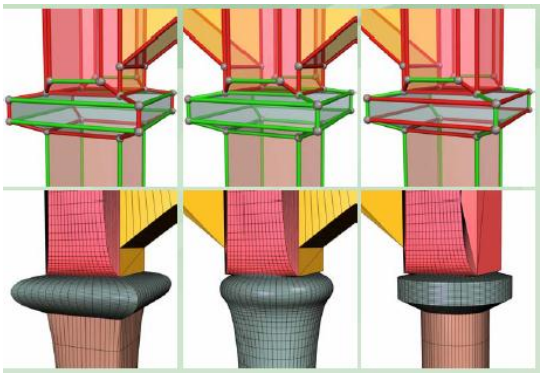


Figure 13. Different Sharpness Flags

In the Figure 11, the only differences of all columns are the sharpness of the edges and each of the columns have an identical control mesh. By combining smooth transitions with sharp creases, different shapes can be created in various ways.

Euler operator is a suitable way to combine the smooth transitions. A mesh can be treated as a graph composed by vertices, edges and faces. The Euler operators are used to manipulate meshes. They can create a new vertex, connect vertices, split a face by inserting a diagonal and subdivide an edge by inserting a vertex. Euler operator is topological operator which only the incidence relationship will be modified by them. The geometric properties will not affect it, for example, the lengths of edges or the positions of vertexes are just the attributes of geometric.

Peter Wonka (2006) refers the Euler operator which proposed by Sven Havemann (2005).

Havemann (2005) gives some examples to explain how the operators work:

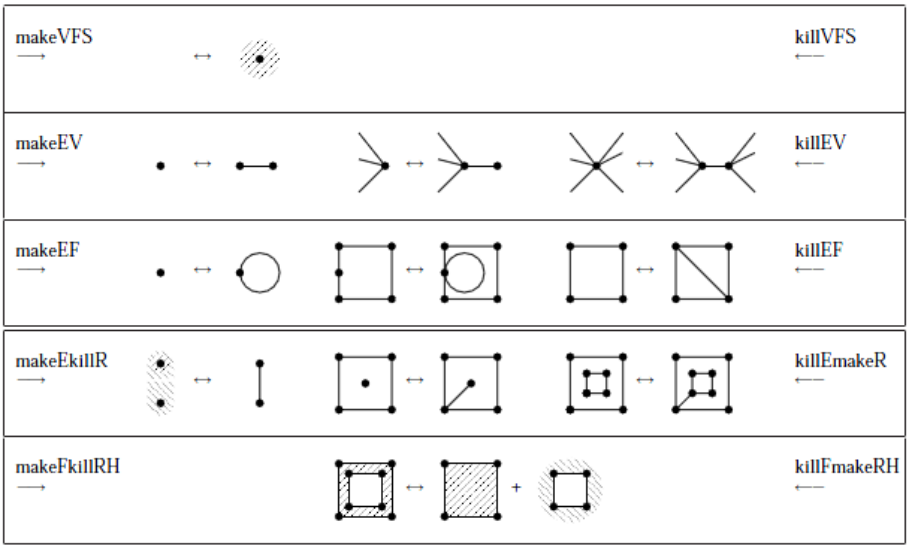


Figure 14. Planar models of Euler operators

makeVFS: Make Vertex, Face, Shell creates the minimal 2-complex, the pointed sphere. It provides one connected component (a shell) that consists of just a face and a vertex attached to it, to initialize the modeling process. The planar model actually consists of the vertex alone, the face is the (unbounded) plane around it.

makeEV: Make Edge & Vertex is the vertex split operator. It allows to split any vertex into a pair of separate vertices connected by a new edge. Thus it introduces one edge and one vertex, which explains the name.

makeEF: Make Edge & Face is the face split operator. It is dual to makeEV as it

splits a given face in two, introducing a new face and a new edge.

makeEkillR: Make Edge, Kill Ring connects a ring with the outer face boundary. This creates just a single edge to decrease the number of face boundaries by 1.

makeFkillRH: Make Face, Kill Ring & Hole turns a ring into a face of its own. This is maybe the simplest, but also the most abstract Euler operator.

Havemann (2005) explains Each row shows different situations for applying the operator. The ↔ arrows can be either read from left to right (make. . .), or from right to left (kill. . .). With the first three operators, any genus 0 shape can be built. The last two operators are concerned with rings and building higher-genus objects. The different versions of the same operator are denoted makeEV (a), (b), (c), etc.

Totally, there are five Euler operators and all of them are invertible. Each of the operators has its own condition:

makeVFS killVFS	none shell has no edges and single vertex
makeEV killEV	none edge has different vertices on both ends
makeEF killEF	vertices belong to the same face boundary of the same face edge has different faces on both sides
makeEkillR killEmakeR	vertices belong to different face boundaries of the same face edge has the same face on both sides, edge is not a loop
makeFkillRH killFmakeRH	can only be applied to a ring a face may not become ring of itself

Table 2 Conditions for legal application of Euler Operators.

In the project, the generation of details is very important as well. As I mentioned before, some complex buildings cannot be built only along with shape grammar. While, to use Eular operators, many math knowledge should be involved.

5 Modeling Algorithm Analysis and Implementation

5.1 Basis

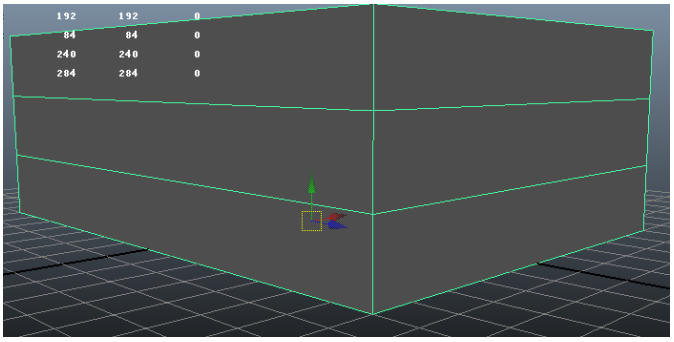


Figure 15 Basis

Some elements can be produced from simple primitives. P.J.Birch, S.P.Browne, V.J. Jennings (2002) indicate that all the major elements can be produced by cubes as the shells. These cubs are used to connect the individual rooms that make up floor by floor. They point out that the plane level is the lowest level of geometry operation permitted on the shell. For these cubes, they can be moved along the axis of their normal. This principle leads these limited actions:

Move any single face
Expand or contract opposing faces
Move the entire shell
Scale the entire shell
Rotate the entire shell

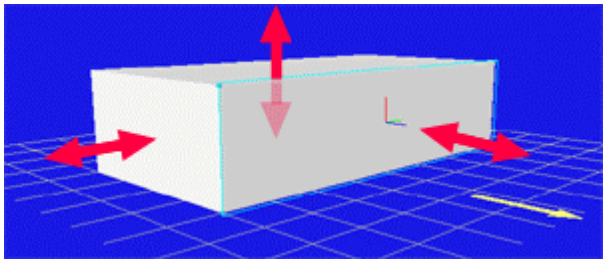


Figure 16 Simple Shell with directional constrains

In this project, cubes are mainly used in generating floors. The main step is:

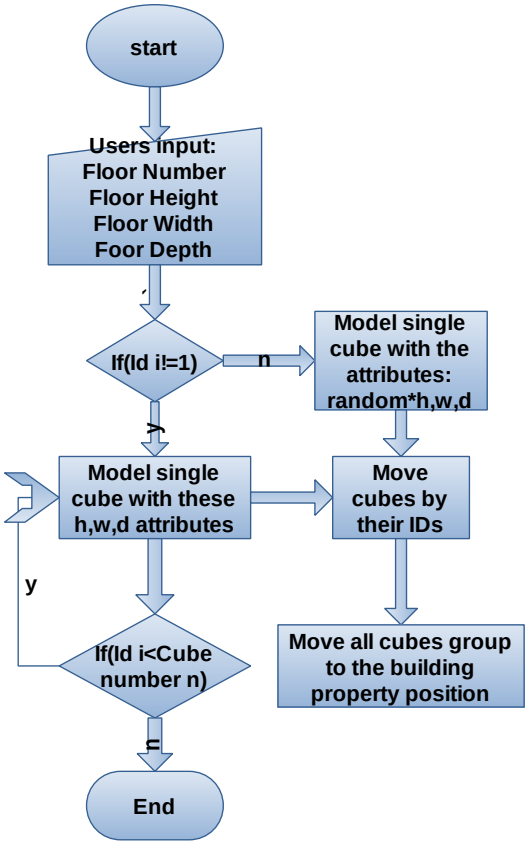


Figure 17 Flow Chart of Basis

As showed in the Figure 19 users input how many floors they want to create and give attributes to these floors, including Floor Height, Floor Width, Floor Depth. If the floor is not ground floor, this floor (cube) will be created with these three attributes. While if the floor is ground floor, then a limited random number will multiply to the Floor Height. Because normally, the ground floor is higher than other floors, the random number is greater than 1 and lower than 1.4. Then these floors will be moved to the right position along the axis Y with this formula:

Ground Floor: $GroundFloorH = FloorHeight/2$

Other Floors: $(FloorID-2)*FloorHeight+ GroundFloorH+0.5*FloorHeight$

In this formula, as the position of a cube is defined by the center of it, one cube should be moved up half of its height, so the Ground Floor is just moved up *FloorHeight/2*. The other floors should be moved up a Ground Floor height and an half of itself height and (ID-2)*FloorHeight (cause the ID begins with 1 instead of 0).

After that, all cubes are grouped and move horizontally to a appropriate position.

In the multiple building modeling, if we use this formula directly for each building, one problem will occur, the random number can make their height of Ground Floors look very different which is unreality. In this case, the height of Ground Floor of first building will be assigned to the rest of buildings as well after another limited random

number multiplied to it.

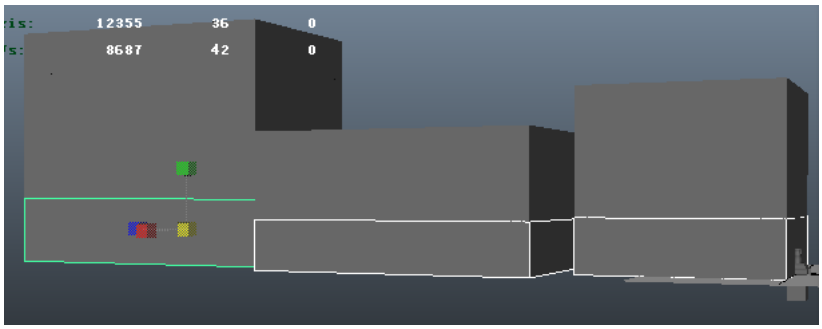


Figure 18 Unify height of Ground Floor

Simple primitive method is one of the most important methods in this project. There are several advantages to use it: Firstly, it is a really simple but efficient way to produce models rapidly, because there are existing primitives in Maya which can be modeled easily and there is Cubes API in C++ as well. Secondly, it easier to unify the modeling method, otherwise, it will be very complex to satisfy most modeling requirement with any other single method. Thirdly, as the shape is limited and modeled by existing lib, the robustness of it is good. What is more, based on these cubes, users can add more details easily on it. Finally, it can satisfy most buildings because the most building bases are cubes.

Obviously, there are still some problems may not that ideal. First of all, some special buildings are based on other basic shapes, for example Islam style is based on rotundity. Another problem is it is hard to reach the interaction requirement, because as a cube, people only can control its width, height, depth, position and rotation. That means, as mentioned before, only plane level will be controlled. In this case, users cannot model “anything” they want.

Constraints:
From opition of P.J.Birch, S.P.Browne, V.J. Jennings (2002), the position of all shells are maintained in relation to each other when one is moved. In general, the constraints of shell are separated into these items:

Locked - No geometric changes are allowed. The floors only produced by cubes, users can not change the shape because the other elements and details are based on the cube shape.

Constrained to plane - The attached face constrains to the movement of face of the plane. That means inside faces are followed with outside of the face of the plane. It is useful for objects attached to the floors like windows, doors, roofs. In this case, if roofs, windows, doors are moved, the height, width and depth of the cubes will be changed with them.

Lock to lower shell – This constraint uses top plane of the up cube as the new top plane and uses bottom plane of the bottom cube as the new bottom plane. In this way. The roofs should be produced up to the top plane and ground should be produced bottom to the bottom plane.

Lock Axis – The axis of roofs should be moved to the center of the bottom of the roofs. In this way, roofs can be easily located to the top of the top plane of the up cube.

Hidden – The constraint of Hidden is used to produce openings. Some parts of the shells should be hidden sometimes. For instance, to create a passageway or a hole, the part of the walls should be hidden.

Size – The width, depth and height of the cubes should be followed with an inverse proportion. The shape of building like a "pole" or a "flake" should be forbidden. On the other hand, all buildings should in a near size, which means the size of the cubes should be in a reasonable range.

Position – All buildings should not be crossed together, so the position of cubes of previous building needs to be recorded. And the gaps between buildings should also be taken into consideration.

First five constraints above are optional and the last two are essential which are already used in this project. All of them are simple but useful constraints. However there are still some discussible points in them. In constraint 2, if we attach geometry object on to a shell, such as a window, how the faces of the window follow the outside of the face of the plane is a problem, if the windows extended, the whole cube should be followed or just attached points will be moved. If answer is the latter one, how to deal with the unexpected twist and deformation is a question. In the constraint of hidden, it is a good method to produce a kind of "cube hole" effect, but the limit is also obvious. It is difficult to build an even relative complex inside view, like a gallery, which may not be a perfect cube transparent inside. However, the other requirements can be fulfilled if we use them appropriately.

