スクリプト言語による建築・都市デザインの記述について

― ゴルジ構造体を事例として ―

○水谷　晃啓[*1]　　菊池　誠[*3]
八束はじめ[*2]　　井出　翼[*4]

キーワード：スクリプト言語　デザイン・プロセス　アルゴリズム　アルゴリズミック・デザイン

１．はじめに

建築や都市のデザインにコンピュータを導入するための研究は以前から行われてきているが、CAD や BIM が補足的に導入されているのが現状で、本質的なデザインのツールとして用いられているとはいえない。本論は近年着目されるスクリプト言語により、建築・都市デザインを記述することで、より本質的なコンピュータの利用を試みた。スクリプトは、アプリケーションの動作内容を制御することが可能なので、描画などの機能を構築しなくとも、アイデアや操作のみをアルゴリズムとして記述すればよいという利点がある。デザイナーにとって必ずしも容易ではなかったプログラミングは、より身近なものとして扱うことが可能となった。すでにハーバード大学や慶応大学などの設計スタジオで取り組まれた、建築や都市のデザインにスクリプトを用いた事例などがあるが、形態生成に主眼が置かれているものが中心で、アイデアやデザインプロセスの記述にまで達した事例は少ない。本研究はスクリプトが建築形態の記述に留まらず、そのアルゴリズムから建築・都市デザインのより本質的な意義を情報として記述できることを槇文彦の「ゴルジ構造体」を事例に示していく。

２．ゴルジ構造体について

「ゴルジ構造体」は 1967 年に槇文彦によって発表された超高密度都市の空間モデルで、この都市モデルは 2 バージョンある。ゴルジ体は槇の Group Form という概念に位置づけられるため、2 つとも同じコンセプトとシステムからできているが、採光や通風といった環境とコミュニケーションの場として用意されるヴォイド空間の形態が異なっている(図 1)。槇のスケッチからこのデザインを起こした遠藤精一は、初めに考え出されたのが円筒型バージョンで、その数週間後につくられたのが円錐型バージョンであったと筆者のインタビューで答えている。

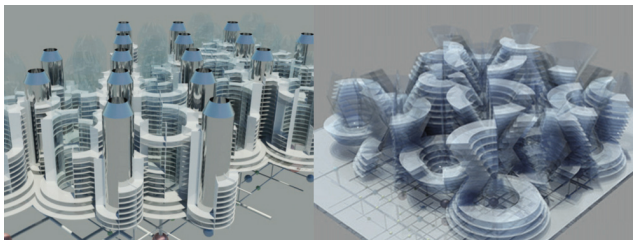


図 1.　　円筒型バージョン　　円錐型バージョン

槇は Group Form の説明において空間の密度について触れているが、高密度状態での空間のありかたについて、同時期に完成した師・丹下の山梨文化会館を事例に引いている。槇は空間の使用密度が高まれば高まるほど、内部空間と外部空間の相関関係が密になると指摘し、高密度の集合体では内部空間と外部空間の対立概念が消失するとしている。このとき、外部空間や内部のヴォイド空間といった使用空間の媒体となる空間の重要性を述べ、山梨文化会館を「各種の使用空間が、媒体空間を立体的にもったかたちであらわされたもの」(注 1) としている。この考えを示すように、最初につくられたバージョンは山梨文化会館のコアと同じく円筒型のヴォイドを媒体空間とし、その周囲に使用空間としてスラブが集積されている。山梨文化会館はコミュニケーション・シャフトによる 3 次元空間のネットワーク化が主題とされたが、この概念はゴルジ体のコミュニケーション塔および光塔を用いた媒体空間と使用空間の考え方に類似している。槇はゴルジ体の以前にも堂島計画において新時代における建物内の機能、人や物やエネルギーの流動性の必要性を指摘していたが、山梨文化会館を見てよりその考えを強くしたのかもしれない。

