December 1995

Vu: A Databse Computer Language for the Simulation of Events in the City

Claude Comair
Osaka University

Atsuko Kaga
Osaka University

Vu:
A Database Computer Language for the Simulation of Events in the City.

Author: Claude Comair
Osaka University, Faculty Engineering,
Dep. Environmental Engineering

Co-Author: Atsuko Kaga
Osaka University, Faculty Engineering,
Dep. Environmental Engineering

The subject of this paper is to present a new computer language dedicated to the creation and maintenance of city databases. The new language is called Vu, (pronounced Vee-Yoo). Vu is the brainchild of Claude Comair who first presented the language in his 1988 master's thesis at Osaka University, Faculty of Environmental Engineering, Mr. Comair is now a researcher and writing a Doctorate thesis at the same university.

This new computer language actually transfers total control to the database, thus, allowing the creation of concise and intelligent databases. Databases of cities are notorious for being complex, long, and containing heterogeneous data types. Furthermore, most of the current database formats focus on few aspects of the definition of city objects. Some databases define the data objects in terms of their shapes, while some others will define them in terms of cost or list of contents. Vu, on the other hand, imposes no limits on the type and order of the information in the data stream.

1 Introduction:

In our close circle of friends, comprised mainly of urban planners who use computer technology to simulate events in the city, we have a firm belief that "he who owns the database, controls the world (the virtual world at least)". If I recall it well (this goes back to the early 80's), we started to believe this after watching a couple of science fiction movies. I remember that in these movies, there was always a bitter and destructive nuclear war that divided the people, then the control of a certain substance that became rare assured the rule of the planet. In one case, it was the "spices", and in another movie it was the "gas". In our case, the database of a city (or a portion of a city) is the most sought after "substance". The database has always presented a range of difficult and sometimes impossible challenges. These challenges have varied, from lack of computer power, computer data storage space, unavailability of adequate computer software, to limited manpower. Despite these limitations, we have managed in our Laboratory at Osaka University, over a period of ten years of relentless research, to generate databases and computer based simulations for many cities and large urban projects. Some of these projects are listed below:

- **The database for the downtown core of the city of OSAKA Japan, (02/1983).**
- **The database of major shrines and the hills of Kyoto, namely Kyomisudera temple, one of the largest wooden structures in the world (08/1983).**
- **The city line and Portopia district of the city of Kobe, Japan (03/1984).**
- **Portions of the Shinjuku district in downtown Tokyo (10/1983).**
- **Parigraph '85.**
- **Kumamoto city and Castle in Japan.**
- **Shanghai (city database and various phases of new development in the city 1987-89)**
- **Shanghai Railway Station (from 1~11)**
- **Kansai International Airport (05/1987).**
- **The ancient city of Lou-Lan in China**
- **Automatic Parametric Data Generation of Ancient Japanese Temples.**

Table 1 Various Projects

It would be impossible to give credit in this paper to every researcher and student who contributed to these projects with the exception of one person, Prof. Tsuyoshi Sasada. All the above research and work was done under his supervision and direction at Osaka University, Faculty of Engineering, Department of Environmental Engineering.

Over the years, we have learned how to overcome the different problems that faced our team. Our success can be credited to the Laboratory's attitude of actively participating in the creation of the tools that we use. Specialized computer controlled hardware and various computer software were developed for the various tasks. In a sense, this attitude allowed us to better control the destiny of our research, and not to be completely dependent on some remotely located development team.

2 Vu: A Computer Language

Vu is a modern computer language. It supports most of the features found in general programming languages. Since Vu is a problem oriented computer language, it offers a specific support to the creation and maintenance of city planning related databases. Vu has evolved over a period of ten years and it is under constant upgrade. Since this paper does not represent the official document of the language, it just describes some of the key features of the language.