5.2 Roofs

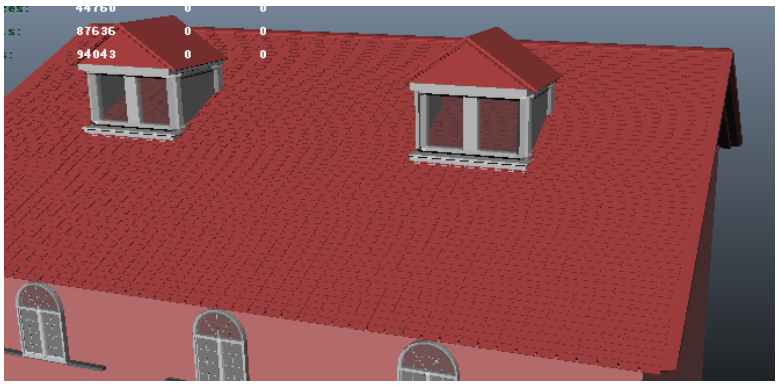


Figure 19 Roofs

P.J.Birch, S.P.Browne and V.J. Jennings (2002) present if we want to add a roof on a shell, this shell must not have any upper floors or an existing floor. The roof can also use shell options. Each face of the roof can be moved, expended. The vertexes of the roof can be moved as well. There are some other operations on the roof to control the ridge height raise or lower which followed overhang height adjustment.

In this project, there are four kind of roof style produced in total:

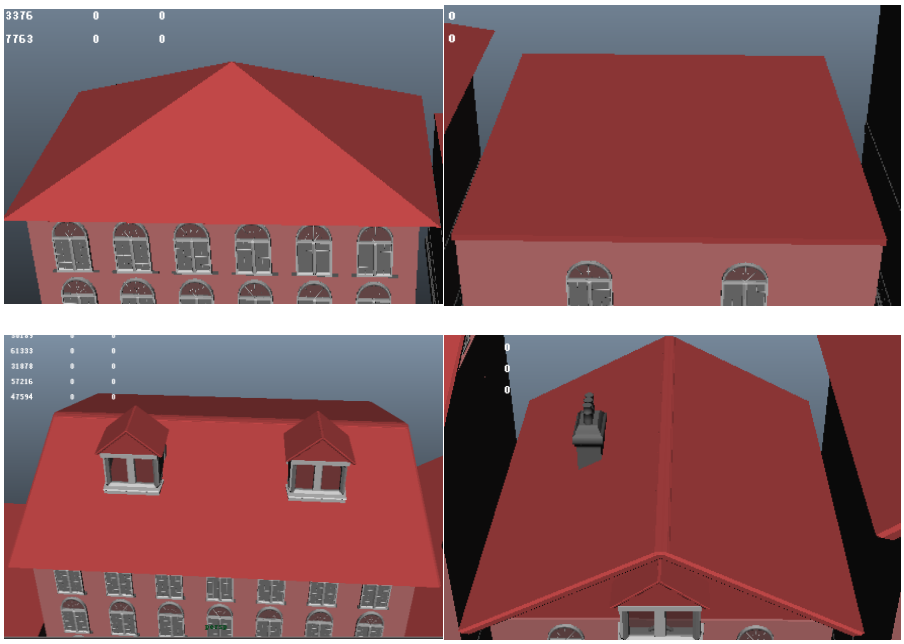


Figure 20 Four kind of Roof

The main step of the roof modeling is:

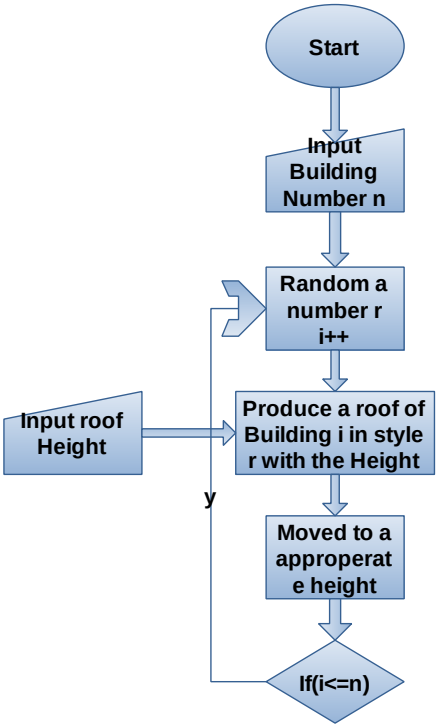


Figure 21 Flow Chart of Roof

Figure 23 indicates the process of producing roofs. Firstly, as mentioned before, users input the amount of buildings. The process of producing roof is included in the process of producing a single building, so the roofs are modeled repeatedly with the basis modeling. After that, the project will make a random number. Different intervals of the number stands for different kind of style roof, for example numbers 1-3 stand for the plane roof, so the probability of each kind is decided by the width of the interval.

For each roof, the users will give a height of roof when they input the attributes of the whole building. The project then uses this height and that random style to generate the roof. Within this process, the width and depth of the building will be past to the roof to decide its size. Finally, the roof will be translated along the Y axis which is contributed by the whole building height and half height of itself as its pivot is at the geometric center of the roof.

Style 1

Style number 1 is based on a Pyramid. The size of bottom plane is little bigger than the size of the building in order to covers everything below the roof. However, there is a problem when we move the roof to the appropriate place: In Maya, the Scale Y (Height of the Pyramid) is the length of hypotenuse but not the distance from the top vertex to the center of the bottom plane. In this case, we can`t use 1/2*Scale Y + HeightOfBasis as the number in axis Y of the roof. Looks like this:

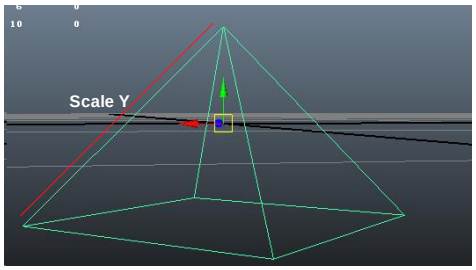


Figure 22 Pyramid Scale Y

Therefore, we need to calculate the Height of the Pyramid using Pythagorean proposition twice:

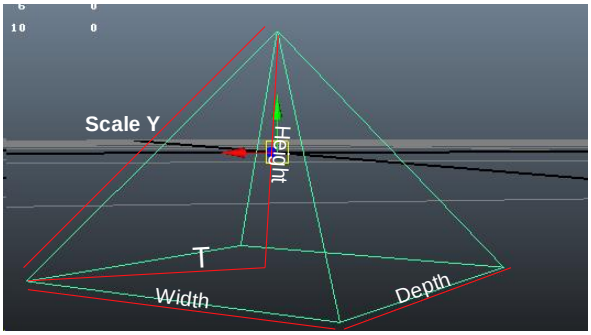


Figure 23 Calculation of Pyramid Height

$$T = sqrt((Width/2)^2 + (Depth)^2)$$
$$Height = sqrt((Scale\ Y/2)^2 - (T)^2)$$

Then the roof should be translated along Y with Height/2 + HeightOfBasis.

Style 2

Style number 2 is a flat roof which is a simple but common roof. To avoid some weird roofs (the ones in other styles but the heights of them are too small, Figure 26), one roof will be assigned to this Style 2 when the height of it is lower than 3. This roof is based on cube.

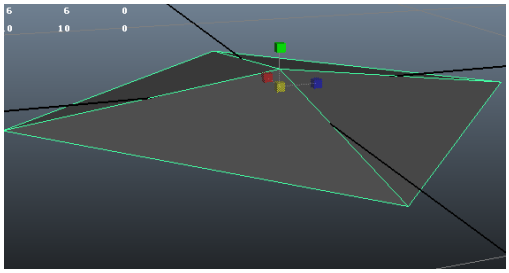


Figure 24 Weird Plane Roof

Style 3

In this roof, it composed by double cubes for the outside planes, double cubes for the middle planes, and one Prism for the bottom base.

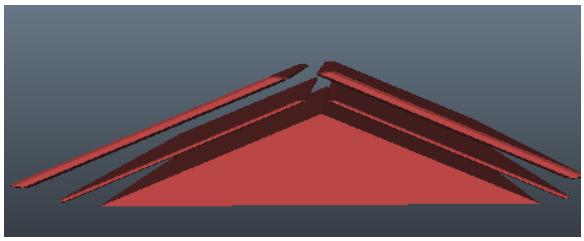


Figure 25 Component of Roof Style 3

The base of the roof is created as roof style 1: the width and depth are depended on the size of building and the height is decided by users. To calculate the position, size and direction of middle and top planes, trigonometric function need to be used.

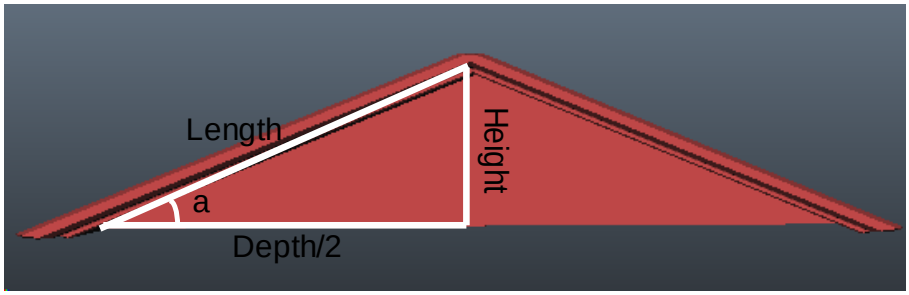


Figure 26 Trigonometric function of Style 3

$$a_{rad} = Arctan(Height/(Depth/2))$$
$$a_{deg} = rad\_to\_deg(a_{rad})$$
$$Roof_{left} = Rotate(a_{deg})$$
$$Roof_{right} = Rotate(-a_{deg})$$
$$Roof_{left} = Translate\ Y(HeightOfBasis+Height/2)$$
$$Roof_{right} = Translate\ Y(HeightOfBasis+Height/2)$$
$$Roof_{left} = Translate\ Z(Depth/4)$$
$$Roof_{right} = Translate\ Z(-Depth/4)$$

Style 4

The roof of style 4 is same to style 3 after rotated 90 degree. However, the width and depth should be exchanged in style 4.

This method is suitable for the roof which looks like a ridge in the middle with a gable on each side.

Constrains:

Height – The height of roof should be limited, because a needle roof is not accepted and a too thin roof will be treated as a plane roof.

Bottom Size – The size of bottom plane is depend on the size of the basis. The bottom size should be a little bit larger because the roof should cover the whole basis.

Probability – A good probability for each kind of roof style should be assigned. The standard of the assignment includes the truth and aesthetics. The probabilities in this project are 1/8, 1/8, 3/8, 3/8 respectively.

Pivot – The pivots of all roofs should be at the center of the bottom plane, because the pivots are not always at the half from the top vertex to the center of the bottom plane. In this way, the roof can be located in the coordinate much easier.

5.3 Windows

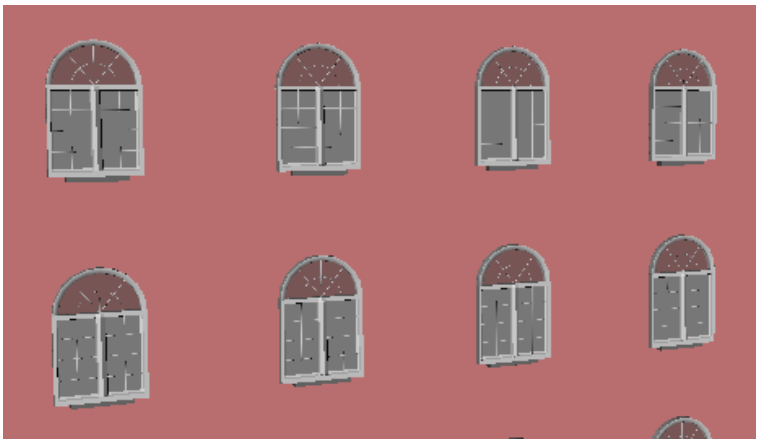


Figure 27 Windows

The main steps for window modeling are:

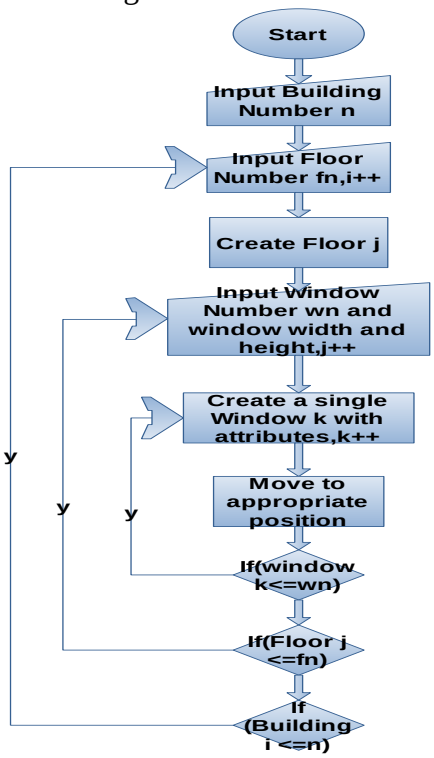


Figure 28 Flow Chart of Window

As showed in Figure 30, the process of producing a window is in three loops. Firstly, like before, the project receives the amount of buildings. In each building producing process, floors are created one by one. After users input how many windows they want, this number of windows will be generated in each floor. The position of them will be calculated when the number of them are confirmed.

The formula of position calculation is like this:

$$Gap = (FloorWidth - WindowWidth * WindowNumber) / (WindowNumber + 1)$$
$$Window_i\,Position = -FloorWidth / 2 + Window_i * Gap + (Window_i - 1) * WindowWidth + 0.5 * WindowWidth$$

In these formulas, the gap stands for the distance between two windows or between window and left or right edges. Therefore, the amount of gaps equals 1 + amount of windows. The width of gap is floor width minus the total width of all windows in one floor. The horizontal position of Window i is located from the left edge and equals sum of first gap and total width of previous windows.

"The window modeler aims to allow the designer to generate these differing window styles with the use of procedural modeling techniques and to interactively modify the structure of the window panels, window frame, frame profile, textures and interior decoration" (P.J.Birch, S.P.Browne, V.J. Jennings, 2002)

From their point, if we use texture mapping on the window, although we can generate pleasant results, the downfall of it will occurs when viewer looks from angles other than straight on. The windows need to be 3D entities with depth which permit viewer from various angles. Then the lighting algorithms can be used on it.