しかしながら、遠藤は「丹下研のコアを意識していなかった」と述べているため、ここでは概念と形態の類似性を指摘するに止まざるを得ないが、仮にスケッチを描いた槇が丹下研のコアを意識的に捉えていたとすると、2 つ目の円錐バージョンに槇のオリジナリティを発見できる。遠藤によると 3 次元的なネットワーク化のために、より隣同士のつながりが強くなる形態を模索した結果、円筒型の上下を横に広げた円錐型バージョンが生まれたとのことである。どちらのバージョンもコミュニケーション塔内部には交通インフラが通されていたが、円筒型が垂直交通＋水平交通で交通ネットワークが構成されたのに対し、円錐型は斜行交通のみでそれが構成できた。67 年の発表から 40 年経って再度製作された模型が、円錐型バージョンであったのは円筒型にはない可能性の発見があったからかもしれない。遠藤が丹下研のコアを意識していなかったが故の展開に、槇が丹下研にはない可能性とオリジナリティを、円錐という形態から感じ取っていたのではないか。このような考えからすると、コアとジョイントによって縦と横が立体的にネットワーク化された山梨文化会館に対し、ゴルジ

体は 3 次元的なネットワーク化を、円錐コアのみによって可能としたと結論付けることができるだろう。

3．ゴルジ構造体のデザインプロセスとそのシステム

ここで論じるゴルジ体のデザインプロセスとシステムは「メタボリズムと未来都市展」（2011 年：森美術館）の準備のために、筆者が槇と遠藤へのインタビューから行った、ゴルジ体の CG 復元作業に基づいている。ゴルジ構造体は交通やエネルギーなどの系として考えられた、オープンエンドシステムと呼ばれる自由に伸縮する開放系のシステムと接続されている。この開放系のシステムを決定づけるのは Critical joint と呼ばれるそれぞれの系をつなぐための結節点である。コミュニケーション塔の下には交通の拠点となる Critical joint があり、ゴルジ体内部の垂直交通と外部の水平交通が連結されている。光塔とコミュニケーション塔が互い違いに配置され、この２つが媒体空間となるヴォイドとして、あらかじめ恒久的なものとして用意される。それらの隙間となる空間に、スラブが暫定的に増殖していくというシステムを持つ。

