Open Design Environment (ODE): Global Design Studio, Experiments in 3D City Simulation

Claude Comair
Atsuko Kaga

Osaka University, Faculty of Engineering,
Environmental Engineering (Sasada Laboratory)
2-1 YamadaOka, Suita, Osaka 565J
JAPAN

This paper depicts the evolution of the research done at the Sasada Laboratory (Osaka University) in the fields of Architectural and Urban related Computer Simulations. This research led to the birth of what we call the "Open Development Environment" (ODE). ODE is presented in this paper through a simple example. In this example, four teams cooperate to produce the database for a simple twin tower complex. The database is kept very simple and the protocol of communication among the different teams is a new computer language called VU Wee-You). *VU was conceived and developed by Claude Comair for the specific purpose of defining architectural and urban objects.*

Keywords: computer assisted design, computer languages, computer generated databases, computer graphics, three-dimensional computer simulation.

1 History/introduction

This paper depicts the evolution of the research done at the Sasada Laboratory over a period of ten years. This research led to the birth of what we call the 'Open Development Environment" (ODE). ODE is a collection of software, hardware and knowhow used to generate computer simulations in the fields of architecture and urban planning. More importantly, ODE is used as a method of communication among the different teams working on a given project. As shown by the list of projects below, some of these projects required a large amount of human resources in order to be completed.

- Osaka City (Feb. 1983)
- Portions of the subway system in the city of Osaka (1983).
- Kyoto (Aug. 1983)
- Shinjuku, Tokyo (Oct. 1983)
- Portopia, Kobe (Mar. 1984)
- Design work of Media City, Osaka (1984)
- Renewal project in Sanda City (1984)
- Various animations for the Japanese pavilion at Tsukuba Expo 1985
- Various animations for Parigraph '85
- ABC complex, Osaka (May 1985)
- Sewer System in Osaka (Feb. 1986)
- Kobe new transportation system (Mar. 1986)
- Shanghai City (Oct. 1986)
- New Kansai International Airport (1987, 1993)

• Parthenon (1989)
• Senri International Business Complex (Sept. 1990)
• Japanese Railway (JR.) (1991, 1993)
• Tae Jon Expo '93 Korea (Sept. 1991)

Table 1. List of some of the animation projects produced by the Sasada Laboratory

The Sasada Laboratory is located at the University of Osaka, Faculty of Engineering, Department of Environmental Engineering, Japan. The portfolio of the Laboratory is extensive and impressive. The projects which were produced by the men and women of the Laboratory range from the production of databases and computer simulation of several segments of different cities throughout the world to specific studies of architectural monuments. The work performed on the databases was varied and included simulation of past, present, and future events. The simulation of ancient events included such projects as the computer reconstruction of the city of Lou-Lan in ancient China. LouLan was one of the famous resting points for the caravans on the famous Silk Road. In the category of simulation of present monuments, it can be shown how certain structures integrate with the existing environment, and how they are affected by natural and artificial elements. In this category, we can mention several projects: the animation of the Arch of Triumph and Notre Dame in Paris, the famous temples of Kyoto, the downtown core of Osaka City, as well as the downtown core of the Shinjuku District in the City of Tokyo, Japan. The final category, the simulation of future events, is where most of our work was done. Perhaps the largest project we have undertaken in this category was the inclusion of several renewal projects into the current urban fabric of the City of Shanghai. This was a h e project that involved hundreds of people with different backgrounds in both China au~ Japan. The entire database of the City of Shanghai and the databases of the new n portions had to be entered into the computer. Once this was achieved, a series of animations was performed onto the database in order to check the impact of the newly added elements on the existing environment. The feedback from the simulations resulted in the alteration of the proposed architecture and some of the designated locations, and a final development renewal director-plan was achieved.

These databases were often huge and very complex to build. They presented challenges that sometimes seemed impossible to overcome. Often specialized software, and in some cases hardware, had to be designed on the "fly" for the task. In this paper, we describe the advances of our research and how our work led us to the development of hardware and software. Most importantly, it depicts the methodology of work which our lab undertook.

2 Open Development Environment (ODE)

ODE, is a collection of software, hardware, and most importantly, a methodology of work. The main purpose of ODE is to allow various people, usually separated by great distances, to work together on a given project and to introduce computer simulation into the working environment. Today, our laboratory is no longer limited to the physical location of our lab. Thanks to global area networks, such as the Internet, our office has been extended into the virtual space of the web. Today, we exchange ideas and collaborate on projects using the network. Lately, we invited a group of about fifty people to collaborate on the design of a 'center house" located in Sumoto Ohama beach on Awaji Island, Hyogo Prefecture, Japan. The total design period extended over a period of one and half months. The work was achieved promptly, and the results were very satisfatory.