Therefore, in this project, the windows are 3D entities rather than just window textures. The window textures have been tried at first. It was very convenient to put them on the wall and replace them when we don`t like. However, the disadvantages seem bigger than this advantage: First of all, as mentioned before, if we change the viewer looks from angles other than straight on, the "truth" will very easily to be realized. Besides that, when the lighting and shadow involved, the texture can do nothing response. As the aim of this project is to provide a tool for animation and video games, the 3D effect is quite important, this bad performance is not allowed. Moreover, the texture of window should be very matched with the texture of basis, but obviously, it is not that easy to be achieved, the degree of coupling is high. Finally, one major feature of this project is the dynamic effect can be used easily, but the texture window will ruin it.

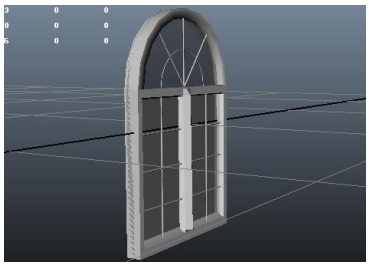


Figure 29 3D Window

“The underlying thread was that any window had an outer frame and internal to this frame a grid work of “lights” or openings, divided using mullions and transomes, within which were intricate decorations such as coils, cusps, leaded lights etc.”
(P.J.Birch, S.P.Browne, V.J. Jennings, 2002)

They define each window includes “parent panel” and “child panel”. A “parent panel” is the outer window frame including the panel within. The ‘child panels’ are the grid work of lights that occur within the outer window frame. Each “child panel” is the “parent panel” at the same time and its children are the “child panels” that are internal to it.

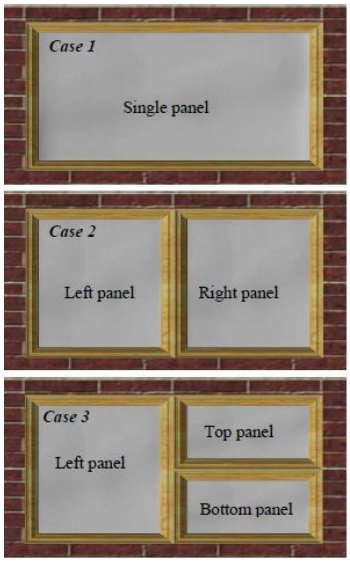


Figure 30 Window modeling principle

The windows can be subdivided easily by inserting mullions and transoms in this way. Each individual panel becomes a single encapsulated object within the width, height, texture, opacity and other details can be adjusted.

This method is worth using for reference but still hasn`t included every cases. If the window is not in a rectangle shape but in an irregular shape which is difficult to subdivide it, this algorithm seems can`t work anymore.

In this project, the window is produced in the way introduced above.

One window can be divided into these 4 parts as showed in Figure 33. The part 1 and part 4 are the frames, part 3 is the window and part 2 is the grid:

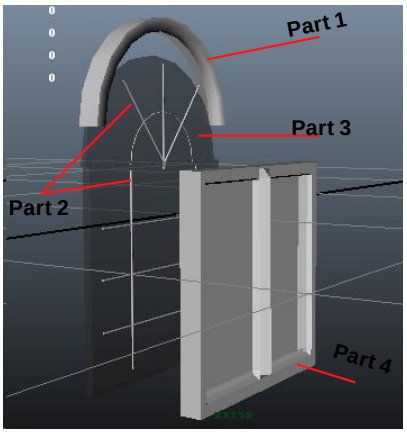


Figure 31 Component of Window

P.J.Birch, S.P.Browne, V.J. Jennings (2002) argue that, to define spline segments, a set of three splines should be referred – top-down, centerline and side-on profiles of the section of the object. The center and side splines are used for defining the shape that illustrates in Figure 34

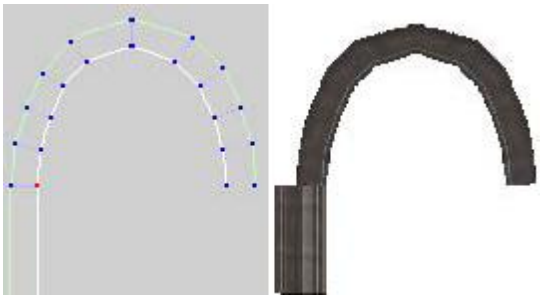
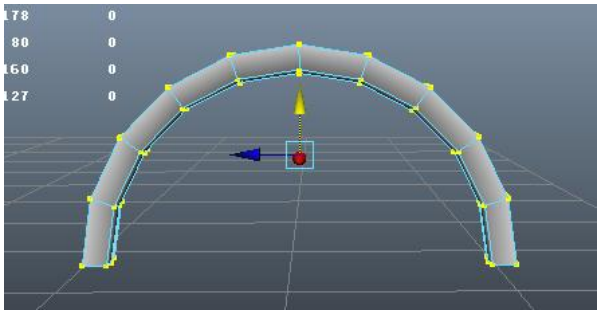


Figure 32 Archway created with splines

The part 1 follows this theory and these three splines can be founded in part 1, so the shape of the window frame is controlled by these splines. If the smoother face is required, we need only add some splines. If the shape is not satisfied, we need only change some of the splines by changing the vertexes on them:



Part 1 is created with these steps specifically: Firstly, produce a flat Cylinder and

rotate it to let the up plane faces to front. And then use Maya tool “Boolean difference” to get the edge by cutting a smaller Cylinder in the original Cylinder.

The up half of part 3 is an half of a flat Cylinder window and the bottom half is a flat cube window. They are assigned by transparent material.

The part 4 was a cube originally, and then some internal vertexes are moved to inside. The list of the vertexes are vtx[23:27], [31:35], [38:40], [42:43], [46:48], [50:51], [55:59], [62:66], [70:74], [78:82], [86:90], [94:98]. The list is showed below:

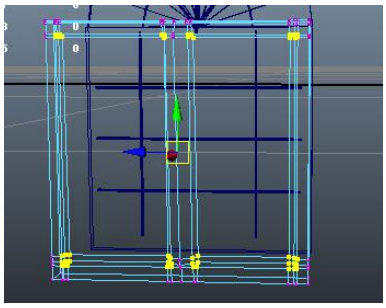


Figure 33 Vertexes of Window

The part 2 is created by the vertexes of the primitives which are these grids attached with. Then make small Cylinders between these vertexes. For example, the up grid is a Cylinder between the vertexes on the edge and the center point.

Constraints:

Grid – If one or some panels` width (height) changed, the other panels will be moved as well to compensate the difference.

Size Control – The height (width) of all panels will be changed at the same time if one of these panels is changed.

Size – The height of window cannot be bigger than the floor`s or cannot be too small. The total width of all windows in a floor can`t bigger than the width of the floor. Actually, for aesthetics, the windows can`t be too full for the floor, so if the size occupies the floor over a proportion, the size will be limited appropriately. The height of window is from 0.6 * Floor Height to 0.9 * Floor Height. The total width of windows is from 0.5 * Floor Width to 0.7 * Floor Width.

5.4 Doors

The door is only exist on the Ground Floor, so in each building generation, the door will be created only first time in the loop. Then project will use the input to model the door.

Figure 34 Doors

The main steps in modeling doors:

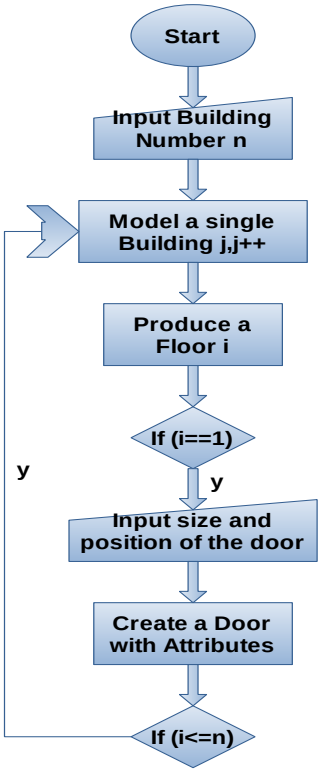


Figure 35 Flow Chart of Door

The position of doors can be changed by the user input. The axis Z is controlled by a percentage of the floor width and the axis Y is a fixed percentage of the floor height. The formulas of position are like here:

$$posZ = percentage * Floor\ Width$$

$$posY = Ground\ Floor\ Position - Floor\ Height\ /2 + Door\ Height * 0.5$$

For one door, overall, it can be divided into 4 parts as showed below. Part 1 is a window and the shell of the window which is created with the same technic as in the Chapter of Windows. Part 2 is the shell of the door with some pattern. Part 3 are the pillows with pattern. Part 4 is the main body of the door.

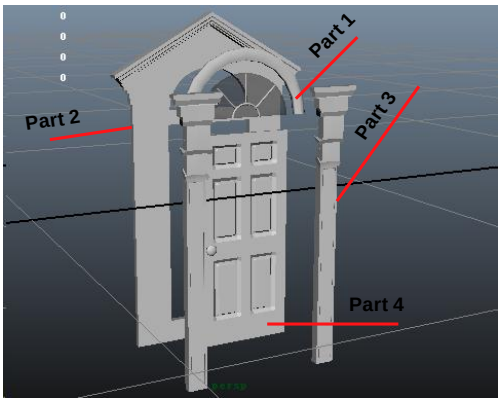


Figure 36 Components of the Door

For part 1, like in Windows, the P.J.Birch, S.P.Browne and V.J. Jennings`s spline segments theory also be adopted which means there are three kind of spline segments in the shell of the window. To create the shell of the window, we use Maya tool "Boolean difference" to get the edge by cutting a smaller flat Cylinder by a bigger flat Cylinder. Same as the Chapter of Windows, this window is also an half of flat Cylinder with a transparent material. The grid inside the window is Cylinders or Torus which are defined by the vertexes they attached. For example, the Cylinder below is defined by the vertexes vtx[120] and vtx[36]. The length of this Cylinder is the distance between these two vertexes and the slope is calculated by Trigonometric function with this fomula:

$$a_{rad} = arctan(vtx[120].Y - vtx[36].Y / vtx[36].Z - vtx[120].Z)$$

$$a_{deg} = rad\_to\_deg(a_{rad})$$

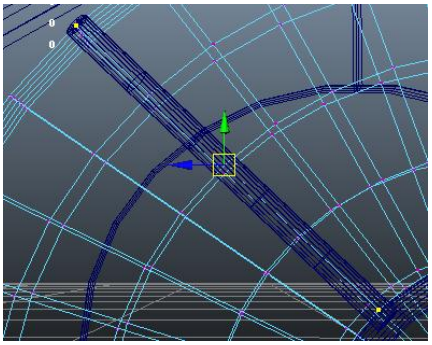


Figure 37 Grid of Door Window

Part 2 is the result of using Boolean difference to this shell, the window and the part 4. The pattern is made by moving vertexes, line by line:

Translate X vtx[45], vtx[44], vtx[75], vtx[151], vtx[121], vtx[120], vtx[186], vtx[48], vtx[181], vtx[187] = 0.1

Translate X vtx[156], vtx[182], vtx[157], vtx[170], vtx[183:184], vtx[169], vtx[185] = 0.08

Translate X vtx[161], vtx[158], vtx[159], vtx[163], vtx[172], vtx[175], vtx[171], vtx[174] = 0.06

...

P.J.Birch, S.P.Browne and V.J. Jennings introduce the top-profile is used to define the body of the object. In Figure 28 volutes is created by adding a series of indents in the top-profile.

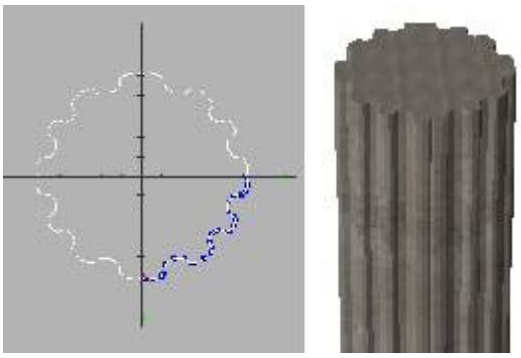


Figure 38 Column profile with 3D construction

The thickness is decided by first calculating the distance between the corresponding vertices of the center and side lines. Then add the length and the coordinates of the vertices to define the cross-section.

The pattern of part 3 uses this theory to model the pillars. The top-profile is created firstly, and then the Extrude tool in Maya is used to extrude the top-profile to a pillar object. The length is decided by the height of the part 4.

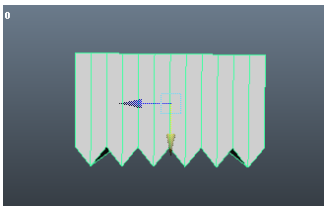


Figure 39 Top-profile of pillar

The part 4 uses the same technic as in part 1, which is moving the vertexes to make the hollow of the door.

Constrains:

Horizontal Position – As mentioned before, the horizontal position is decided by the percentage of Floor Width. However, no door is located at the very edge of a wall in the real world and if we input 0% or 100%, the half of the door will be out of the wall as the pivot is at the center of the door. Therefore, the percentage should be constrained from 5% + DoorWidth/2 to 95% - DoorWidth/2

Size – The too "fat "or too "thin" door is not allowed and the height of the door cannot be over the height of Ground Floor. In this case, the constrain should be HeightOfDoor <= HeightOfGroundFloor/1.5, HeightOfDoor/2<= WidthOfDoor<= 1.5 * HeightOfDoor.

Locked to GroundFloor – The door should be changed when the Ground Floor changed. The size of door should be changed when the size of Ground Floor changed. The door should

be translated when the Ground Floor is translated.