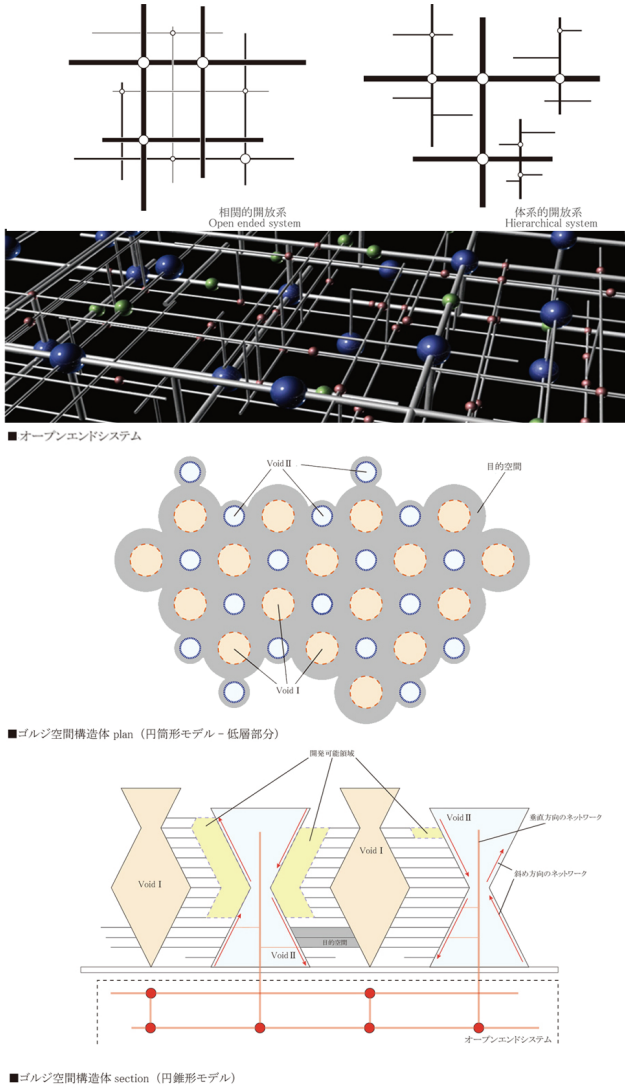


図 2.オープンエンドシステムとゴルジ構造体の説明図

つまり、ゴルジ体はスラブの積層物であるビルとビルの間が、結果的に都市的なヴォイド空間となる、通常の空間形成のプロセスを反転させることで、超高密度状態においても環境や空間構成の秩序が保たれるように考えられている。以下にデザインプロセスと空間構成のシステムをまとめ、次節においてこれをスクリプト化していく。

STEP 1：Critical joint の生成

各系の結節点となる主要な Critical joint が配置され、補足的な Critical joint がランダムに配置される。

STEP 2：Open ended system の生成

自由に伸縮する各系によって Critical joint がネットワーク化され、開放系のシステムが出来上がる。

STEP 3：光塔およびコミュニケーション塔の生成

光塔とコミュニケーション塔が配置され、ゴルジ体内部の垂直交通と外部の水平交通が接続される。

STEP 4：スラブの生成

先んじて用意されたヴォイド空間の隙間を埋めるように増殖する。

4．スクリプトによる記述

「ゴルジ構造体」のデザインとそのプロセスのスクリプティングには CAD との相性が良く、建築業界で広く導入されているオートデスク社・3ds Max のスクリプト言語（maxscript）を使用した。ステップごとの各操作を関数として定義し、ゴルジ体のデザインプロセス全体がスクリプトによってコード化されている。ここではデザインプロセスとシステムに着目するために円筒型を扱っている。

STEP 1

```
--関数 1:ゴルジ体の内部交通と接続点となる主要な Critical joint の配置を行う関数
fn critical_joint_1 size posz obj = (
    for i = 0 to 3 do(
        posy = i*size
        for ii = 0 to 4 do(
            t = sphere radius:10000 wirecolor:blue
            t.name="joint_1_"+(t.pos.z as string)+"_"+(i as string)+(ii as string)
            t.pos = [ii*size, posy , posz]
            append obj t
        )
    )
   return obj
)
--補足的な Critical joint の配置を与える為の値（寸法）の配列
major = #(0 , 200000 , 400000, 600000)
minor = #(50000 , 100000, 150000)
--関数２：与えられる寸法から補足的な Critical joint の配置をランダムに行う関数
fn critical_joint_2   num posz obj=(
    for i = 1 to num do (
        t = sphere radius:10000 wirecolor:red
        t.pos = [ (major[(random 1 4)]+minor[(random 1 3)]) , (major[(random 1 3)]+minor[(random 1 3)]) ,posz ]
        t.name = "joint_2_" + (i as string)+"_"+(t.pos.z as string)
        append obj t
```

```
    )
    return major
)
--関数３：補足的な Critical joint の重なりを解消するための関数
fn lap posz obj =(
    for i in obj do(
        for ii in obj do if (i != ii) and (i.pos==ii.pos) then(
            done = false
            i.pos = [ (major[(random 1 3)]+minor[(random 1 3)]) ,
(major[(random 1 3)]+minor[(random 1 3)]) ,posz ]
        )
    )
    return major
)

postion1_0 = #()    -- 1 層目に配置される主要な Critical joint を格納するための配列
critical_joint_1 200000 0 postion1_0    --関数１より１層目に配置される主要な Critical joint の生成

postion1_1 = #()    -- 2 層目に配置される主要な Critical joint を格納するための配列
critical_joint_1 200000 100000 postion1_1   --関数１より２層目に配置される主要な Critical joint の生成

postion2_0 = #()    -- 1 層目に配置される補足的な Critical joint を格納するための配列
critical_joint_2 10 0 postion2_0    --関数２より１層目に配置される補足的な Critical joint の生成

-- 1 層目に配置される補足的な Critical joint に重なりがある場合は関数３を用いて再配置を行う
done = false
do (
    done = true
    lap 0 postion2_0
)while not done
postion2_1 = #()    --関数２より２層目に配置される補足的な Critical joint を格納するための配列
critical_joint_2 15 100000 postion2_1   --関数２より２層目に配置される補足的な Critical joint の生成

-- 2 層目に配置される補足的な Critical joint に重なりがある場合は関数３を用いてに再配置を行う
done = false
do (
    done = true
    lap 100000 postion2_1
)while not done
```