2.1 Free Format Database

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. Again, good sense and style can dramatically enhance the readability of programs. The following examples illustrate what are considered to be good and bad styles in programming. Vu will not see the difference between these two versions, and both versions will produce the same result. However, the usage of well chosen identifiers, well commented code, and a consistent style are the best recommendations you can get from a fellow programmer.

```
object  $box (side = 10)
              {
              cube (side, side, side);
              }
$box (side = 5);
kill $box;
bye;
```

Code 1

```
object $box(side=10) {cube(side, side ,side );
} $box (side=5);kill $box; bye;
```

Code 2

2.2 Comments in the Database

Vu accepts two different ways to comment. Vu considers a comment as any sequence of characters enclosed between the opening construct "/*" and the closing construct "*/".

```
/*

This could have been a very long comment
It could expand over several lines
...

*/
```

Code 3

The second form of a comment starts with the opening construct "//" and ends with a new-line character.

```
// this is until the end of the line comment
```

Code 4

Comments are ignored by Vu. It is certainly a good habit to comment your code. Comments render the code more readable, thus, easier to upgrade, change, and debug.

2.3 Various Data Types of Variables:

Vu allows the user to declare several types of variables. These types are:

- **Variable Values**
- **Variable Points**
- **Variable Matrices**
- **Variable Strings**
- **Variable Cameras**
- **Variable Binary Data Blocks**
- **Variable Text Data Blocks**

Table 2 Vu Variables

2.3.1 Defining Variables in Vu:

By declaring a variable, Vu reserves an area of the memory large enough to hold the data type of the variable. The user gives a name of his/her choice to this area, then the name of the variable could be used in any statement or expression where the type can be used. At any given time, the user can change the content of a variable, hence the name variable. When a variable is no longer necessary, Vu provides the possibility to "kill" the variable and release the reserved memory. Variables turns Vu into a powerful programming language. Thus, the programmer's control over the database is increased. Vu also provides a large set of operators and functions to manipulate mathematical and lexical operations on these variables.

In many cases, Vu uses the first letter of a variable name to recognize its type. Thus, the programmer does not need to constantly refer to a declaration sheet to remember the type of certain variables. The disadvantage of this method remains in the fact that the user has to type the "type-character" at the start of the variable name. However, this is a small price to pay in order to make the code more readable. The table below shows the different "type-characters":

- **Identifiers starting with the "." character indicate that this variable is a 3D point.**
- **Identifiers starting with the "#" character indicate that this variable is a 4x4 matrix.**
- **Identifiers starting with the "@" character indicate that this variable is a string.**
- **Identifiers starting with the "$" character indicate that this is an object. Objects are not variables and they will be presented later in this paper.**
- **Identifiers starting with the "~" character indicate that this is a camera variable.**

Table 3. Type Character

The following examples illustrate how easy it is to declare, initialize, and recognize the type of variables at first glance.

```
declare       .a = (10,10,10);       // a point
declare student_age = 35;            // a value
@student_name = "John";              // a string
#by10 = |
       (10,0,0,0)
       (0,10,0,0)
       (0, 0,10,0)
       (0, 0, 0,1)|;                 // a matrix
```

Code 5

The initialization of the variables is an optional process in Vu. If the variables are not initialized, Vu assigns a default value according to the following table:

type	initial value
values	0.0
points	(0.0,0.0,0.0)
strings	""
matrix	IDMTX
camera	camera
Binary Data Block	No default
Text Data Block	No default

Table 4. Default Values

Vu variables can be declared anywhere in the program or the database. When Vu encounters an undeclared variable for the first time, it will define and initialize it automatically. Therefore, it is not necessary to explicitly declare a variable in Vu. However, it is a better habit to use the "declare" statement to explicitly declare the variables. The declare statement allows the programmer to declare one or more variables. This feature allows the programmer to avoid having to repeatedly type the word "declare". The usage of the declare statement is shown in the following example:

```
declare pi = 3.14;
declare
        {
        .a = (10,10,10);
        student_age = 35;
        @student_name = "John";
        #by10 = |
                        (1,0,0,0)
                        (0,1,0,0)
                        (0,0,1,0)
                        (0,0,0,10)
                        |;}
```