Through the following simple example, this paper will present ODE. In this example, four teams cooperate to produce the database for a simple twin tower complex. Of course, the database is kept very simple since the aim of this paper is to present a way of working, and not to teach computer coding.

When people work together, they need a protocol that guarantees proper communication among the members of the team. The proposed protocol is a threedimensional simulation computer language called VU. VU was conceived and developed by Claude Comair at Sasada Lab, Osaka University, Faculty of Engineering, Department of Environmental Engineering, in 1984. VU's main purpose is to define architectural and

urban objects. Once defined, VU objects can be submitted to a variety of mathematical transformations in order to achieve various simulations.

The example presented in this paper was written using the VU language. The language is simple and allows a free format database to be built. Although this paper uses the VU language, it is not a manual for the language. Minimal information about the language is presented here, just enough to allow the reader understand the example. If the reader is interested in knowing more about VU, the bibliography gives two references on the language. Further information on the VU language can be obtained directly from the author.

3 Brief presentation of the VU language

VU is a modem computer language. It supports most of the features found in general programming languages. VU was conceived and developed for the purpose of defining architectural and urban objects. Once defined, VU objects can be submitted to a variety of mathematical transformations in order to achieve various simulations.

3.1 Free format

A VU database is viewed by the VU interpreter as a stream of ASCII characters. VU accepts a free format input. This means that successive separators are ignored by VU. In fact, VU programs could be written on a single line.

3.2 VU comments

VU accepts two different ways to make comments about the code. VU considers a comment any sequence of characters enclosed between the opening construct "/" and the closing construct "*/"The second form of a comment starts with the opening construct "//"

and ends with a new-line character. Comments do not affect the program and are ignored by VU.

```
/* this is a comment */
// this is an "until the end of the line comment"
```

3.3 VU variables

VU allows the user to declare various types of variables. This project declares some of the following types:

• Variable Values
• Variable Points
• Variable Matrices
• Variable Strings
• Variable Cameras
• Variable Binary Data Blocks
• Variable Text Data Blocks

Table 2. VU Variables

Value variables are either of type integer or type double. Their names can be of any length and must start with a character followed by any sequence of characters and digits. Point variables represent locations in 3D space. Names of point variables follow the same naming rules as for value variables, but they must start with a "." as in ".origin". The following is a section of code that declares the value variable "age" and a point variable .0.

```
declare age = 20;
declare .o = (10,10,10);
// this could have been written:
declare {
```

```
        age = 20;
        .o = (10,10,10);
}
```

3.4 Flow control

VU supports a wide array of flow control statements. VU supports the "for", "while", and "do..while" loop statements, as well as the "if", "if ... else", and "case" conditional statements. VU also supports other forms of jump statements, mainly object calling. Flow Control statements are generally included in a computer program to alter its linear execution. By including flow control statements in the definition of objects in the database, the programmer gives the objects some kind of artificial intelligence or behavior.

3.5 VU objects

Objects in VU are reserved areas in the memory. In this respect, they are very much like variables. In an object, the user can store any sequence of statements. This sequence can be executed over and over at will, simply by calling the name of the object. The creation of an object starts with the keyword "object", followed by an object name. The name must start with the "$" sign. The name can be followed by a number of other elements. In this project, the object names are just followed by a parameter list. The body of the object comes last. The body is called the object list. The object list is enclosed in curly brackets, "{ }", and it is made of one or more statements. It is possible for an object to call other objects. Therefore, complex objects can be constructed from simpler ones. This modularity offers better control and maintenance of large databases.

3.6 VU catalogues and projects

Objects can be packaged into specialized catalogues. Objects may have the same names if they reside in different catalogues. This is possible because VU adds the catalogue name to each objects inside that catalogue. Each VU program can have one unique section called "project". The code embedded inside the "project" calls all the objects from the different catalogues to produce the final output.

3.7 VU modeling stack

VU supports a 3D modeling stack and a set of commands to manage the stack. These commands are summarised in Table 3 below:

flush:	Cleans the stack and resets the top of the stack.
push:	Pushes a matrix onto the stack.
pop:	Pops a matrix onto the stack.
replace:	Replaces the matrix at the top of the stack with the provided matrix.
concat:	Combines or multiplies the provided matrix with the matrix at the top of the stack.