STEP 2

```
--関数４：主要な Critical joint をネットワーク化する主要な系を派生させる関数
fn openend1 posz= (
    for i = 0 to 4 do(
        posx=i*200000
        r1 =cylinder height:640000 radius:4000 transform:(matrix3
[1,0,0] [0,0,1] [0,1,0] [posx,-20000,posz]) wirecolor:gray
        r1.name = "path_a_"+(i as string) + (r1.pos.z as string)
    )
        for i = 0 to 3 do(
        posy=i*200000
        r2 =cylinder height:840000 radius:4000 transform:(matrix3
[0,-1,0] [0,0,1] [1,0,0] [-20000,posy,posz])wirecolor:gray
        r2.name = "path_a_"+(i as string) + (r2.pos.z as string)
    )
)
--関数５：補足的な Critical joint に対して系を派生させる関数
fn openend2 obj = (
    for i in obj do
    (
        r1  =cylinder  height:major[(random  2  3)]  radius:2000
transform:(matrix3  [1,0,0]  [0,0,1]  [0,1,0]
[i.pos.x,i.pos.y-minor[(random 1 2)] ,i.pos.z]) wirecolor:white
        r2  =cylinder  height:major[(random  2  3)]  radius:2000
transform:(matrix3 [0,-1,0] [0,0,1] [1,0,0] [i.pos.x-minor[(random 1
2)],i.pos.y,i.pos.z])wirecolor:white
    )
)
--関数６：主要な Critical joint に対してゴルジ体と接続するための縦の系を派生させる関数
fn vertical obj = (
    for i in obj do
    (
        v = cylinder height:150000 radius:4000 wirecolor:gray
        v.pos= i.pos
    )
)
openend1 0 --関数４より１層目に配置された Critical joint をつなぐ主要な系の生成
openend1  100000 --関数４より２層目に配置された  Critical  joint をつなぐ主要な系の生成

joint_2 = join postion2_0 postion2_1  --すべての補足的な Critical joint を格納
openend2 joint_2 --関数５より補足的な Critical joint に対し系を生成
vertical postion1_0 --関数６より主要な Critical joint に対しゴルジ体と接続するための縦の系を生成
```