Code 5

2.3.2 Simulation of Arrays

Identifiers in Vu could be followed by an expression or a succession of expressions separated by commas, all enclosed between square brackets "[ ]". This addition to the identifier is called the stem. The limited scope of this paper will not allow us to cover the Vu expressions. For now let us consider that Vu expressions are constant values and value variables.

```
a[10] = 2;
.p[1] = (10,10,10);
.p[a[10]] = .p[1];
a[1,1,1,1] = 30.0;
```

Code 6

In the above example, .p[a[10]] is equivalent to .p[2] because a[10] was defined to be equal to 2. The stem could be used to mimic the usage of arrays, but the programmer must keep in mind that indexed identifiers in Vu are not arrays. Vu, after resolving the expressions in the stem, will concatenate the result to the end of the base name. Thus, the variable a[10] in Vu could exist alone, and does not implicate that the variables a[0] to a[9] exist. Programmers who are used to using arrays in other languages may use the stem to mimic the usage of arrays. However, they should keep in mind the following facts:

1. In other languages, arrays when declared, will hold a chunk of memory large enough to accommodate all the elements of the array. This memory remains held during the entire life of the array, and all the elements of the array will coexist. In the C programming language, for example, an array of 40 characters is declared as follows and will need 40 bytes of memory:

```
/* C declaration of an array of 40 chars */
char   a[39];
```

Code 7

   After this declaration, individual elements of the array "a", can be referenced by using the base name for the array "a" followed by an index or a number enclosed in square brackets, "[ ]" representing the rank of the element in the array. For example, a[9] represents the 10th element of the array "a".
2. In Vu, the existence of variable a[10] does not necessarily mean that the variables indexed from 0 to 9 exist. If the user wants to mimic the usage of an array of 11 (0 to 10) elements in Vu, for example, the programmer must declare each variable indexed from 0 to 10, and must ensure that none of the elements are destroyed by mistake. The declaration of the different variables could be easily achieved by using a loop. The following examples illustrate the process:

```
// one dimension array
// declare the counter
declare i;
// declares a[0] to a[4]
for (i = 0; i<= 4; i++) declare a[i] = i;
// print them:
for (i = 0; i<= 4; i++)
      print ("a["+,i+,"] = "+,a[i]);
```

Code 8

The above code outputs the following text:

```
a[0] =        0
a[1] =        1
a[2] =        2
a[3] =        3
a[4] =        4
```

Output 1. Output of Code 8

```
// two-dimensions array
// declare @a[1,1] to @a[10,12]
for (i = 1; i<= 5; i++)
      for (k = 1; k<= 6; k++)
            declare@a[i,k] =
"["+i+","+k+"]";
// print them:
for (i = 1; i<= 5; i++)
      {
      print (" ");
      for (k = 1; k<= 6; k++)
      print (@a[i,k]+," "+);
      }
```

Code 9

The above code outputs the following text:

```
[1,1] [1,2] [1,3] [1,4] [1,5] [1,6]
[2,1] [2,2] [2,3] [2,4] [2,5] [2,6]
[3,1] [3,2] [3,3] [3,4] [3,5] [3,6]
[4,1] [4,2] [4,3] [4,4] [4,5] [4,6]
[5,1] [5,2] [5,3] [5,4] [5,5] [5,6]
```

Output 2. Output of Code 9

2.3.3 Scope of a Variable

In Vu, declarations of variables can only happen in two places. Variables are declared either inside an object (this includes variables declared as parameters), or outside an object. Those variables declared outside an object are said to have a global scope. Variables that are declared inside an object are said to have a local scope. Global scope variables are known inside and outside the objects from the moment they are declared until the moment they are killed. Local variables are known only inside the object that declares them. Variables declared inside an object need not be killed. Vu kills local variables when it finishes executing the object. When Vu terminates, Vu returns all the memory to the operating system whether the program explicitly kills all globally declared variables and objects or not. It is still a better habit to explicitly kill any variable as soon as it is not needed any longer.