5.5 Bay Window

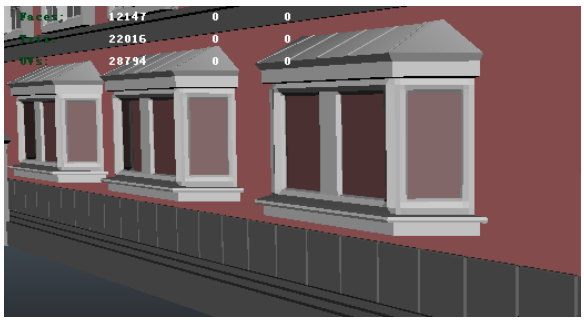


Figure 40 Bay Window

The main steps of Bay Window modeling are showed below:

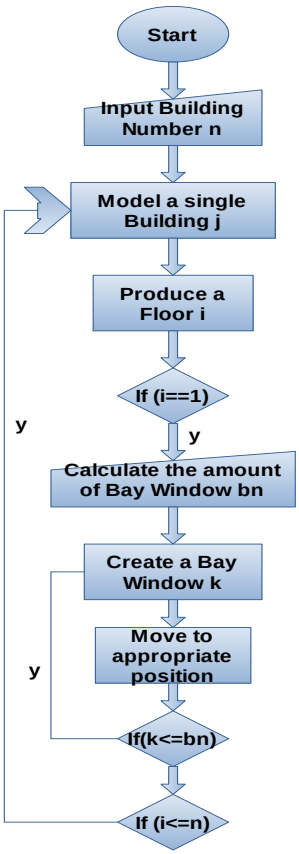


Figure 41 Flow Chart of Bay Window

As showed on this diagram, the Bay Window only exists on the Ground Floor. In the process of each building production, if the ground floor is producing, the amount of Bay Window should be calculated. After that, these Bay Windows will be located at appropriate position.

To calculate the amount of the Bay Window, the width of floor, the width of bay

window, the width of the door, the position of the door and the gap should be taken into calculatioin. The calculation formulas are:

$$gap = 0.3 * GroundFloorHeight$$

$$windNumL = (doorPosition+FloorWidth/2-BayWindowWidth/2-0.2)/(gap+BayWindowWidth)$$

$$windNumR = (-doorPosition+FloorWidth/2-BayWindowWidth/2-0.2)/(gap+BayWindowWidth)$$

In these formulas, the gap is defined by the project. Here it is decided by the Ground Floor Height. The Ground Floor is divided by the door into two parts, left and right. Each part calculates the amount of bay window individually. For example, in the left part, as the door position could be either positive or negative and the floor width/2 is the origin, the usable width should be door position plus floor width/2. To avoid the bay window overlap with the door or the door too close to the edge, the usable width should also minus half width of the bay window and a little gap. Then divide it by the sum of gap and bay window width. If the usable width is not enough for a bay window, nothing will be created.

For one bay window, it can be divided into these parts:

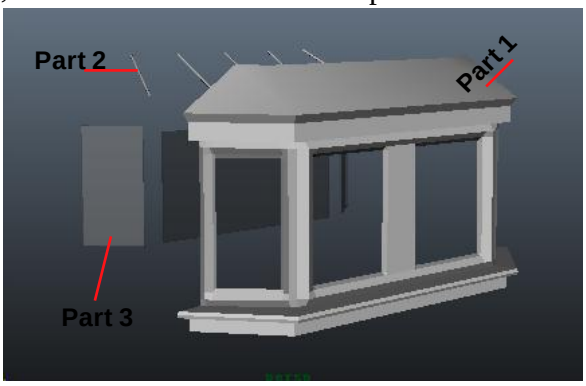


Figure 42 Components of Bay Window

Part 1 is the shell of the bay window. Part 2 is the pattern of the shell and part 3 is the window.

The part 1 is based on the 6-face-prism. It is half of the prism. The details on it are made by the Maya Extrude Tool and the movement of vertexes on it. Then use the Boolean difference tool to make the holes.

The windows are transparent cubes.

Constrains:

Forbid Overlap – The bay windows should not overlap with the door or overlap themselves.

Size – the size of the bay window is connected with the size of ground floor. The edge of the bay window should not be over the edge of ground floor

Locked to GroundFloor – The bay window should be changed when the Ground Floor changed. The size of bay window should be changed when the size of Ground Floor changed. The bay window should be translated when the Ground Floor is translated.

5.6 Bricks

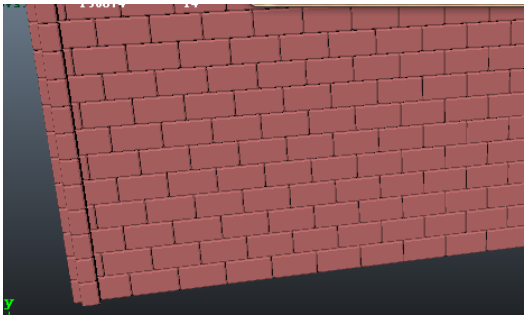


Figure 43 Bricks

The brick is optional when modeling a single building:

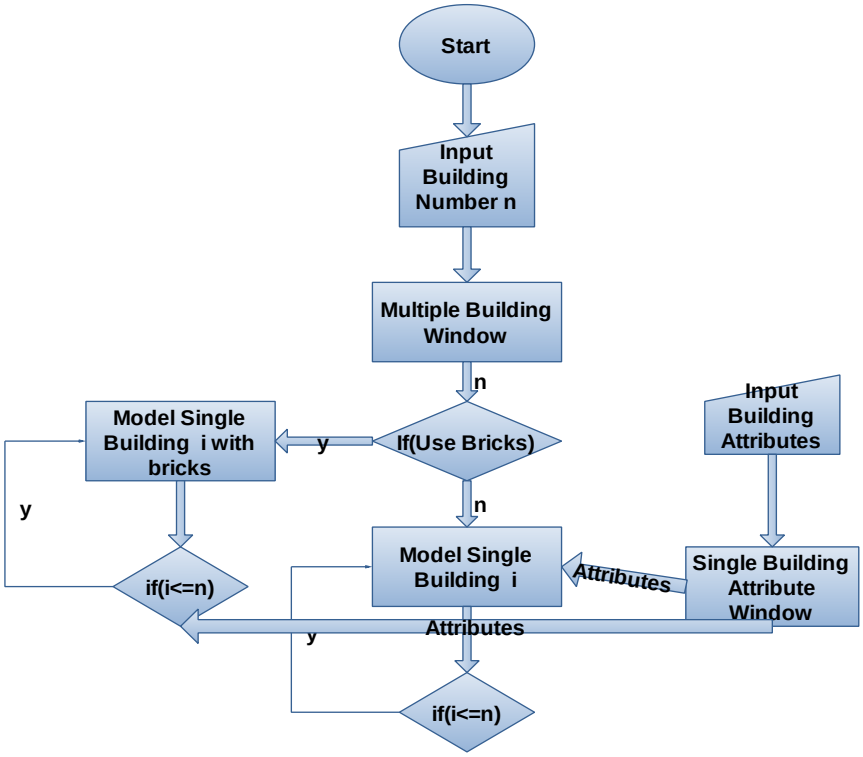


Figure 44 Flow Chart of brick

There is an option before a single building is produced, users can decide to use bricks or cubes to create the whole building. If bricks are used, the cubes will be replaced.

The calculation of amount of bricks is different for even and odd rows.

The amount of bricks in each odd row is calculated by these formulas:

$$TepNum_r = Floor\ Width / Brick\ Width$$
$$Remainder_r = Floor\ Width - Brick\ Width * TepNum_r$$
$$Num_r = TepNum_r + 1$$

The result of *Floor Width / Brick Width* cannot always be an integer. In Maya, the division will only calculate out an integer, so here, the TepNum is the number of full size bricks. The remainder is the width of that rest brick. Therefore, the total number of bricks in a row will be $TepNum_r + 1$

For the even rows, the formulas are different:

$$TepNum_r = (Floor\ Width - Brick\ Width/2) / Brick\ Width$$
$$Remainder_r = Floor\ Width - Brick\ Width * TepNum_r$$
$$Num_r = TepNum_r + 2$$

To make the displacement between two rows, in the even rows, half of one brick is created firstly, then the rest floor width will use the odd row`s algorithm to get the brick number. As a result, the total number could be $TepNum_r + 2$

The amount of bricks in each column is calculated by this formula:

$$TepNum_c = Floor\ Height * Floor\ Number / Brick\ Height$$
$$Remainder_c = Floor\ Height - Brick\ Height * TepNum_r$$
$$Num_c = TepNum_c + 1$$

The calculation of one column bricks is same as the calculation of one row bricks.

The calculation of position of bricks is different for even and odd rows:
For even rows

$$BrickPosition = (Brick_i\text{-}1)*BrickWidth + BrickWidth /2$$

$$RestBrickPosition = (Brick_i\text{-}1)*BrickWidth + BrickWidth /2 + Remainder/2 + BrickWidth /2 - BrickWidth /2$$

$$HalfBrickPosition = RestBrickPosition + Remainder /2 + BrickWidth/4;$$

For odd rows

$$BrickPosition = (Brick_i\text{-}1)*BrickWidth + BrickWidth /2$$

$$RestBrickPosition = (Brick_i\text{-}1)*BrickWidth + BrickWidth /2 + Remainder/2 + BrickWidth /2$$

The BrickPosition is position for the full size brick. HalfBrckPosition is the positon of an half of brick in even rows as mentioned before. The RestBrickPosition is the position of the remainder size brick. The BrickPosition is the total width of previous bricks plus a half of brick because the pivot is at the center of the brick. The RestBrickPosition is the total BrickPosition add half of itself. The HalfBrickPosition is the total BrickPosition add half of itself.

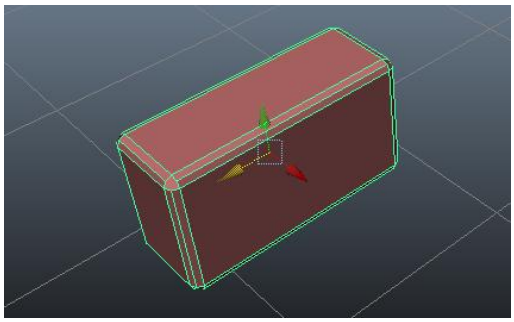


Figure 45 Brick Model

For one brick, it is based on a cube. The sharp edges of it are beveled with the offset = 0.02

The limit of using brick is also obvious. It consumes a lot of resources. As a result, the laptop can only support one or two brick-building. If model a city with brick-building, it will be very memory consuming and time consuming. In this case, if the dynamic effect or other effects are not involved, it is better not to use brick modeling.

The brick is the basis of using dynamic effect. For example, the effect of building explosion can be implemented directly; the building will be exploded into pieces of bricks.

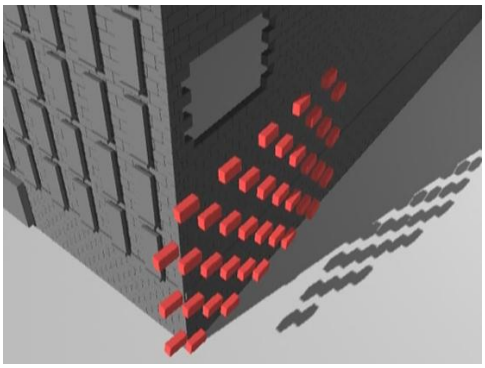


Figure 46 Effect of Explosion

Constrains:

Size – All the full size bricks should be in a same large. No size could be changed unless the parameters of calculation have changed at the same time.

Shape – No shape of bricks can be changed, because the other shape bricks may hard to build a building and will not suitable for the algorithm any more.

Lock movement – All bricks should be moved together unless there is particular requirement, for example, dynamic effect.

5.7 Tiles

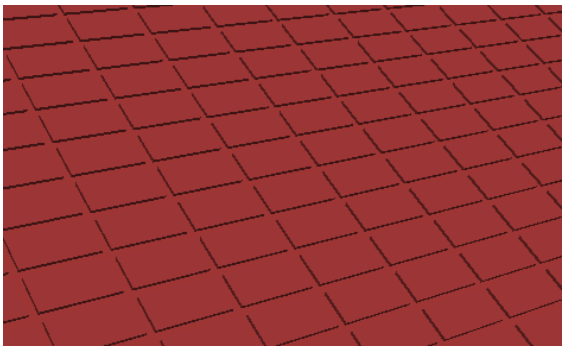


Figure 47 Tiles

The reason why separate the tiles from the roofs is the tile is optional. The main steps of modeling tiles are:

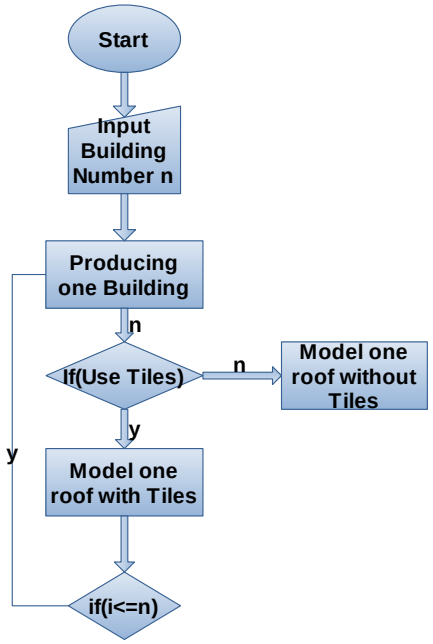


Figure 48 Flow Chart of Tile

As an optional choice, if users select to use tiles, the project will produce tiles up to the roofs. It happens after roof modeling. The tile is not used on the plane roof.

The amount of tiles is depend on the width of roof, the width of tile, the depth of tile, the depth of slope of roof.

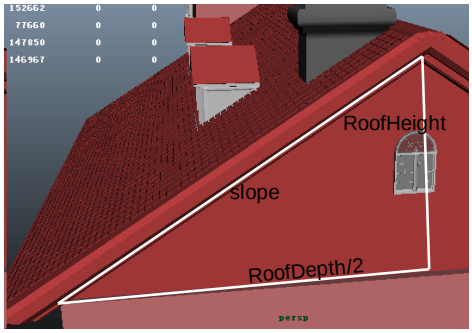


Figure 49 Pythagorean proposition of Roof

The formulas are:

*slope = sqrt(RoofDeepth/2*RoofDeepth/2+RoofHeight*RoofHeight)*

TrowNum = RoofWidth / TileWidth

TcolNum = slope / TileDepth

*RemainderZ = TrowNum * TileWidth*

*RemainderX = TcolNum * TileWidth*

rowNum = TrowNum + 1

colNum = TcolNum + 1

In these formulas, the slope is calculated by Pythagorean proposition. Like bricks, *RoofWidth / TileWidth and slope / TileDepth* can`t be always integers, so *TrowNum and TcolNum* are the amount of full size tiles. Remainder is the width of the rest brick. The rowNum is the amount of bricks in a row. The colNum is the amount of bricks in a column.