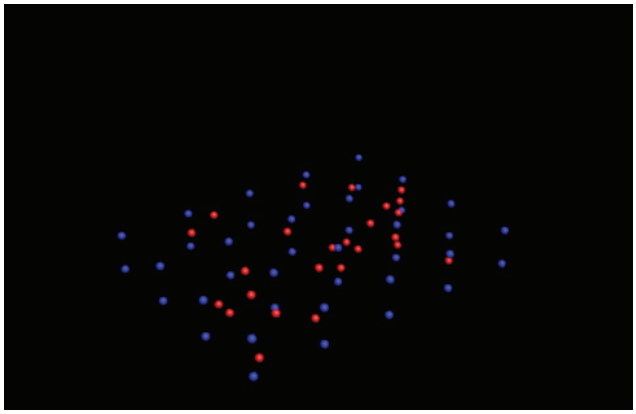


図 3. STEP 1 のスクリプトを実行して得られた結果

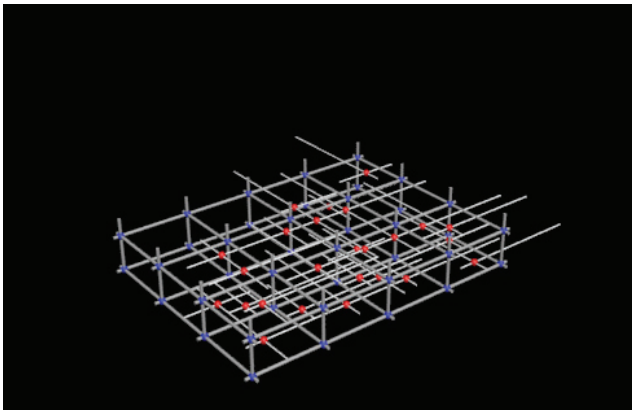


図 4. STEP 2 のスクリプトを実行して得られた結果

STEP 3

```
--関数７：コミュニケーション塔を作成するための関数
fn communication heigh obj =(
    for i in obj do(
        comm = cylinder height:heigh radius:25000 wirecolor:white
        comm.pos= [i.pos.x,i.pos.y,i.pos.z+30000]
        convertToPoly(comm)
        comm.EditablePoly.SetSelection #Face #{25..26}
        comm.EditablePoly.delete #Face
        meditMaterials[1].opacity = 50
        comm.material = meditMaterials[1]
        meditMaterials[1].twoSided = on
    )
)
--関数８：光塔を作成するための関数
fn voids size posz obj height =(
    for i = 0 to 2 do(
        posy = i*size+size/2
        for ii = 0 to 3 do
        (
            t = point()
            t.pos = [ii*size+size/2, posy , posz]
            append obj t
        )
    )

    for i in obj do(
        void = cylinder height:height radius:40000 wirecolor:gray
        void.pos= i.pos
        convertToPoly(void)
        void.EditablePoly.SetSelection #Face #{25..26}
        void.EditablePoly.delete #Face
        void.material = meditMaterials[1]
        cap = Cone height:30000 radius1:40000 radius2:25000
wirecolor:gray
        cap.pos = [i.pos.x,i.pos.y,i.pos.z+height]
        convertToPoly(cap)
        cap.EditablePoly.SetSelection #Face #{121..122}
        cap.EditablePoly.delete #Face
        cap.material = meditMaterials[1]
    )
)

communication 300000 postion1_1 --関数７を用いて高さ 300000
のコミュニケーション塔を Critical joint 上に配置

void_pos=#() --光塔の配置位置を格納するための配列
voids 200000 130000 void_pos 200000 --関数８を用いて高さ
200000 の光塔をコミュニケーション塔の中央に配置
```