2.3.4 Killing a Variable

Once a variable is declared, Vu reserves a chunk of memory. This chunk is reserved as long as the variable is not killed. It is a good habit to release back to Vu the memory by killing a variable as soon as it is not needed any longer. This will allow the user to write larger code . Indexed variables behave no differently from any other variables. Again, a loop is handy to kill all the declared elements.

```
// kill example: declares a[1] to a[5]
for (i = 1; i<= 5; i++) declare a[i];
      // Use them here ...
// release the memory when done for good
for (i = 1; i<= 5; i++) kill a[i];
// release i
kill i;
```

Code 10

The user dynamically manages the memory usage. This way the usage of the memory is more efficient. This is rather important when running Vu in multi-user or multi-tasking environments, in which the resources of the computer are shared among many.

By allowing the user to dynamically manage all the space allocated to variables and mostly to index variables, we hope that Vu will satisfy pointer advocates. After all, arrays are much easier to master and use than pointers. Nevertheless, they have been snubbed by many programmers because they grab the memory at definition time. This memory is fixed in size whether it is needed or not. Furthermore, the allocated memory cannot be released dynamically. The array has to go out of scope for its memory to be released. If the array is declared to be global, The memory will be locked for the entire life of the program. Vu offers the possibility of dynamically managing arrays. Vu allows the programmer to add new elements to the array it also allows to subtract elements from the array. This is achieved simply with the declare and kill statements.

2.3.5 Alias of a Variable

Each variable in Vu has a name and an alias. The alias of a variable is by default the name of the variable itself as shown in the following example of declare statements:

```
declare a=10;
declare b<-a;
```

Code 11

The first statement means that the variable "a" started to exist and that it holds the value 10. It also means that the alias of "a" is "a".

The second statement means that variable "b" also started to exist, its alias is "a" instead of its own name as in the previous statement. Variable "b" holds the value of "a" which is 10.

The second statement in the above example uses the Vu alias assignment operator "<-". Now that the alias of "b" is "a", it is said that the variable "b" represents "a", and if a value is assigned to "b" as in the following code, then Vu assigns the value 20 to "a" automatically as well:

```
b = 20;
```

Code 12

It is important to know that the two variables exist independently and they occupy separate areas of memory. Therefore, when killing one of the variables, the other variable remains. In the above example, killing "b" first will not affect "a" in any manner. On the other hand, if "a" is killed first, "b" will not only remain to exist, but when a value is assigned to "b", Vu will try to assign the same value to its alias "a". When Vu is not going to find the variable "a" anywhere, it is going to reset the alias of "b" to "b" itself, and further assignment statements involving

variable "b" will not trigger an unnecessary search for a non-existent variable.

Alias assignments are often used in object parameters. This allows an object to affect outside variables. Consider the following example of an object that swaps two outside variables:

```
object $Swap(x = 0, y = 0)
       {
       declare tmp = x;
       x = y;
       y = tmp;
       }
declare { a = 10; b = 20; }
// will not swap
       $Swap(x = a, y = b)
// will swap
       $Swap(x <-a, y<-b)
```

Code 13

2.3.6 Value Variables

Value variables in Vu hold integers (Hexadecimal, Octal, Decimal) as well as maximum precision Floating point values (in Decimal and Engineering notations).

```
age = 35;              // Integer
memory = 0xf125        // Hexadecimal
char = 077;            // Octal
f = 2.5;               // Float (double)
big = 2.3e21           // Float (double)
```

Code 14

2.3.7 Points Variables:

```
//points in space
.a = .b = .c = (10,10,10);
.o = .a;
```

Code 15

In 3D space, the point is the smallest entity that someone can address. A point represents a location of the 3D space. This space will be referenced in this document as the "real world". All the primitives and objects that the user will create belong to the real world.