Firstly, we arrange the tiles horizontally:

```
for(column i=1; i<= TcolNum;i++)
{
for(row j=1;j<= TrowNum;j++)
{
moveX = (TileDepth/2+(i-1)*TileDepth);
moveY = RoofY;
moveZ = TileWidth/2-RoofWidth/2+(j-1)*TileWidth;
}
j++;
moveZ = TileWidth/2-RoofWidth/2+(j-1)*TileWidth + TileWidth /2 + RemainderZ/2
}
moveX = (TileDepth/2+(i-1)*TileDepth) + TileDepth /2 + RemainderX/2
```

After that, we rotate the whole group of tiles. The degree of rotation is calculated by the trigonometric function

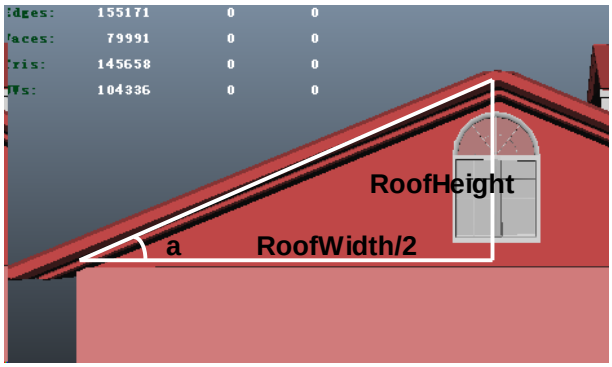


Figure 50 Trigonometric function of Tile

a_{rad} = *atan(height/(deepth/2)*

a_{deg} = *rad_to_deg*(a_{rad})

Rotate a_{deg}

Then move the tile group to the appropriate position

Move FrontTiles = depth/4

Move BackTiles = -depth/4

The limit of using tile is like brick. Hundreds or even thousands of bricks will be created for one building, so it is very resource consuming. The tiles can use dynamic effect directly as well.

One brick is based on a flat cube and the edges of it are beveled.

Constrains:

Lock to Roof – The movement and rotation of tiles should be locked to roof. When the roof is moved or rotated, the tiles should be followed.

Style – The tiles do not exist on some particular roof, for example, the plane roof.

5.8 Other Components

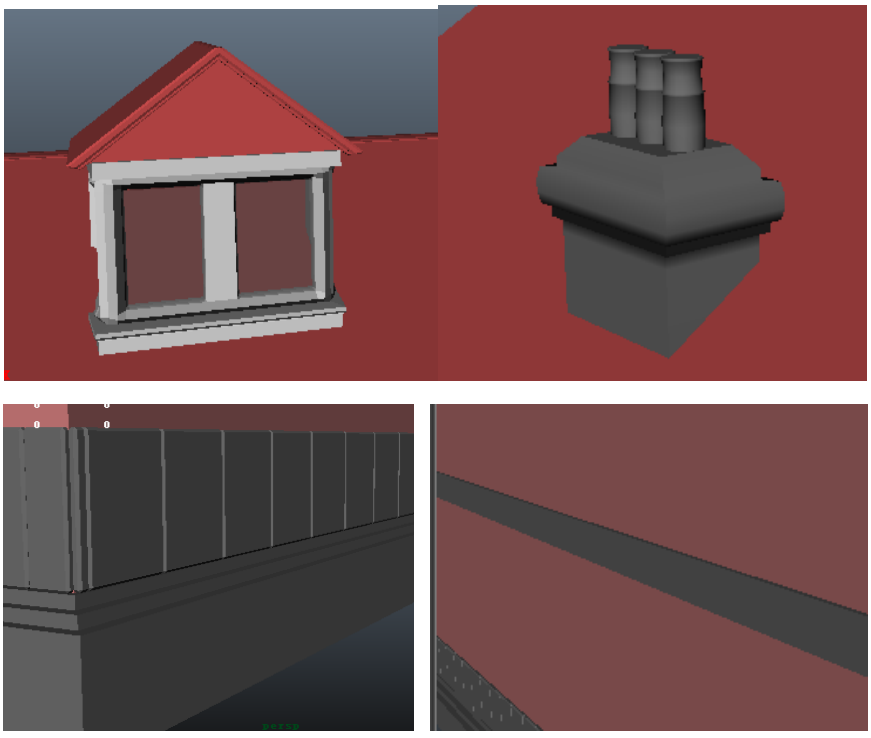


Figure 51 Other Components

These components are generated automatically and not optional:

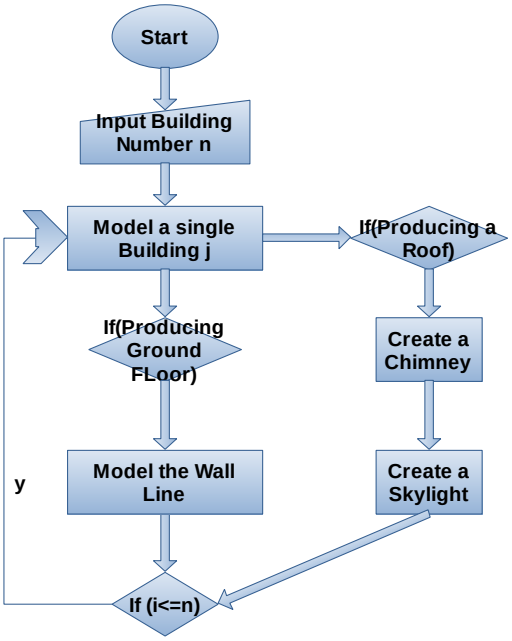


Figure 52 Flow Chart of Other Component

These components are the accessory of a building. When the ground floor is producing, the wall lines will be created. The size of them is depends on the size of the floor and they will be attached to the walls. There are three layers of wall lines, and their position can be calculated as

$$Line\ up\ X = Width/2$$
$$Line1\ X = Width/2$$
$$Line2\ X = Width/2 + 0.2$$
$$Line3\ X = Width/2 + 0.4$$

$$Line\ up\ Y = First\ Floor\ Height$$
$$Line\ 1\ Y = First\ Floor\ Height/16$$
$$Line\ 2\ Y = First\ Floor\ Height/16 + 0.1$$
$$Line\ 3\ Y = First\ Floor\ Height/16 + 0.2$$

Above to these lines, there are a row of decorative bricks. The number of bricks is calculated as:

$$TepNum = Floor\ Width / Brick\ Width$$
$$Remainder = Floor\ Width - Brick\ Width * TepNum$$
$$Num = TepNum + 1$$

The position is calculated by these formulas:

```
for($i=1;$i<=$brickNumZ;$i++)
{
    Move X = Floor Depth/2
    Move Y = 3*GroundFloorHeight/16+0.3
    Move Z = FloorWidth/2+BrickWidth/2+(i-1)*BrickWidth
}
```

On one roof, one chimney and one or two skylights will be created.

For the skylight, it can be divided into these parts:

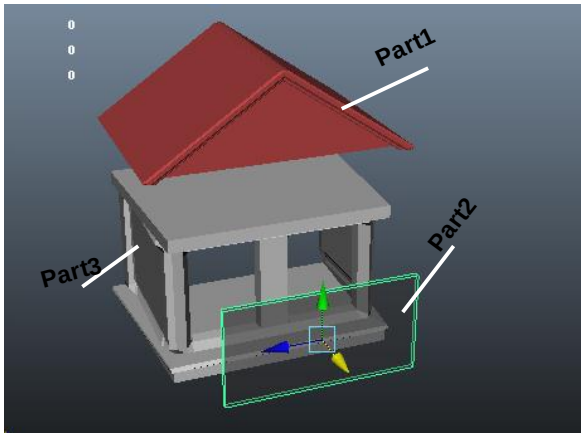


Figure 53 Components of Skylight

The part 1 is modeled as a roof which has introduced before. The part 2 is a flat cube with transparent material. The part 3 is made as the bay window. However in the bay window, it is based on 6-face prism while in the skylight, it is based on 8-face prism.

The position of it is
Move Y = GroundHeight + FloorHeight*FloorNum + RoofHeight/2
Move X = RoofDepth/2
Move Z: if (RoofWidth > Standard)
{Skylight1 = length/4
Skylight2 = -length/4}
Else
{Skylight = length/2}
If the roof width is enough, there will be two skylights created. Otherwise, only one roof will be modeled instead.

For the chimney, it can be divided into these parts:

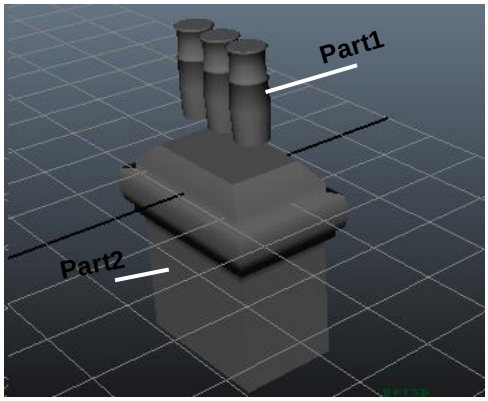


Figure 54 Components of Chimney

The part 1 is based on a Cylinder and some vertexes are modified. The part2 is based

on a cube and up edges and some vertexes are modified.

Constrains:

Lock to Wall – When the wall is moved or rotated, the wall lines should be followed.

Lock to Roof – When the roof is moved or rotated, the skylights and chimney should be followed.

Amount of Skylights – If the width of roof is smaller than 20, only one skylight will be modeled. Otherwise, there will be 2 skylights.

5.9 City Map

Voronoi Diagram can be defined as: Voronoi Diagram is a continuous polygon created by a group which composed by a perpendicular bisector of a straight line connecting two adjacent points. N differentiated points on the plane divide the plane with the principle of most adjacent; each point associated with its nearest neighbor region. Delaunay triangle is the triangle connected by the correlative points which are the sharing points on the adjacent Voronoi polygons. The Delaunay triangle`s circumcircle center is the vertex of Voronoi polygon which associated with this triangle. Voronoi triangle is the bipartite graph of Delaunay graph:

For a given initial point set P, there are several Triangulations. Delaunay triangulation has the following characteristics:

- Delaunay triangulation is unique;
- The outer boundary of the triangle network constitutes convex polygon "shell" of a point set P;
- There is no any point in the internal of the circumcircle of the triangle. Conversely, if a triangular network satisfies this condition, then it is the Delaunay triangulation.
- If the minimum angles of each triangle in the triangulation network are in ascending order, the number of Delaunay triangulation rank is biggest, in this meaning, Delaunay triangulation network is the triangulation network which closest to the regularization.

Delaunay triangulation network characters also can be expressed as the following

features:

- In the Delaunay triangulation network, within the circumcircle of the triangle no other point, known as empty circle features;
- when the network is building, it always select the closest neighbor points to build triangles and does not intersect with the constraint segment;
- The built triangle network always has the optimal shape characters, if the diagonals of convex quadrangle which are formed by two arbitrary adjacent triangles can be interchanged, then the smallest angle of the 6 inside corners of two triangles would not be getting larger ;
- Wherever the network is built from in the region, the result will be consistent.

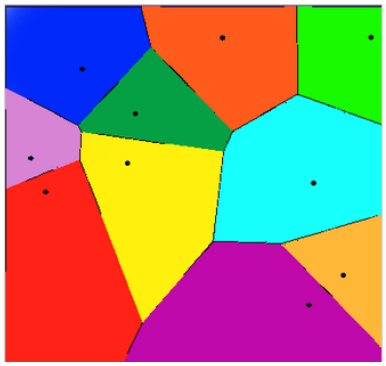


Figure 55 Voronoi Diagram

The generating of the map is based on the Voronoi Diagram. The main step is:

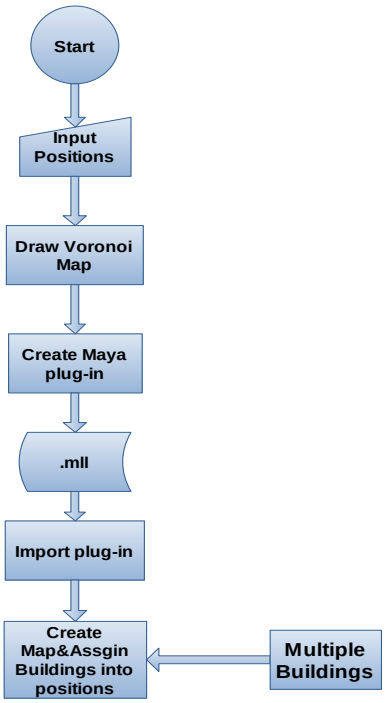


Figure 56 Flow Chart of City Map

The Voronoi Diagram part is implemented in the Maya C++ API. There are two C++ projects: the first one is used for users to draw the location points. The second one is used for creating Maya plug-in by these data. Then we use this plug-in to create the City Map in Maya. Finally, put these buildings on the map.

Voronoi Diagram Drawing steps:
Draw some points on a MFC window and each point stands for the position of each building:

Figure 57 Drawing Points

This is implemented by C++ MFC. Users can draw points on the board anywhere and then this program will build the Voronoi Diagram.

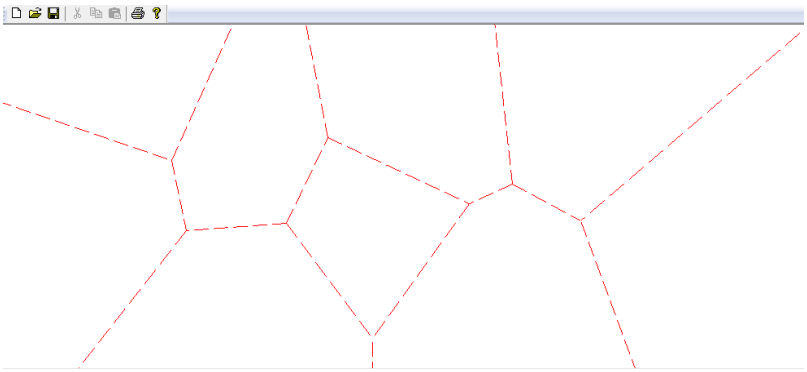


Figure 58 Build Voronoi Diagram

As we introduced above, Voronoi Diagram is created by a group which composed by a perpendicular bisector of a straight line connecting two adjacent points.

The data of these position points and the lines will be recorded in a file.