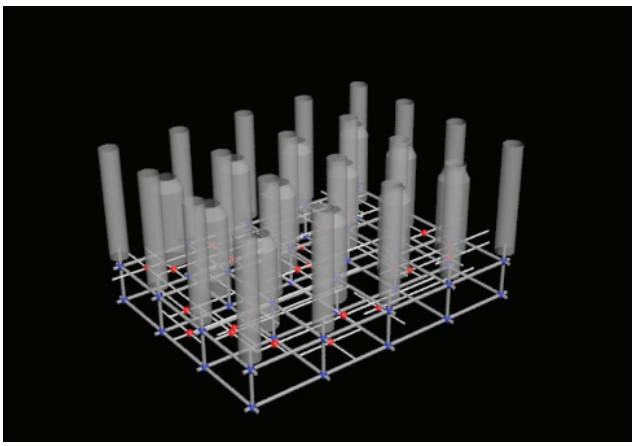


図 5. STEP 3 のスクリプトを実行して得られた結果

STEP 4

```
--関数９：コミュニケーション塔に対してスラブを作成するため
の関数
fn sulab1 num obj=(
    for i = 1 to num do(
        for ii in obj do(
            r1= 25000
                su = tube radius1:r1 radius2:(r1+25000)
height:300 pos:[ii.pos.x , ii.pos.y, 130000+6000*(random 1
30)] wirecolor:red
            meditMaterials[2].opacity = 50
            meditMaterials[2].Diffuse = color 252 255 0
            su.material = meditMaterials[2]
        )
    )
)
--関数１０：光塔に対してスラブを作成するための関数
fn sulab2 num obj=(m
    for i = 1 to num do(
        for ii in obj do(
            r2= 40000
                su = tube radius1:r2 radius2:(r2+60000)
height:300 pos:[ii.pos.x , ii.pos.y, 130000+6000*(random 1 20)]
wirecolor:red
            su.material = meditMaterials[2]
        )
    )
)
--地盤を作成する
gl = box length:700000 width:900000 heigth:200 wirecolor:white
gl.material = meditMaterials[1]
gl.pos=[gl.width/2-50000,gl.length/2-50000,125000 ]

sulab1 20 postion1_1 --関数９を用いてコミュニケーション塔に
スラブをランダムに配置（1 本あたりスラブは 20 枚）
sulab2 20 void_pos --関数１０を用いて光塔にスラブをランダム
に配置（1 本あたりスラブは 20 枚）
```

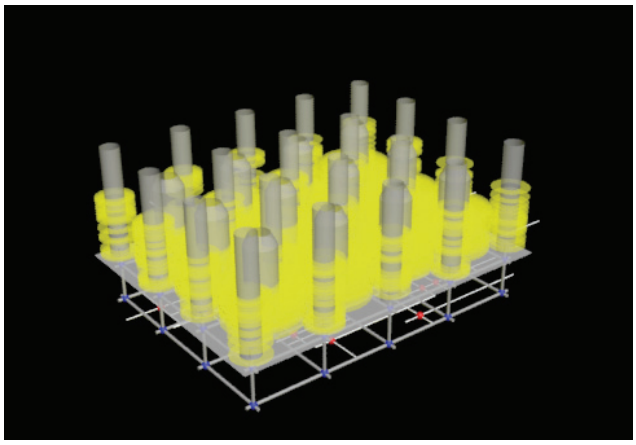
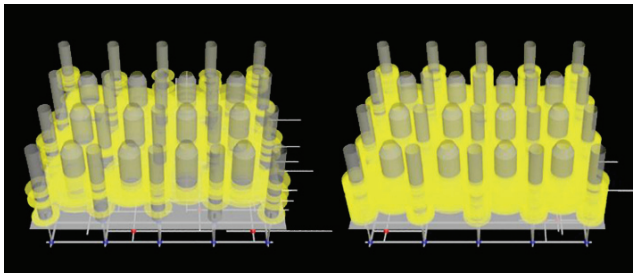


図 6. STEP 4 のスクリプトを実行して得られた結果



1 本に 5 枚のスラブ　　1 本に 30 枚のスラブ

図 7. 塔 1 本あたりのスラブ数を変化させて実行した例

５．シミュレーション結果の検証と考察

2 節と 3 節でゴルジ体の緒言とデザインについて述べ、4 節でそれらをスクリプト化したが、本節ではシミュレーション結果の検証からスクリプトの記述内容を考察する。まず STEP1 で Critical joint がスクリプトにより、どのように配置されたか検証を行う。STEP1 では関数 1 によって主要な Critical joint（以下 C1）が 4 行 5 列の 200M グリットの交点に配置され、その格子内を 4 等分する 50M グリットの交点に補足的な Critical joint（以下 C2）が関数 2 によって配置される。この操作は 1 層ごとに行われるため、Z=0 , 10000 の各層で 2 回繰り返される。C1 は 200M グリットの交点全てに配置されるので、実行の度に同じ位置に配置された。それに対し C2 は、108 個の交点のうち 10 個の交点がランダムに選択されるため、毎回異なる位置に配置された。