In order to locate points in the real world, the user sets an arbitrary location as the origin (named o) of his/her working space. A rectangle (Cartesian) coordinate system, called "the world coordinate system", is established at the origin. This can be obtained by drawing three mutually perpendicular axes, named oX oY oZ. It is usual for architects to consider the oXoY plane as the surface of the ground on which the building is constructed. The third direction, oZ, is perpendicular to the oXoY plane. Two different orientations for the oZ direction are possible. Suppose that the viewer is standing at the origin (o) on the XoY plane (on the ground). One possible direction is represented by the vector pointing from the origin towards the head of the viewer. This coordinate system is called the "right-handed coordinate system". The second possible direction for the oZ axis is the one starting at the origin and going in the opposite direction of the head of the observer. Such a coordinate system is called the "left-handed coordinate system". The right-handed coordinate system seems more appropriate to describe architectural objects since these objects are usually standing on the ground and pointing upward. The figure below shows the difference between the left-handed and right handed coordinate systems.

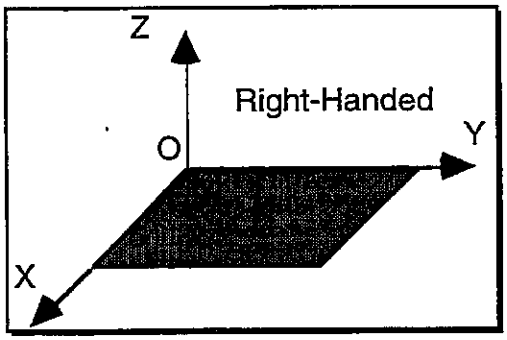


Figure 1. Right Handed Coordeinate System

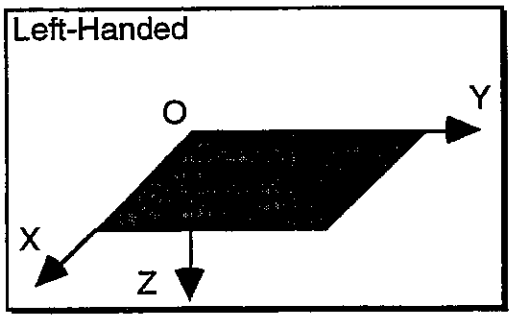


Figure 2. Right Handed Coordeinate System

By using a coordinate system, points can be identified by a unique ordered tripletof numbers, called coordinates (x,y,z) . These coordinates represent the direct distance of the point to the YoZ, XoZ, XoY planes, respectively. Thus, a point "p" in space can be identified by the ordered triplet (1,1,1), for example. The origin "o" is identified by (0,0,0).

A single point cannot be viewed for the simple reason that a point has no dimension. Vu simulates the motion of a pen that can take any position of the 3D space. This pen moves from the current location to a new one. If the pen is "up" it does not leave a trace in space. On the other hand, if the pen is "down" it leaves a trace between the points. The current location of the pen is known as the "cardinal point". Vu offers a certain number of operations between points, including special operators for dot product, cross product, and points transformed by matrices among many others.

2.3.8 Matrix Variables

Vu matrices are currently 4x4 matrices. Vu provides a large amount of matrix operations. This includes matrices and matrices, points and matrices as well as values and

matrices operations. Furthermore, matrices can be pushed on the model stack to alter the 3D world in which the simulation is taking place. (More on this topic will be covered later).

```
#m = ro (45);
#t =  |       (1,0,0,0)
              (0,1,0,0)
              (0,0,1,0)
              (10,11,12,1)
      |;
```