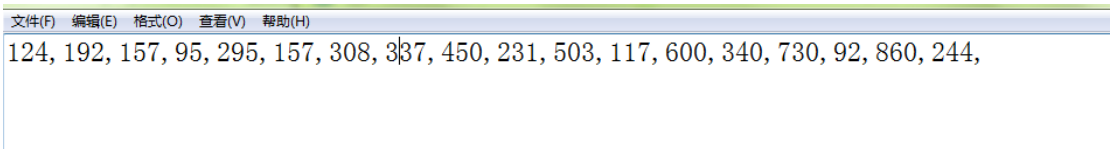


Figure 59 Data of Voronoi Diagram

Then the data will be read in another C++ project and a Maya plug-in file (.mll) will be created. After that, Maya will import this plug-in and execute it. An exactly same map will be created in Maya. Each original point is represented by a small sphere and the lines are represented by straight curves:

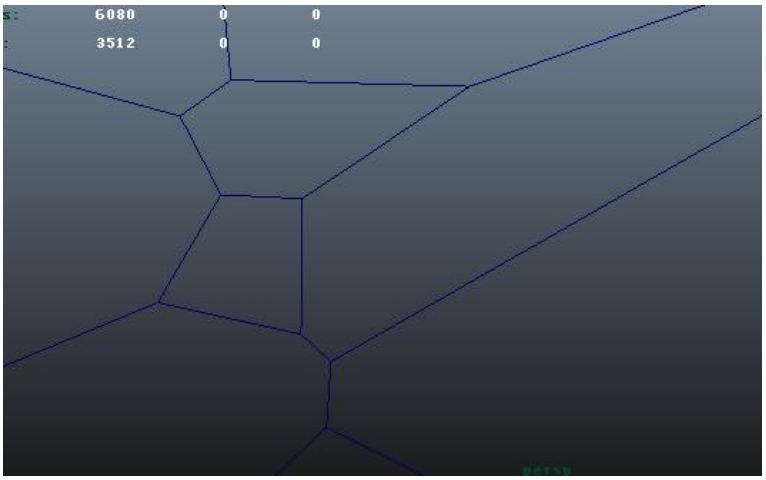


Figure 60 City Map in Maya

Finally, locate all modeled buildings on these positions.

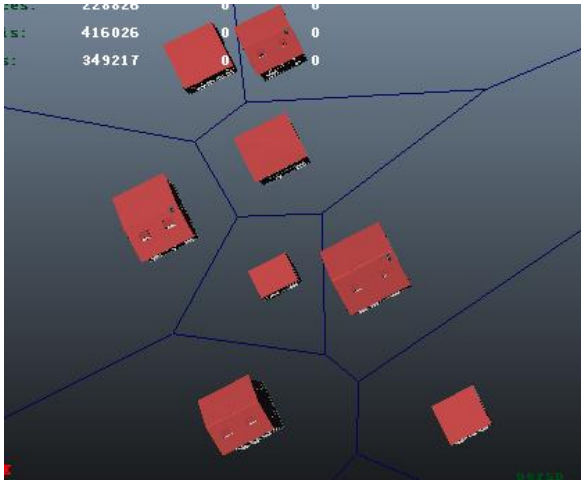


Figure 61 Locate the Buildings on the map

Constrains:

Avoid Overlap – No overlap is allowed when the map is not good

Reject position – When the space of a polygon is not enough for the building, this polygon will be rejected.

5.10 User Interface

The user interface is simple but covers almost all functions of this project.

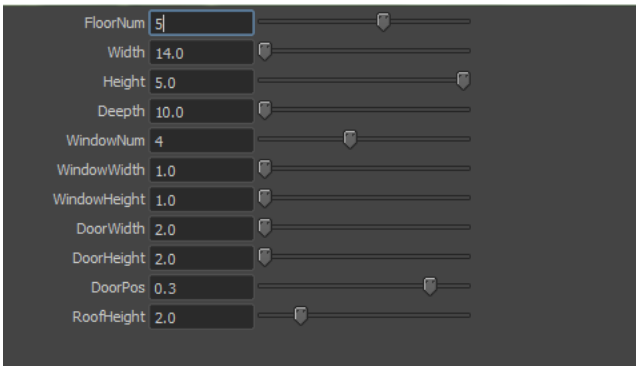


Figure 62 Window of Attributes of Single Building

In this window, user can modify various attributes of one single building. When any attribute changed, the old building will be deleted and new building will be created with these updated attributes.

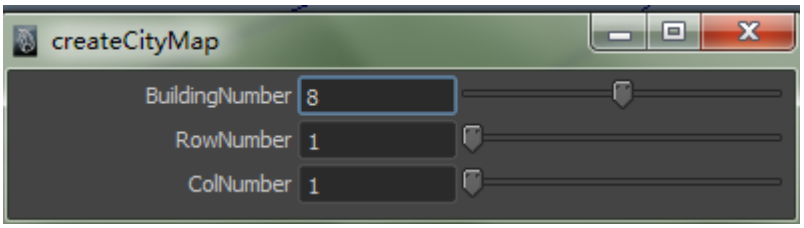


Figure 63 Window of Building Number

In this window, users can decide how many buildings they want and how to arrange them. All buildings are created with random attributes. When a new number input, all old buildings will be deleted, this is to make sure different buildings will be generated when the input changed.

6 Result Analysis

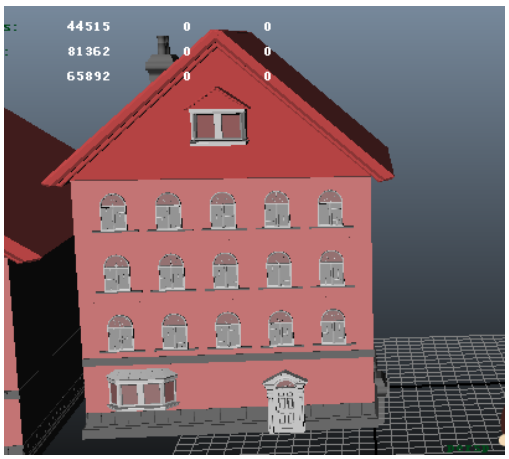


Figure 64 Single Building Result

For one building, the final effect of it looks reasonable and detailed. All components are in logical sizes and stay at appropriate position. All constrains mentioned before can be reflected here. Every basic element of architecture is implemented in this building.

The interaction between users and models works well, every input responded immediately and accurately unless it exceeds the constraints. If the result exceeds the constraints, the project can adjust it automatically.

In terms of realistic, the models are created in detail and even some patterns are created on the objects. Any angle of view is accepted as all objects are in 3D rather than using textures.

However, there are still some unsatisfied points. For one thing, the variety of element in it is not rich, only basic elements are involved. What is more, the back face of the building lacks adornment.

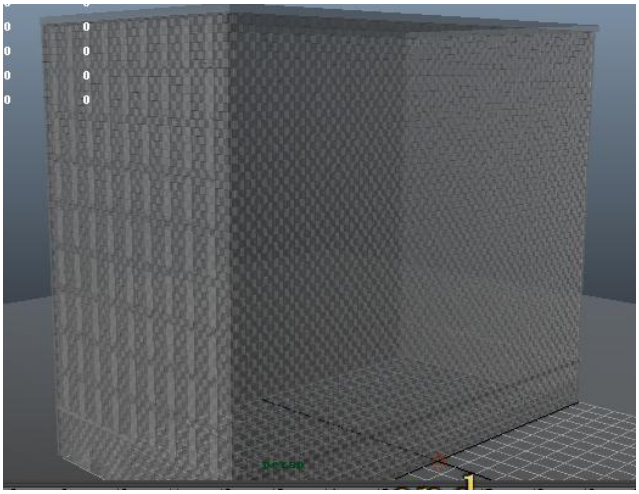


Figure 65 Result of Single Building with Bricks

For one building created by bricks, the entire building looks further real. All bricks are organized well and each wall has the same size as the one created by cube. No interspace exists in the building, so it looks very smooth and trim.

The dynamic effect has also been tested. The performance is excellent: we need only put force fields near it and the building then can be destroyed into bricks.

The consumption is the only problem of it. Only one or two these brick-building can be created in the laptop because of the memory limitation.

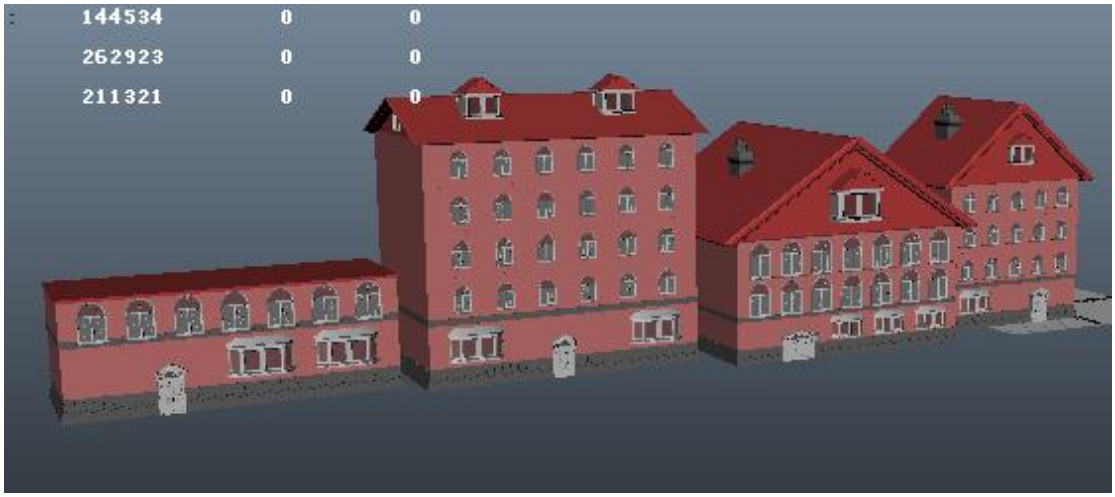


Figure 66 Result of Multiple Buildings

The performance of multiple building is also good. All buildings are in logical even with the random attributes. Each building is different from the others also because of the random attributes. All of their components are stayed in a size level, no one is ridiculous big or small.

However, sometimes the combination of these random attributes may lead some not very real buildings. For example, in the figure above, the very left building is not common in the real world.

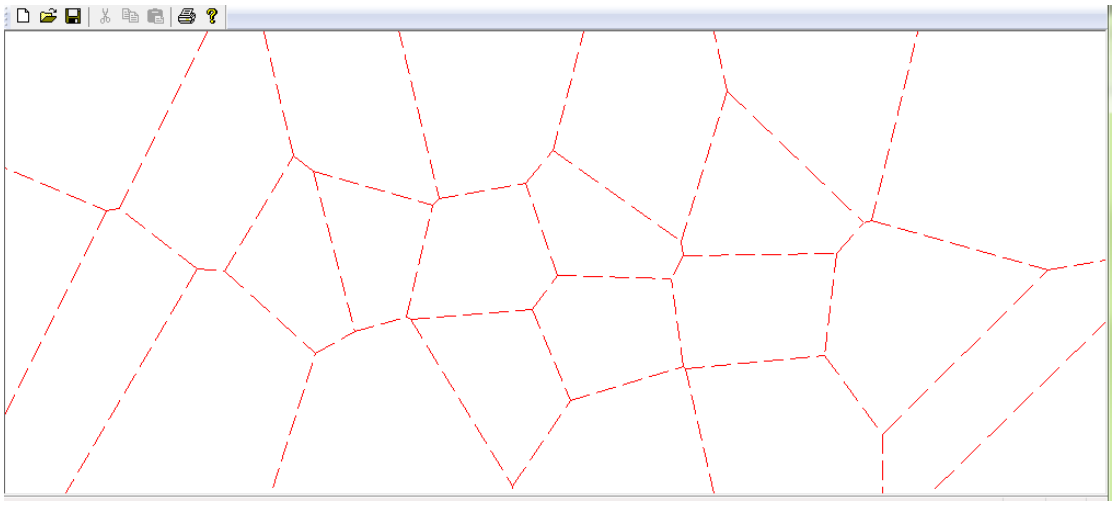


Figure 67 Result of Drawing Voronoi Diagram

For the Voronoi Diagram, users can draw as many points as they want at any position on the board. The Diagram will be draw no matter how complex the input points are. And all the data of this diagram are recorded successfully.

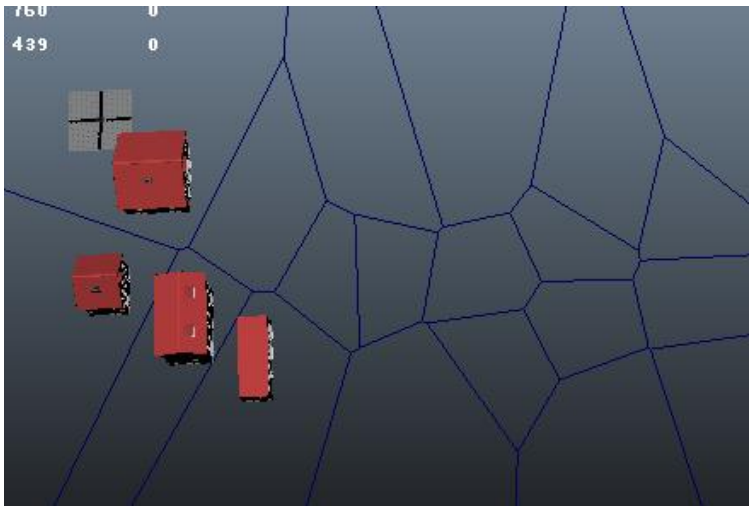


Figure 68 Result of Creating map and assign buildings on it

Buildings can assign to the map appropriately and the constraints introduced above are worked.

On the other hand, the problems of the map can`t be denied. This map relies on the users input too much, so there are some potential problems in it. For the first, if the size of the map is too small, the arrangement of buildings will be in disorder. In addition, the direction the buildings face to is hard to be decided as there seems no a good standard. Lastly, if the number of buildings is over the position the map provides, the rest buildings will stay at original place.