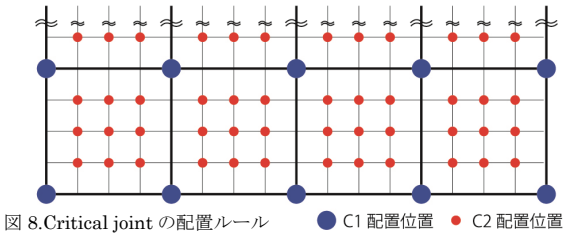


図 8.Critical joint の配置ルール

次に STEP2 において各系がスクリプトによってどのように生成されているか検証を行う。STEP2 では関数 4 によって C1 同士を接続する系が生成され、関数 5 から C2 から派生する系が生成されたのち、関数 6 によって上下の段を縦につなぐ系が生成される。先に示したように C1 は定位置に配置されるので、関数 5 によって生成される系も毎回同じパターンで生成された。一方、関数 6 から生成される C2 から派生する系は、各系の長さは 200M と 400M からランダムに選択され、かつ C2 は毎回異なる配置パターンとなるため、毎回異なる生成パターンとなった。

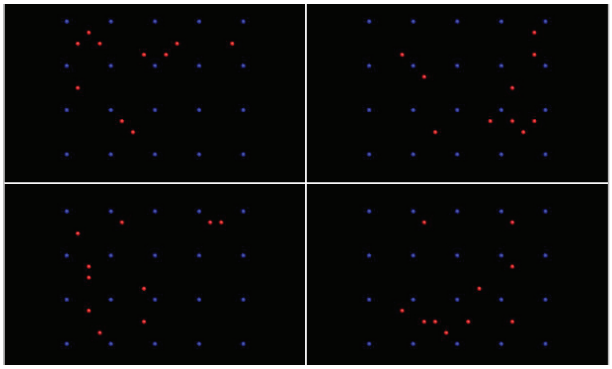


図 9.スクリプトによって得た 4 パターンの配置位置

次いで STEP3 において光塔・コミュニケーション塔がスクリプトによってどのように配置されたか検証を行う。STEP3 では関数 7 によってコミュニケーション塔が C1 上に配置され、関数 8 によって 200M グリットの中心に光塔が配置される。パラメータとして関数 7 には高さ 300M が与えられ、関数 8 には高さ 200M が与えられているので、コミュニケーション塔の高さは 300M、光塔の高さは 200M となっている（図 4）。最後に STEP4 においてスラブがスクリプトによってどのように配置されているか検証を行う。STEP4 ではコミュニケーション塔は関数 9 によって、光塔は関数 10 によってスラブがそれぞれランダムに生成される。関数 9・10 にはパラメータとして 20 を与えたので、各塔には 20 枚のスラブが生成された。このパラメータの値を 5・30 と変化させると、各塔に生成されるスラブの枚数は 5 枚・30 枚となる（図 6）。シミュレーション結果が示すように STEP1・2 では、オープンエンドシステムの「自由に伸縮する開放系のシステム」というコンセプトをスクリプトによって記述することができた。また STEP3・4 では「媒体空間として用意される恒久的なヴォイドの隙間にスラブが暫定的に増殖する」というゴルジ体のコンセプトとシステムを、スクリプトによって記述することができた。

６．まとめと今後の展開

ゴルジ構造体のデザインとそのプロセスをスクリプト言語により記述したが、これまで図や文章で示されていた設計意図と、図面や模型で示されていた情報を、そこに書き込むことができた。実行可能な環境が整っていれば、誰でも同じ操作と結果を得ることができる。このコードに書かれるアルゴリズムを通して、寸法や形態・アイデアがどのようなプロセスでデザインしたのか伝えることができるため、コミュニケーションのツールとしても活用できることが分かった。これらの可能性を示せたことで、スクリプトを建築・都市デザインの記述に用いる意義と、有効性を論じることができたと考える。また、コードのパラメータを変化させることで、同じアルゴリズムから複数のバリアントを生み出すことが可能なので、スタディの手法としても有効であるといえる。設計条件の変更などがあった場合、パラメータだけでなくアルゴリズムを部分的に書き換えたり、コードを書き加えたりすることで常にバージョンアップさせていくこともできるため、より一般的なデザインツールへと発展していくことが期待できるといえる。