Code 16

2.3.9 Camera Variables:

```
~c =  {
      focal ( 70 );
      position (-200,-200,200);
      target (0,0,0);
      ...
      };
camera = ~c;  // Set the hypothetical camera
~b = camera;
~t = ~b;
camera
      {
      focal ( 100 );
      position (-200,-200,400);
      target (0,0,10);
      ...
      };
```

Code 17

Vu supports a camera type variable. Vu has an internal default camera that is the active camera. The name of the internal camera is simply "camera". Camera variables could be assigned "camera", and "camera" could be assigned any camera variable . Vu hypothetical cameras act exactly like real cameras. The above code shows that a camera structure contains among other things a focal value, a position, and a target. Unfortunately, the explanation of this topic is beyond the scope of this paper.

2.3.10 String Variables:

```
@fname = "Forrest"; @name = 'Gump';
@fullName = @fname + " " + @name;
```

Code 18

A String in Vu is any text that is enclosed in either a single or double quotes (as shown in the above example) . Operators are provided to sort, concatenate, search, and compare, just to name a few.

2.3.11 Binary Data Block

```
BBlock "testImage" { 0011001.... }
```

Code 19

A BBlock is a large binary chunk of binary data. This is useful to include multimedia data types (movie, sound, graphics etc.).

2.3.12 Text Data Blocks:

```
TBlock "listText" { this is a test ....}
```

Code 20

A TBlock is a large ASCII data block. This type of variable is convenient to include large amount of text into the definition of objects. This function is useful to include general information into the database (letters, list of names, etc.). TBlocks can be searched for patterns, patterns can be extracted form TBlocks and saved in string variables. Furthermore, they can be concatenated just like strings. In other words, a TBlock looks like a very large string. The difference between the two lies in the internal implementation of the two types.

2.4 Code or Object Blocks:

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. Once created, an object resides in the memory until destroyed using the kill command. The following is an example of a creation and use of an object:

```
// Create an object

$table[1] (size = 10, shape = 1)
      {
      case (shape)
            {
            1: cube (size,size,size);
            2: cylinder (size,size,36);
            }
      }

// Call the object

// Use defaults for all parameters
$table[1];

// Use default shape
$table[1](size = 5);

// use default size
$table[1](shape = 2);

// order of parameters is not important
$table[1](size = 5, shape = 2);
$table[1](shape = 2, size = 5);

// Reset the defaults of the parameters.
reassign $table[1](size = 5, shape = 2);

// Recall the object with the new defaults.
$table[1];
```

Code 21

The creation of an object starts with the keyword "object", followed by an object name, followed by an optional stem, or index. The stem could be followed by one or more parameters separated by commas and all enclosed in parentheses, "( )". 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 could be constructed from simpler ones. This modularity offers better control and maintenance of large databases.

2.5 A large set of 2D and 3D primitives.

Vu maintains a 3D pen. Vu offers several commands to move (and draw) in 3D space. Aside from the pen, Vu recognizes several simple and complex 3D primitives (polygonal and spline based).

2.6 Model Stack

Vu supports a 3D modeling stack, and a set of commands to manage the stack. These commands are:

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

Table 5. 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 built-in 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 allow the user to designate a flight path and make an object follow it.**

Table 6. 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.

2.7 Import and Export of Data

Vu offers the possibility of importing and exporting several types of files. Vu make it easy to import and export data form or to several different environment.

2.8 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 functions or object calling.