7 Evaluation

The first object of this project was individual building controlled which has been achieved. Various building attributes are under user`s control, include not only the attributes of the entire building, but also the attributes of the components of the building. When any attribute is updated, the entire building will be immediately rebuilt under the constraints. This is a very essential function of my project from start to finish. However, one attribute, which was on the plan, has not involved-the style of the architecture. Limited time is the main reason of it. There is no time to implement other styles as it is very time consuming to code all components of a building of one style.

The second object was multiple building creations which also have been done. A number of buildings could be produced with random attributes and this number is given by the users. As the attributes are assigned randomly for each building, all buildings should be different. This object is also a core object which has to be

achieved.

The third object was “build architecture with bricks”. The project has also achieved this object, although the efficiency of execution is a problem can`t be solved. Several methods have been attempted, but there was almost no improvement because of the limitation of Maya. This object was a critical and high level object. The final effect is good.

The fourth object was “assign appropriate texture to architectures”. This project didn`t achieve this level mainly because the time. This object is very important for improving the realistic of the buildings, but doesn`t affect the function of this project.

The final object was “city map production” This function has been finished partly. Users can draw Voronoi Diagram and import it into Maya as a Maya-plugin. After that, a city map will be created and the modeled buildings will be arranged to this map. However, the effect is not ideal enough as some functions not yet implemented. For example, the direction these building face to cannot be adjusted with the changing of their position and the buildings can`t adapt a ridiculous map.

8 Further Work

As introduced above, due to the lack of time, this project still has many points could be improved and many problems should be solved. First of all, the details on a building are far more enough to fulfill the requirement of animation or video games. Therefore, more components and details should be added on to the buildings. Furthermore, the type of building is too monotonous. More different kind of styles should be optional. A good method for solving the low efficiency of execution when the bricks involved is necessary. Moreover, the function of the city map should be completed.

Bibliography

(1) Entertainment software Association 2011, "Essential Facts About The Com puter And Video Game Industry", [Online] http://www.theesa.com/facts/pdfs/ESA_EF_2011.pdf.

(2) Müller, P., Tijl Vereenooghe 2006, "Procedural 3D Reconstruction of Puuc Buildings in Xkipché, The 7th International Symposium on Virtual Reality", Archaeology and Cultural Heritage VAST.

(3) Havemann ,S. 2005, "Generative Mesh Modeling", Institut für ComputerGraphik,.

(4) Yoav I H Parish, Müller ,P. 2001, Procedural Modeling of Cities, SIGGRAPH , Pages 301-308.

(5) Birch, P.J.& Browne, S.P.& Jennings, V.J. & Day, A.M. 2002, "Rapid Procedural-Modelling of Architectural Structures", ACM Inc.

(6) D.K. Ching 1995, "A Visual Dictionary Of Architecture", John Wiley & Sons, Inc.

(7) Focillon, H& Kubler, G 1992, "The Life of Forms in Art", Zone Books.

(8) Havemann, S 2005, "Generative Mesh Modeling", Institut für ComputerGraphik.

(9) Kandinsky, W 1991, "Compositions, by the Museum of Modern Art".

(10) Muller,P&Wonka, P&Haegler,S&Ulmer,A&Gooll,V2006,"Procedural modeling of buildings", SIGGRAPH 2006: ACMSIGGRAPH 2006 Papers,pp. 614–623.

(11) Wonka, P& Wimmer, M& Silion, F& Ribarsky, W 2003, "Instant Architecture", ACM Inc.

(12) Wonka, P 2006, "Procedural Modeling of Architecture", Siggraph.

(13) Lanier,L 2011, "Maya studio projects : Texturing and lighting", Sybex.

(14) McKinley, M 2010,"Maya studio projects : game environments and props" Sybex.

(15) Mark R., Kazmier, C 2005,"MEL scripting for Maya animators", Morgan Kaufmann.

(16) David, A.D 2003,"Complete Maya programming : an extensive guide to MEL and the C++ API ", Morgan Kaufmann .

(17) Greuter S., Parker J., Stewart N., and Leach G. 2003, "Real-time procedural generation of "pseudo infinite" cities", In Proceedings of GRAPHITE, ACM Press, Pages 87-95.

(18) Kelly .G, McCabe ,H. July, 2006, "Interactive Generation of Cities for Real-Time Applications", SIGGRAPH,

(19) Brand ,S 1994. "How buildings learn", Viking.

(20) Eppstein, D. 1999, "Raising roofs, crashing cycles, and playing pool" Discrete & Computational Geometry 22(4):569-592.

(21) William ,M. 1944, "The Logic of Architecture", Massachusetts Institute of Technology.

(22)Lechner, T. 2003, “Procedural City Modeling”, Midwestern Graphics.

Appendices - Essential Source Code

CreateFloors_cube8.mel:

```
global proc creatFloors(int $buildNo,float
$firstFloorH,int $floors,float $width,float
$h,float $deepth,
int $windNum,float $windWidth,float
$windHeight,
float $doorWidth,float $doorHeight,float
$doorPos,
float $roofHeight)
{
int $i;
string $name1;
string $buildName = "ZY"+$buildNo;
group -em -name $buildName;
if($buildNo!=1)
{
$firstFloorH = $firstFloorH*rand(0.9,1.5);
}
$height = $h*($floors-1)+$firstFloorH;
creatRoofs($buildNo,$width,$deepth,$roofHeig
ht,$height,$h,$windHeight);
for($i = 1;$i <= $floors; $i++)
{
$name1 = $buildName+"_floors"+$i;
polyCube -name $name1;
parent $name1 $buildName ;
hyperShade -assign lambert2 $name1;
float $m = ($i-1.5)*$h+$firstFloorH;
if($i>1)
{
scale -r $deepth $h $width;
move -r 0 $m 0 $name1;
creatWindows($buildNo,$windNum,$windWidt
h,$windHeight,$width,$h,$deepth/2,$m,$name1
);
}
else
{
$fheight = $firstFloorH;
$mfheight = $fheight/2;
scale -r $deepth $fheight $width;
move -r 0 $mfheight 0 $name1;
float $doorsPos =
creatDoors($buildNo,$doorWidth,$doorHeight,$
width,$fheight,
$deepth/2,$mfheight,$doorPos);
creatBayWindows($buildNo,$windNum,$wind
Width,$deepth/2,$width,$firstFloorH,$doorsPos,
$doorWidth);
creatFootline($buildName,$fheight,$mfheight,$
width,$deepth);
}
FreezeTransformations;
}
};
```

CreateFloors_brick1.mel

```
global proc creatBrickFloors(string
$buildName,float $width,float $height,float
$deepth)
{
int $brickYNum = $height/0.3;
float $brickYspace = $height-$brickYNum*0.3;
int $brickZNum = $width/0.54;
float $brickZspace = $width-$brickZNum*0.54;
int $brickXNum = $deepth/0.54;
float $brickXspace =
$deepth-$brickXNum*0.54;
float $brickY = 0.15;
float $brickZ = 0.27;
float $brickX = 0.27;
float $moveX = $deepth/2-0.1;
float $moveNX = -$deepth/2+0.1;
float $moveZ = $width/2-0.1;
float $moveNZ = -$width/2+0.1;
$groupnameF = $buildName+"_brickFront";
group -em -name $groupnameF;
```

```
parent $groupnameF $buildName;
$groupnameB = $buildName+"_brickBack";
group -em -name $groupnameB;
parent $groupnameB $buildName;
$groupnameL = $buildName+"_brickLeft";
group -em -name $groupnameL;
parent $groupnameL $buildName;
$groupnameR = $buildName+"_brickRight";
group -em -name $groupnameR;
parent $groupnameR $buildName;
float $polyBeveloffset = 0.02;
polyCube -w 0.2 -h 0.3 -d 0.54 -name
brickExample;
polyCube -w 0.2 -h 0.3 -d $brickZspace -name
brickExampleZ;
polyCube -w 0.2 -h $brickYspace -d 0.54 -name
brickExampleY;
polyCube -w 0.2 -h $brickYspace -d
$brickZspace -name brickExampleYZ;
polyCube -w 0.54 -h 0.3 -d 0.2 -name
brickExample1;
polyCube -w $brickXspace -h 0.3 -d 0.2 -name
brickExample1X;
polyCube -w 0.54 -h $brickYspace -d 0.2 -name
brickExample1Y;
polyCube -w $brickXspace -h $brickYspace -d
0.2 -name brickExample1XY;
polyBevel -offset $polyBeveloffset -segments 2
brickExample;
polyBevel -offset $polyBeveloffset -segments 2
brickExampleZ;
polyBevel -offset $polyBeveloffset -segments 2
brickExampleY;
polyBevel -offset $polyBeveloffset -segments 2
brickExampleYZ;
polyBevel -offset $polyBeveloffset -segments 2
brickExample1;
polyBevel -offset $polyBeveloffset -segments 2
brickExample1X;
polyBevel -offset $polyBeveloffset -segments 2
brickExample1Y;
polyBevel -offset $polyBeveloffset -segments 2
brickExample1XY;
for($bY=1;$bY<=$brickYNum;$bY++)
{
for($bZ=1;$bZ<=$brickZNum;$bZ++)
{
$brickName =
$buildName+"_brick_front"+($bY-1)*$brickZN
um+$bZ;
duplicate -name $brickName brickExample;
parent $brickName $groupnameF;
$brickZ = ($bZ-1)*0.54+0.27;
move -r $moveX $brickY $brickZ;
if($bY%2==0)
{
move -r 0 0 -0.27;
if($bZ==1)
{
setAttr ($brickName+".scaleZ") 0.5;
move -r 0 0 0.135 $brickName;
}
if($bZ==$brickZNum)
{
$brickName =
$buildName+"_brick_front"+($bY-1)*$brickZN
um+$bZ+1;
duplicate -name $brickName brickExampleZ;
parent $brickName $groupnameF;
$brickZ =
($bZ-1)*0.54+0.27+$brickZspace/2+0.27-0.27;
move -r $moveX $brickY $brickZ;
$brickName =
$buildName+"_brick_front"+($bY-1)*$brickZN
um+$bZ+2;
polyCube -w 0.2 -h 0.3 -d 0.27 -name
$brickName;
polyBevel -offset $polyBeveloffset -segments 2
$brickName;
parent $brickName $groupnameF;
$brickZ = $brickZ+$brickZspace/2+0.135;
move -r $moveX $brickY $brickZ;
}
}
else
{
if($bZ==$brickZNum)
{
```

```
$brickName =
$buildName+"_brick_front"+($bY-1)*$brickZN
um+$bZ+1;
duplicate -name $brickName brickExampleZ;
parent $brickName $groupnameF;
$brickZ =
($bZ-1)*0.54+0.27+$brickZspace/2+0.27;
move -r $moveX $brickY $brickZ;
}
}
}
$brickZ = 0.27;
$brickY = ($bY)*0.3+0.15;
if($bY==$brickYNum)
{
$brickY =
($bY-1)*0.3+0.15+$brickYspace/2+0.15;
for($bZ=1;$bZ<=$brickZNum;$bZ++)
{
$brickName =
$buildName+"_brick_front"+($bY)*$brickZNu
m+$bZ;
duplicate -name $brickName brickExampleY;
parent $brickName $groupnameF;
$brickZ = ($bZ-1)*0.54+0.27;
move -r $moveX $brickY $brickZ;
if($bZ==$brickZNum)
{
$brickName =
$buildName+"_brick_front"+($bY)*$brickZNu
m+$bZ+1;
duplicate -name $brickName brickExampleYZ;
parent $brickName $groupnameF;
$brickZ =
($bZ-1)*0.54+0.27+$brickZspace/2+0.27;
move -r $moveX $brickY $brickZ;
}
}
}
}
move -r 0 0 $moveNZ $groupnameF;
$brickY = 0.15;
$brickZ = 0.27;
for($bY=1;$bY<=$brickYNum;$bY++)
{
for($bZ=1;$bZ<=$brickZNum;$bZ++)
{
$brickName =
$buildName+"_brick_back"+($bY-1)*$brickZN
um+$bZ;
duplicate -name $brickName brickExample;
parent $brickName $groupnameB;
$brickZ = ($bZ-1)*0.54+0.27;
move -r $moveNX $brickY $brickZ;
if($bY%2==0)
{
move -r 0 0 -0.27;
if($bZ==1)
{
setAttr ($brickName+".scaleZ") 0.5;
move -r 0 0 0.135 $brickName;
}
if($bZ==$brickZNum)
{
$brickName =
$buildName+"_brick_back"+($bY-1)*$brickZN
um+$bZ+1;
duplicate -name $brickName brickExampleZ;
parent $brickName $groupnameB;
$brickZ =
($bZ-1)*0.54+0.27+$brickZspace/2+0.27-0.27;
move -r $moveNX $brickY $brickZ;
$brickName =
$buildName+"_brick_back"+($bY-1)*$brickZN
um+$bZ+2;
polyCube -w 0.2 -h 0.3 -d 0.27 -name
$brickName;
polyBevel -offset $polyBeveloffset -segments 2
$brickName;
parent $brickName $groupnameB;
$brickZ = $brickZ+$brickZspace/2+0.135;
move -r $moveNX $brickY $brickZ;
}
}
else
{
if($bZ==$brickZNum)
{
```

```
$brickName =
$buildName+"_brick_back"+($bY-1)*$brickZN
um+$bZ+1;
duplicate -name $brickName brickExampleZ;
parent $brickName $groupnameB;
$brickZ =
($bZ-1)*0.54+0.27+$brickZspace/2+0.27;
move -r $moveNX $brickY $brickZ;
}
}
}
$brickZ = 0.27;
$brickY = ($bY)*0.3+0.15;
if($bY==$brickYNum)
{
$brickY =
($bY-1)*0.3+0.15+$brickYspace/2+0.15;
for($bZ=1;$bZ<=$brickZNum;$bZ++)
{
$brickName =
$buildName+"_brick_back"+($bY)*$brickZNu
m+$bZ;
duplicate -name $brickName brickExampleY;
parent $brickName $groupnameB;
$brickZ = ($bZ-1)*0.54+0.27;
move -r $moveNX $brickY $brickZ;
if($bZ==$brickZNum)
{
$brickName =
$buildName+"_brick_back"+($bY)*$brickZNu
m+$bZ+1;
duplicate -name $brickName brickExampleYZ;
parent $brickName $groupnameB;
$brickZ =
($bZ-1)*0.54+0.27+$brickZspace/2+0.27;
move -r $moveNX $brickY $brickZ;
}
}
}
}
move -r 0 0 $moveNZ $groupnameB;
$brickY = 0.15;
$brickX = 0.27;
for($bY=1;$bY<=$brickYNum;$bY++)
{
for($bX=1;$bX<=$brickXNum;$bX++)
{
$brickName =
$buildName+"_brick_left"+($bY-1)*$brickXNu
m+$bX;
duplicate -name $brickName brickExample1;
parent $brickName $groupnameL;
$brickX = ($bX-1)*0.54+0.27;
move -r $brickX $brickY $moveZ;
if($bY%2==0)
{
move -r -0.27 0 0 ;
if($bX==1)
{
setAttr ($brickName+".scaleX") 0.5;
move -r 0.135 0 0 $brickName;
}
if($bX==$brickXNum)
{
$brickName =
$buildName+"_brick_left"+($bY-1)*$brickXNu
m+$bX+1;
duplicate -name $brickName brickExample1X;
parent $brickName $groupnameL;
$brickX =
($bX-1)*0.54+0.27+$brickXspace/2+0.27-0.27;
move -r $brickX $brickY $moveZ;
$brickName =
$buildName+"_brick_left"+($bY-1)*$brickXNu
m+$bX+2;
polyCube -w 0.27 -h 0.3 -d 0.2 -name
$brickName;
polyBevel -offset $polyBeveloffset -segments 2
$brickName;
parent $brickName $groupnameL;
$brickX = $brickX+$brickXspace/2+0.135;
move -r $brickX $brickY $moveZ;
}
}
else
{
if($bX==$brickXNum)
{
```

```
$brickName =
$buildName+"_brick_left"+($bY-1)*$brickXNu
m+$bX+1;
duplicate -name $brickName brickExample1X;
parent $brickName $groupnameL;
$brickX =
($bX-1)*0.54+0.27+$brickXspace/2+0.27;
move -r $brickX $brickY $moveZ;
}
}
}
$brickX = 0.27;
$brickY = ($bY)*0.3+0.15;
if($bY==$brickYNum)
{
$brickY =
($bY-1)*0.3+0.15+$brickYspace/2+0.15;
for($bX=1;$bX<=$brickXNum;$bX++)
{
$brickName =
$buildName+"_brick_left"+($bY)*$brickXNum
+$bX;
duplicate -name $brickName brickExample1Y;
parent $brickName $groupnameL;
$brickX = ($bX-1)*0.54+0.27;
move -r $brickX $brickY $moveZ;
if($bX==$brickXNum)
{
$brickName =
$buildName+"_brick_left"+($bY)*$brickXNum
+$bX+1;
duplicate -name $brickName brickExample1XY;
parent $brickName $groupnameL;
$brickX =
($bX-1)*0.54+0.27+$brickXspace/2+0.27;
move -r $brickX $brickY $moveZ;
}
}
}
}
move -r $moveNX 0 0 $groupnameL;
$brickY = 0.15;
$brickX = 0.27;
for($bY=1;$bY<=$brickYNum;$bY++)
{
for($bX=1;$bX<=$brickXNum;$bX++)
{
$brickName =
$buildName+"_brick_right"+($bY-1)*$brickXN
um+$bX;
duplicate -name $brickName brickExample1;
parent $brickName $groupnameR;
$brickX = ($bX-1)*0.54+0.27;
move -r $brickX $brickY $moveNZ;
if($bY%2==0)
{
move -r -0.27 0 0 ;
if($bX==1)
{
setAttr ($brickName+".scaleX") 0.5;
move -r 0.135 0 0 $brickName;
}
if($bX==$brickXNum)
{
$brickName =
$buildName+"_brick_right"+($bY-1)*$brickXN
um+$bX+1;
duplicate -name $brickName brickExample1X;
parent $brickName $groupnameR;
$brickX =
($bX-1)*0.54+0.27+$brickXspace/2+0.27-0.27;
move -r $brickX $brickY $moveNZ;
$brickName =
$buildName+"_brick_right"+($bY-1)*$brickXN
um+$bX+2;
polyCube -w 0.27 -h 0.3 -d 0.2 -name
$brickName;
polyBevel -offset $polyBeveloffset -segments 2
$brickName;
parent $brickName $groupnameR;
$brickX = $brickX+$brickXspace/2+0.135;
move -r $brickX $brickY $moveNZ;
}
}
else
{
if($bX==$brickXNum)
{
```

```
$brickName =
$buildName+"_brick_right"+($bY-1)*$brickXN
um+$bX+1;
duplicate -name $brickName brickExample1X;
parent $brickName $groupnameR;
$brickX =
($bX-1)*0.54+0.27+$brickXspace/2+0.27;
move -r $brickX $brickY $moveNZ;
}
}
}
$brickX = 0.27;
$brickY = ($bY)*0.3+0.15;
if($bY==$brickYNum)
{
$brickY =
($bY-1)*0.3+0.15+$brickYspace/2+0.15;
for($bX=1;$bX<=$brickXNum;$bX++)
{
$brickName =
$buildName+"_brick_right"+($bY)*$brickXNu
m+$bX;
duplicate -name $brickName brickExample1Y;
parent $brickName $groupnameR;
$brickX = ($bX-1)*0.54+0.27;
move -r $brickX $brickY $moveNZ;
if($bX==$brickXNum)
{
$brickName =
$buildName+"_brick_right"+($bY)*$brickXNu
m+$bX+1;
duplicate -name $brickName brickExample1XY;
parent $brickName $groupnameR;
$brickX =
($bX-1)*0.54+0.27+$brickXspace/2+0.27;
move -r $brickX $brickY $moveZ;
}
}
}
}
move -r $moveNX 0 0 $groupnameR;
delete brickExample;
delete brickExampleZ;
delete brickExampleY;
delete brickExampleYZ;
delete brickExample1;
delete brickExample1X;
delete brickExample1Y;
delete brickExample1XY;
select -r $groupnameF;
sets -e -forceElement lambert2SG;
select -r $groupnameB;
sets -e -forceElement lambert2SG;
select -r $groupnameL;
sets -e -forceElement lambert2SG;
select -r $groupnameR;
sets -e -forceElement lambert2SG;
}
```