Table 3. Stack Commands

The stack allows the creation of complex transformations that will be applied to the 3D entities, thus, creating different worlds where objects are temporarily affected. The stack statement is probably the most important modeling statement in VU. Complex transformations are achieved by combining simpler transformations. VU has several builtin transformations, mainly:

- General 3D rotation around a general axis in 3D space.
- x,y,z rotations.
- General 3D scale, as well as x,y,z scale.
- General translation in 3D space as well as x,y,z translation.
- VU provides a mechanism to automatically align any 3D object or primitive along a chosen axis in space.

This feature allows the user to designate a flight path and make an object follow it.

Table 4. VU Transformation

The user can also create customized transformations. In turn, these transformations can be combined with the built-in set. It is important to note that if an Object is transformed by a matrix that resides on the stack, this transformation does not affect the database of the object itself. It only affects the way the user perceives the object.

3.8 VU camera and absolute cardinal point Ward)

VU has an internal default camera which is the active camera. The name of the internal camera is simply "camera" . VU's hypothetical cameras act in many ways like real cameras, with minor differences. A VU camera does not need to be focused and can defy the laws of physics. A VU camera structure contains, among other thins, a "focal" value, a "position", and a "tar". "focal" is of type value, whereas, "position and "target" are of type point which cant used wherever a point is used. Consider the following section of code extracted from the file Team3.VU:

```
8.      object $WindowFrame
9.      {
10.     // A dummy move just to set Acard
11.         up (0, 0, 0);
12.     /*  range is distance between the
13.         Acard and the camera */
14.         declare range = vlen (Acard .-
        position);
15.     /* if range is less than 120
16.         show more details */
17.         if (range < 120)
18.             cube (2,8,8);
19.         else
20.             cube (0,8,8);
21.     }
```

In line 14 of the code, the special point "position" is used to measure the distance between the position of the camera and the "Acard' point. The internal VU function "vlen" accomplishes that since it returns the length of a 3D vector. "vlen", in this case, measures the length of the vector defined by the difference of the two points, "position" and "Acard". In other words, "vlen" measures the distance between the two points. The distance is stored in the variable "range". In line 17, the object "$WindowFrame" checks if the value stored in the variable "range" is less than 120 units. If this is the case, a "cube" with the dimensions (2,8,8) is used to represent the shape of the window frame. On the other hand, if the value of range is less than 120 units, a "cube" with the dimensions (0,8,8) is used. This translates into less details. This demonstrates that VU allows the creation of intelligent objects: objects that can take decisions and change their appearances. This project also defines intelligent trees. The trees will show more details as we get closer to them. The tree uses a more 'intelligent" function that allows a smooth change in its appearance as the camera gets closer and closer (line 26 in the code below). The "$WindowFrame" cuts out or reveals all its details at the 120 units threshold. The following is a fragment of the object "$Tree" from the file Team4.VU:

```
8.      object $Tree (.TPos = (0,0,0))
9.      {....
16.         declare
17.             {
18.             /* Modeling stack has affected the tree
```

```
19.                position */
20.                .RealPosition = ptmtx(.TPos,RESMTX);
21.                /* range is distance between the tree
                   and the
22.                camera */
23.                range = vlen(.RealPosition.-position);
24.                }
25.            if (!range) range = 0.0001;
26.            declare res = 2 + (260/ range);
27.            sphere (8,res,res);
```

"Acard" is another special VU point that needs some further explanation. It represents the absolute location of VUs internal 3D pen'. Line 11 of the same code makes a dummy move in order to set the "Acard" point. Al though the up" command moves the 3D "pen" in relative coordinate space (a space affected by the content of the modeling stack), VU makes sure that Acard' is calculated properly in absolute space. The following code shows the exact values of the "Acard' and how it is gets affected by the contents of the stack. In this case, "Acard" was easy to predict; however, this is not the case in most situations.

```
up (0,0,0);              // Acard set to 0,0,0
dn (10,0,0);          // Acard is set to 10,0,0
push tr (10,0,0); // pushs a translation of 10 unit along
the x axis
up (0,0,0);              // Acard set to 10,0,0
dn (10,0,0);          // Acard is set to 20,0,0
pop (1);                 // Eliminates the translation effect
```