Flow Control statements are generally included in a computer program to alter its linear execution. Single CPU (Central Processing Unit) based computers execute the instructions in a computer program one instruction at a time (even when they appear to execute the statements in a parallel or multitasking fashion). Flow control allows the programmer to simulate real life events. Consider the following events and how they translate directly into a computer program:

Example 1:

If the weather is nice, I will walk to school; otherwise, I will take the bus.
This translates to:

```
if (GoodWeather)
      $Walk(destination = .toSchool);
else $TakeTheBus(destination =.toSchool);
```

Code 22

Where:

"if .. else" is a conditional statement. **"GoodWeather"** is a variable that is set to true is the weather is nice and to false otherwise. **"$Walk"**, and **"$TakeTheBus"** are functions that take an input parameter called **"destination"** which was initialized with the point variable **".toSchool".**

Example 2:

The following segment of code simulates the operations of a control panel.

```
case ControlPannel
      {
      1:    startEngin;
      2:    stopEngin;
      3:    goFaster;
      4:    goSlower;
      }
```

Code 23

By including flow control statements in the definition of objects, the programmer gives to the objects some kind of artificial intelligence or behavior. Thus, data and objects in the database are able to behave differently under different conditions (simulate an inferno, or an earthquake, for example). In Vu, this decision-making capability has been transferred from the interpreter to the database. This transfer has many advantages, mainly:

1) Data can mimic real life situations.
2) Building databases is costly and complex. Therefore, it is in the database where most of the investment should be made and should remain. Usually large databases are costly and time consuming to build. Such databases are meant to survive the test of time and the changes made to hardware and software. This means that transporting the database from one technology to another must be made easily. If all the decision-making data as well as static data are embedded into the database, all we need to transport the database to a newer environment is to re-write the interpreter for the new target technology. Databases of cities certainly fall under this category since they are usually extremely large and must be updated on a regular basis.
3) Flow control allows us to produce very concise databases To illustrate this, consider the following example:

```
// filename: cubes.vu
for (i=1;i<=10;i++) cube(i,i,i);
```

Code 24

The above example (56 bytes including the comment line) would have required a much larger database if expressed in terms of move (up) and draw (dn) actions followed by three-dimensional coordinates as shown in the following partial listing (9640 bytes).

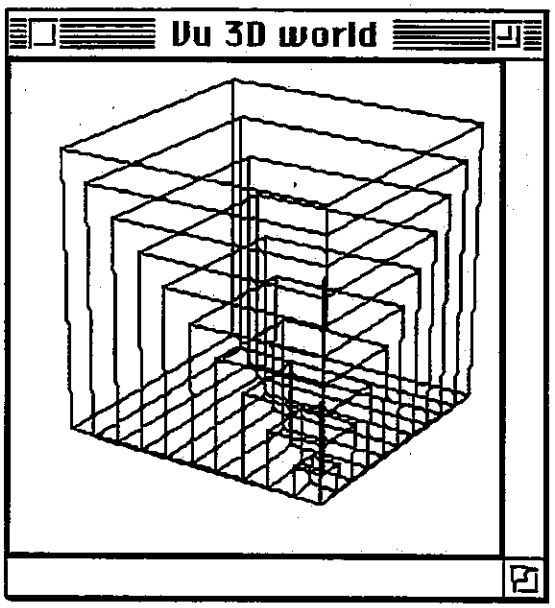
```
// filename: cubes.pri
up (0.000000,0.000000,      0.000000 );
dn (1.000000,0.000000,      0.000000 );
dn (1.000000,0.000000,      1.000000 );
                  ...
dn (0.000000,0.000000,10.000000 );
dn (10.000000,0.000000,10.000000 );
dn (10.000000,10.000000,10.000000 );
```

Code 25

3 Final Note

It is obvious that by including all the above features into the database, we have truly passed the total control to the database. By doing so, we obtain a more concise, more versatile, and portable database.

Finally, I have to mention that this paper does not represent the formal definition of the Vu language. We are currently working on the official Vu manual and hope to be able to make it available soon.



Output 3. Output of Code 24 or Code 25

4 Bibliography

Comair, C.; A. Kaga. 1995. "Vu: A Database Computer Language for the Simulation of Events in A City" In *Proceedings of European Simulation Multi-conference 1995* (Currently under publication).

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; and 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*, 193-198.

Sasada T.; Y. Kawasaki; A Kaga; and 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.; and 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; and 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.