Voronoi.cpp

```
int CVoronoiView::Triangulate(int nver)
{
    bool Complete[maxtriangle];
    int Edges[2][ maxtriangle * 3];
    int Nedge;
    int ntri;
    int nedge;
    vertex edge[2][maxtriangle];
    int xmin,xmax,ymin,ymax,xmid,ymid;
    double dx,dy,dmax;

    int i,j,k;
    int temp;
    double xc=0.0,yc=0.0,r=0.0;
    bool inc=false;
    for(i=0;i<2;i++)
    {    for(j=0;j<maxtriangle * 3;j++)
            Edges[i][j]=-1;
    }

    xmin = Vertex[0].x;
    ymin = Vertex[0].y;
    xmax = xmin;
    ymax = ymin;
    for (i=1;i<nver;i++)
    {
        if (Vertex[i].x < xmin)    xmin =
```

```
Vertex[i].x;
            if (Vertex[i].x > xmax ) xmax =
Vertex[i].x;
            if (Vertex[i].y < ymin)    ymin =
Vertex[i].y;
            if (Vertex[i].y > ymax)    ymax =
Vertex[i].y;
      }

      dx = xmax - xmin;
      dy = ymax - ymin;
      if (dx > dy)
            dmax = dx;
      else
            dmax = dy;

      xmid = (xmax + xmin) / 2;
      ymid = (ymax + ymin) / 2;

      temp=nver;
      for(i=0;i<nver-1;i++)
      {
            for(j=i+1;j<nver;j++)
            {
                  if(Vertex[j].x<Vertex[i].x)
                  {

      Vertex[temp].x=Vertex[i].x;

      Vertex[temp].y=Vertex[i].y;
                        Vertex[i].x=Vertex[j].x;
                        Vertex[i].y=Vertex[j].y;

      Vertex[j].x=Vertex[temp].x;

      Vertex[j].y=Vertex[temp].y;
                  }

            }
      }

      Vertex[nver].x = xmid - 2*dmax;
      Vertex[nver].y = ymid - dmax;
      Vertex[nver + 1].x = xmid;
      Vertex[nver + 1].y = ymid + 2*dmax;
      Vertex[nver + 2].x = xmid + 2* dmax;
      Vertex[nver + 2].y = ymid - dmax;
      Triangle[0].vv0 = nver ;
      Triangle[0].vv1 = nver + 1;
      Triangle[0].vv2 = nver + 2;
      Complete[0] = false;
      ntri=1;
      for( i = 0;i<nver;i++)
      {
            Nedge = 0;
            for(j = 0;j<ntri;j++)
            {
                  if(!Complete[j])
                  {
                        inc=incircle(Vertex[i].x,
Vertex[i].y,
Vertex[Triangle[j].vv0].x,Vertex[Triangle[j].v
v0].y, Vertex[Triangle[j].vv1].x,
Vertex[Triangle[j].vv1].y,
Vertex[Triangle[j].vv2].x,
Vertex[Triangle[j].vv2].y, xc, yc, r);
                        if(xc+r<Vertex[i].x)
                              Complete[j]=true;
                        if (inc)
                        {
                              Edges[0][ Nedge + 0]
= Triangle[j].vv0;
                              Edges[1][ Nedge + 0]
= Triangle[j].vv1;
                              Edges[0][ Nedge + 1]
= Triangle[j].vv1;
                              Edges[1][ Nedge + 1]
= Triangle[j].vv2;
                              Edges[0][ Nedge + 2]
= Triangle[j].vv2;
                              Edges[1][ Nedge + 2]
= Triangle[j].vv0;
                              Nedge = Nedge + 3;
                              Triangle[j].vv0 =
Triangle[ntri-1].vv0;
                              Triangle[j].vv1 =
Triangle[ntri-1].vv1;
                              Triangle[j].vv2 =
```

```
Triangle[ntri-1].vv2;
                    Complete[j] =
Complete[ntri-1];
                    j = j - 1;
                    ntri = ntri - 1;
                }

            }
        }

        for (j = 0; j< Nedge-1;j ++)
        {
            for( k = j + 1;k< Nedge;k++)
            {

    if(Edges[0][j]==Edges[1][k]&&Edges[1][j]
==Edges[0][k])
                {
                    Edges[0][j] = -1;
                    Edges[1][j] = -1;
                    Edges[0][k] = -1;
                    Edges[1][k] = -1;
                }

    if(Edges[0][j]==Edges[0][k]&&Edges[1][j]
==Edges[1][k])
                {
                    Edges[0][j] = -1;
                    Edges[1][j] = -1;
                    Edges[0][k] = -1;
                    Edges[1][k] = -1;
                }
            }
        }


        for(j = 0;j< Nedge;j++)
        {
                if ( Edges[0][j] >= 0 &&
Edges[1][j] >= 0)
                {
                    Triangle[ntri].vv0 =
Edges[0][j];
                    Triangle[ntri].vv1 =
Edges[1][j];
                    Triangle[ntri].vv2 =
i;

                    Complete[ntri] =
false;
                    ntri = ntri + 1;
                }
        }
    }
        for(j=0;j<ntri;j++)
        {

    center[j]=circlecenter(Vertex[Triangle[j]
.vv0].x,Vertex[Triangle[j].vv0].y,
Vertex[Triangle[j].vv1].x,
Vertex[Triangle[j].vv1].y,
Vertex[Triangle[j].vv2].x,
Vertex[Triangle[j].vv2].y);
        }
        nedge=0;
        for(i=0;i<ntri-1;i++)
        {
            for(j=i+1;j<ntri;j++)
            {
                if(Triangle[i].vv0 >=nver||
Triangle[i].vv1>= nver|| Triangle[i].vv2 >=
nver)
                {

    if(Triangle[j].vv0 >=nver||
Triangle[j].vv1>= nver|| Triangle[j].vv2 >=
nver)
                        continue;

    if(Triangle[i].vv0==Triangle[j].vv0||Triangl
e[i].vv0==Triangle[j].vv1||Triangle[i].vv0==Tria
ngle[j].vv2)
                    {

    if(Triangle[i].vv1==Triangle[j].vv0||Triangl
```

```
e[i].vv1==Triangle[j].vv1||Triangle[i].vv1==Triangle[j].vv2||Triangle[i].vv2==Triangle[j].vv0||Triangle[i].vv2==Triangle[j].vv1||Triangle[i].vv2==Triangle[j].vv2)
                    {
    edge[0][nedge]=center[i];
    edge[1][nedge]=center[j];
                        nedge++;
                    }
                }

if(Triangle[i].vv1==Triangle[j].vv0||Triangle[i].vv1==Triangle[j].vv1||Triangle[i].vv1==Triangle[j].vv2)
                {
    if(Triangle[i].vv0==Triangle[j].vv0||Triangle[i].vv0==Triangle[j].vv1||Triangle[i].vv0==Triangle[j].vv2||Triangle[i].vv2==Triangle[j].vv0||Triangle[i].vv2==Triangle[j].vv1||Triangle[i].vv2==Triangle[j].vv2)
                    {
    edge[0][nedge]=center[i];
    edge[1][nedge]=center[j];
                        nedge++;
                    }
                }

            }
            else
            {
    if(Triangle[i].vv0==Triangle[j].vv0||Triangle[i].vv0==Triangle[j].vv1||Triangle[i].vv0==Triangle[j].vv2)
                {
    if(Triangle[i].vv1==Triangle[j].vv0||Triangle[i].vv1==Triangle[j].vv1||Triangle[i].vv1==Triangle[j].vv2||Triangle[i].vv2==Triangle[j].vv0||Triangle[i].vv2==Triangle[j].vv1||Triangle[i].vv2==Triangle[j].vv2)
                    {
    edge[0][nedge]=center[i];
    edge[1][nedge]=center[j];
                        nedge++;
                    }
                }
            }
        }
        this->nedge=nedge;
        for(i=0;i<2;i++)
        {
            for(j=0;j<nedge;j++)
                this->edge[i][j]=edge[i][j];
        }
    return ntri;
}
```