It is also possible to obtain the real position of any point in space using the point to matrix multiplication function, as in line 20 of the "$Tree" object. The "RESMTX' is a matrix that represents the content of the entire modeling stack.

```
20.                .RealPosition = ptmtx(.TPos,RESMTX);
```

4 The TwinTower case study project

This simple example demonstrates how a project could be divided over four different production teams. After an initial meeting, each production team proceeds to produce the various tasks that would lead to the final simulation. The project's aim is to model and animate a simple twin tower building and its immediate surroundings. The project and the database is kept extremely simple. The project features two intelligent objects, namely $Tree, $WindowFrame. Both objects monitor the presence of the camera. When the camera gets nearer, they show more details. The object $Tree was written to "lose" or show its details gradually, whereas, object $WindowFrame switches between high resolution and low resolution at the 120 unit threshold.

Team 1 produced the file "Teaml.VU" which relies on work done by Team 2. Therefore, it must include the file "Team2.VU". This is done in line 6 of the file "Teaml.VU". Team l's job is to set the camera and then call the object $Architecture\TwinTowers.

```
1.          /*
2.          ** Filename Team1.VU
3.          ** Team 1 relies on the work done by
4.          ** Team 2. Therefore, Import their work
5.          */
6.          import text "Team2.VU";
7.          project TwinTowers
```

```
8.          {
9.          /* set various point of views 0-3 */
10.         case (0)   // Use another value to change
                camera
11.             {
12.             0: camera {
13.                 focal (35);
14.                 position (-200,-200,200);
15.                 target (0,60,30) ;
16.                 }
17.             1:  camera {
18.                 focal (24);
19.                 position (-90,-50,80);
20.                 target (0,60,0) ;
21.                 }
22.             2:  camera {
23.                 focal (24);
24.                 position (-95,-8,30);
25.                 target (0,60,15) ;
26.                 }
27.             3:  camera {
28.                 focal (24);
29.                 position (-50,-8,20);
30.                 target (0, 60, 15) ;
31.                 }
32.             }
33.         // call object $TwinTowers from catalogue
                Architecture
34.         $Architecture\TwinTowers;
35.         }
36.         bye;
```

When Teaml runs its file for the first time, it would produce no results. However, as the different teams add various details into their respective files, Team 1 starts to see more and more details as it runs its program. Figures 1,2,3,and 4 show the results after running the "Teaml VU" file at random times.

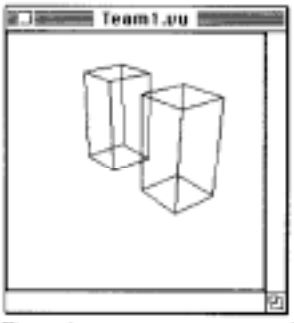

Figure 1.

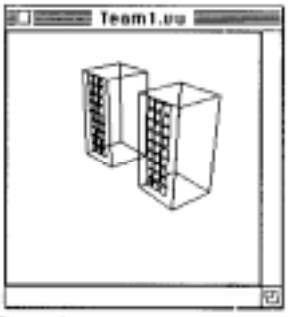

Figure 2.

Figure 1 shows the project at a very early stage The object $Architecure\Tower does not yet contain any reference to $OpeningsV.heWindows (by Team 3) and $Landscape\Trees (by Team 4). Figure 2 shows the project after including a call to object $Openings\TheWindows (by Team 3) into object $Arch.tecture\Tower (by Team 2).

In Figure 3, Team 2 added the garden area to object $Architecture\Tower, and the street line by adding a call to the object $Landscape\Street inside the object $Architecture\TwinTowers, as well as a call of the object $Landscape\Trees. Both these landscape objects are the responsability of Team 4. However, since Team 4 has finished writing the code for object $Landscape\Street, the street started to appear at this stage. Object $Landscape\Trees starts to appear as part of the project a bit later in time as Figure 4 shows.

In Figure 4, the project is completed. Teaml can proceed to inspect the project from various points of view.

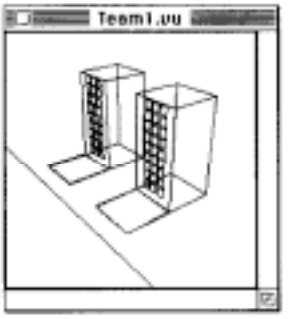

Figure 3.