※この研究は大林財団の支援を受けて行われている。

[参考文献]

1. 槇文彦, 遠藤精一：建築文化, Golgi 構造体, 1967. 6
2. 森美術館：メタボリズムの未来都市, pp. 88~89, 新建築社, 2011
3. 長倉威彦：建築家の文法, 建築雑誌, pp. 30~31, 1992. 05
4. 渡辺仁史：スクリプトによる形態生成, 未来を拓く新しい建築システム, pp. 53~56, 建築技術, 2005
5. コスタス・テルジディス, 田中浩也（訳）：アルゴリズミックアーキテクチュア, 彰国社, 2010

[注]

注 1）建築文化 1967. 06「環境デザインへの二つのアプローチ」

＊1　芝浦工業大学大学院理工学研究科　博士課程後期
＊2　芝浦工業大学工学部建築工学科　教授
＊3　芝浦工業大学システム理工学部環境システム学科　教授
＊4　芝浦工業大学大学院理工学研究科　博士課程前期

Describing Architectural and Urban Designs Using Script Language

- Golgi Structure as an example -

○Akihiro MIZUTANI*1 Makoto KIKUCHI*3
Hajime YATSUKA*2 Tsubasa IDE*4

Keywords : Scripting language, Design process, Algorithm, Algorithmic design

Introduction

The research to introduce computer for architectural and urban design has been conducted for some time. In the present paper, a trial is conducted to introduce computer in a more fundamental manner by describing architectural and urban designs using a script language, which is gaining attention in the recent years. Unlike regular programs, script language controls the operations of the application and has its advantage in that it is not necessary to construct a base program. It is possible to execute a code by describing ideas and operations, without constructing functions such as drawing. In the past, the programming was not necessarily an easy task for designers, but now it can be dealt with in much friendlier manner. The examples of using script language for architectural and urban designs conducted at design studios of Harvard and Keio University, have been published thus far. However, such examples are focused toward shape generation and little examples exist where the description of ideas and design process is discussed. In the present research, "Golgi Structure" by Fumihiko Maki is used as an example to prove that the use of script language is not only limited to describing architectural forms, but can be used also to describe fundamental significance of architectural and urban designs.

Design Description Methodologies Using Script Language

The following steps are followed to describe Golgi Structure using script language

STEP 1: Generation of critical joints

Major critical joints which are the nodes for each system are laid out, and some additional critical joints are laid out randomly.

STEP 2: Generation of open ended system

The critical joints form a network by the systems that have freedom to expand and shrink, thereby generating an open system.

STEP 3: Generation of the Towers of Light and the Communication Towers

The Towers of Light and Communication Towers are laid out and the external, horizontal traffic is connected to the vertical traffic inside the Golgi structure.

STEP 4: Generation of slabs

Slabs reproduce themselves as if to fill the gaps of the previously prepared void space.

Each operation at each step will be defined as a function. The design and the process of the Golgi Structure are described using a script language.

Summary and Future Development

The design intentions previously presented through figures and words, as well as information previously presented by drawings and models, were successfully written into script language describing the Golgi Structure. The code that consists of approximately 200 lines ensures that when executable environment is prepared, anybody could conduct the same operation and could obtain the same result. It was also found out that this could also be used as a communication tool, since the design process that took place to determine dimensions, shapes and ideas can be communicated through the algorithm described in this code. By presenting these potentials, it is believed that the significance and validity of using script language to describe architectural and city design is proven. It is also believed that this code is valid as a design study tool, as by altering the parameters in the code, several variations can be generated using the same algorithm. When there are changes to the design conditions etc., consistent version upgrade is possible by altering not only the parameters but also part of the algorithm, or by adding codes. Thus, it is expected that this code could evolve into more generic design tool.

＊1 Graduate Student, Graduate School of SIT
＊2 Professor, Architecture Building Engineering, SIT.
＊3 Professor, Architecture and Environment Systems, SIT
＊4 Graduate Student, Graduate School of SIT