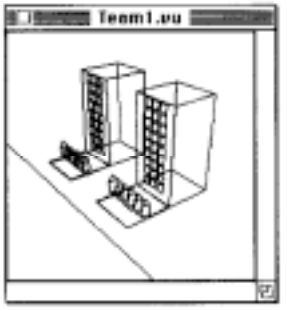

Figure 4.

The following is a summary of each team's work:
Team 1 assembles the project.
Team 2 is responsible for the architecture.
Team 3 is responsible for the openings.
Team 4 is responsible for the landscape.

The following shows the teams' interdependencies:
Team I uses the work done by Team 2 directly and all the other teams indirectly.
Team 2 uses the work done by Teams 3 and 4.
Teams 3 and 4 do not use any other team's work.

The following lists the contents of the various catalogues in the various files:
"Teaml VU" contains: project TwinTower.
"Team2.VU" contains: catalogue Architecture contains: object $Tower, and object $TwinTowers
"Team3.VU" contains: catalogue Openings contains: object $WindowFrame, object $RowOfWindows, and object $The Windows
"Team4.VU" contains: catalogue Landscape contains: object $Tree, object $Trees, and object $Street

In the following figures, the "intelligence" of the $Openings\WindowFrame and the $Landscape\Tree objects is put to the test. Figure 5 (case 1, line 10 of file Teaml.VU) clearly demonstrates that some of the window frames which are closer to the camera position show a double frame. From this vantage point, the tree line shows little details.

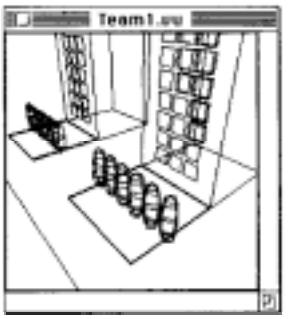

Figure 5.

Figure 6.

Figure 6 (case 2, line 10 of file Teaml.VU) demonstrates that the windows of the tower near the position of the camera show double frames, whereas, this is not the case for the window frames of the tower further away from the camera. Figure 6 also gives a better close up view of the tree line. The tree which is the closest to the camera shows 7 horizontal sections, whereas, the 6th tree shows only 5. Finally, Figure 7 (case 3, line 10 of file Teaml.VU) shows a very close up look of the tree line.

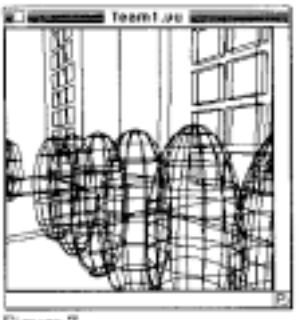

Figure 7

In the following section, the complete listings of the remaining files of the project are given. The listing uses meaningful identifier names for the project, catalogues, objects, and variables. It is out of the scope of this paper to explain the code on a line by line basis. This task is left as an exercise for the reader. Please feel free to get in touch with the author of the paper via e-mail for specific explanations.

(Listing of File Team2.VU)

```
1.      /*
2.      ** Filename Team2.VU
3.      ** Team 2 relies on the work done bye Team 3
        and Team 4
4.      ** Import their work
5.      */
6.      import text "Team3.VU";
7.      import text "Team4.VU";
8.      catalogue Architecture
9.      {
10.     /* object $Tower */
11.     object $Tower
12.     {
13.        $Openings\TheWindows;
```

```
14.             // The frame of the Tower
15.             cube (50, 50, 100);
16.             // Add the Landscape
17.             $Landscape\Trees(.TPos = (-30,0,0) );
18.             // Add The Garden area in front of the
                building
19.             push tr (-50, 0, 0);
20.             cube(50,50,1);
21.             pop (1);
22.         }
23.         object $TwinTowers
24.         {
25.             $Architecture\Tower;
26.             push tr (0, 100, 0);
27.             $Architecture\Tower;
28.             pop (1);
29.             $Landscape\Street;
30.         }
31.         }
32.         bye;
```

(Listing of File Team3.VU)

```
1.          /*
2.          ** Filename Team3.VU
3.          ** Team 3 is working on the catalogue
4.          ** Openings
5.          */
6.          catalogue Openings
7.          {
8.          object $WindowFrame
9.          {
10.         // A dummy move just to set Acard
11.             up (0, 0, 0);
12.         /*  range is distance between the
13.             Acard and the camera */
14.             declare range = vlen (Acard .- position);
15.         /* if range is less than 120
16.             show more details */
17.             if (range < 120)
18.                 cube (2,8,8);
19.             else
20.                 cube (0,8,8);
21.         }
22.         object $RowOfWindows
23.         {
24.             declare a = 0;
25.             push tr (0, 0, 0);
26.             while (a < 5)
27.                 (
28.                 concat tr (0, 10, 0);
29.                 $Openings\WindowFrame;
30.                 a += 2;
31.                 }
32.             pop (1);
```

```
33.         }
34.         object $TheWindows
35.         {
36.             declare h = 0;
37.             push tr (0, 0, 0);
38.             while (h < 18 )
39.                 {
40.                 concat tr (0, 0, 10);
41.                 $Openings\RowOfWindows;
42.                 h += 2;
43.                 }
44.             pop (1);
45.         }
46.         }
47.         bye;
```

(Listing of File Team4.VU)

```
1.          /*
2.          ** Filename Team4.VU
3.          ** Team 4 is working on the catalogue
4.          ** Landscape
5.          */
6.          catalogue Landscape
7.          {
8.          object $Tree (.TPos = (0,0,0))
9.          {
10.             // Translate to .TPos.
11.             push tr .TPos;
12.             // Translate the sphere upward
13.             push tr (0,0,10);
14.             // Scale the sphere into an ovoid
15.             push sc (.6,.6,1.5);
16.             declare
17.                 {
18.                 /* Modeling stack has affected the tree
19.                 position */
20.                 .RealPosition = ptmtx(.TPos,RESMTX);
21.                 /* range is distance between the tree
                    and the
22.                 camera */
23.                 range = vlen(.RealPosition.-position);
24.                 }
25.             if (!range) range = 0.0001;
26.             declare res = 2 + (260/ range);
27.             sphere (8,res,res);
28.             pop (3);
29.         }
30.         object $Trees(.TPos=(0,0,0),NumberOfTrees=6)
31.         {
32.             declare i = 0;
33.             push tr (0, 0, 0);
34.             while (i < NumberOfTrees)
35.                 {
36.                 $Landscape\Tree (.TPos = .TPos);
```

```
37.                 concat tr (0, 10, 0);
38.                 i ++;
39.                 }
40.             pop (1);
41.         }
42.         object $Street
43.         {
44.             up (-70,-80,0);
45.             dn (-70,260,0);
46.         }
47.         }
48.         bye;
```

5 Conclusion

The previous example shows how various parts of a single project could be built simultaneously with the use of a simulation language such as VU. As a matter of fact, a plathora of software has been developed in at the Sasada Laboratory at Osaka University allowing the large population of our lab to cooperate on various projects. Lately, as mentioned earlier, we expanded our physical laboratory space into the virtual space of the Internet. We are hoping to find other laboratories on the web which share our interest in order to cooperate together on future projects.

6 Bibliography

Comair, C.; & A. Kaga. 1995. VU: A Database Computer Language for the Simulation of Events in A City (Part I)" In Proceedings of European Simulation Multiconference 1995

Comair, C.; & A. Kaga. 1995. "VU: A Database Computer. Language for the Simulation of Events in A City (Part II) " In Proceedings of Pan Pacific Conference on Information Systems 1995

Sasada, T. 1994. Open Design Environment and Collaborative Design" In Proceedings of The 12th European Conference on Education in Computer Aided Architectural Design, 3-6.

Kaga A.; Y. Kawasaki; & T. Sasada. 1994. "Design. projects and computer supported cooperative works: A study about doing an open design" In Proceeding of the 17th Symposium on Computer Technology of Information Systems and Applications, 193198.

Sasada T.; Y. Kawasaki; A Kaga; & W. S. ho. 1994. 'A Study On Development And Use Of Design Tools -Open Design Environment-" In Proceeding of the 17th Symposium on Computer Technology of Information Systems and Applications, 187-192.

Kaga A.; & T. Sasada, 1993. "Design projects and computer supported cooperative works: A framework of research" In Proceeding of the 16th Symposium on Computer Technology of Information Systems and Applications, 193-198.

Sasada T.; Y. Kawasaki; & T.Nakayama. 1993. "A study on a Design Tool by ODE" In Proceeding of the 16th Symposium on Computer Technology of Information Systems and Applications, 235-240.

Sasada T.; Y. Kawasaki; & T. Nakayama. 1993. "A Study On Presentation of Environmental Design" In *Proceeding of the 16th Symposium on Computer Technology of Information Systems and Applications,* 241-246.