ACTIVE Virtual Reality Modeling Language (AVRML)

In this section, we'll concentrate wholly on Active VRML (Virtual Reality Modeling Language), which works on many browsers and is freely available. We'll be teaching you the most dynamic and 'Active' of them all, Microsoft's **Active VRML**. Knowing Microsoft, which came out with such ingenious names as 'Microsoft Word' for their word processor and 'Microsoft Project' for their Project Manager, you shouldn't be too surprised to know that they've decided to add the word **Active** to the names of all their Internet related products. Perhaps they hope that we'll forget their previous ' in**ACTIVE**ity ' on this front and forgive the late arrival of their products. Indeed, its only recently that Microsoft woke up to the fact that its virtual monopoly on the computer software industry may be broken, if it didn't start dabbling in Internet related technologies soon. The people at Microsoft started burning the midnight oil, especially after seeing the industries exuberant response to the very immature anddefinitely premature Java programming language. So, they renamed and revamped their OLE Controls (OCX) and launched a whole slew of Internet applications, of which Active VRML is a part.

Active VRML is a complete programming language, unlike VRML 1.0, which was the first version of the VRML standard. VRML 1.0 did not have many of the vital features of a programming language. No variables, no loops, not even an if statement. All it could do was display a Virtual Reality model on the screen, with absolutely no interaction with the user !!! Active VRML has changed all that. Now we can produce some real Multimedia applications and embed them into our Web pages.

You've probably noticed, that to keep ahead of the pack, you need to be a veritable linguist ! There are tens of computer languages out there, jostling for your attention, while you struggle to master and keep up with the few languages you know. Some, like Java, appear promising at first, and then fizzle out. Others like C/C++ are perennial favorites because of their versatility and speed, yet even they are not appropriate for every thing. So far, its been a fact of programming life that you have simply got to learn new and more complex languages and try and not feel nostalgic about old ones.

But enough about the background of VRML, lets get down to doing what you've come here for, learning Active VRML.

There are two ways in which you can embed VRML applications into your HTML page. One way is to write a program in Active VRML, which can be called and used by Microsoft Internet Explorer. This is an extremely crucial aspect of VRML technology and forms the basis of its charm. The other way is to have another program called REPL which can be run separately by you. At first we'll use the REPL alternative, because it makes learning easier, but eventually you'll have to use the first method. After all, the USP of VRML is that when a netsurfer, with a hair trigger mouse button and an itchy finger, drops by your page, he sees live animation and hears the sound files you've created for him. This is possible because as soon as he lands on your page, an .avr file zooms across to his machine and is promptly played by his browser. So now, you no longer need cumbersome and finicky Java to play your animations, you can use Active VRML for all your Multimedia needs !!

But there is a small point we forgot to mention, the Internet Explorer does not understand Active VRML

!!!! Then how, you ask, can it play .avr files ? Well, that's where ActiveX Objects come into the picture. Since Active VRML is a programming language, only an ActiveX Object can understand it. So to utilize VRML, you need to have a relevant ActiveX Object registered on your machine along with an Active VRML file. There is another side to this partnership. Since the Active VRML file is played using an ActiveX Object, we can also run the same animation file from a Visual C++ program or a Visual Basic program or any other application running under Windows 95, since they all support OCX's. This means that a programmer, using Visual C++, which is quite inappropriate for animation, can now run an Active VRML file from his application using ActiveX Objects and can include Multimedia in his C++ program.

To use Active VRML you need to have the following.

1. The ActiveX Development Kit .... (12 MB)
2. The Active VRML.........................(6 MB)

Once you have downloaded all the relevant material and installed the ActiveX Development Kit, you go to the Microsoft Internet subdirectory. We have a subdirectory there called A-VRML, and this directory contains the program REPL. If you want you can work here, or else you can make another directory and set path to this one .We've also given you a few sample .gif .jpg .bmp and .wav files along with the programs. We recommend you use the files provided and not any other ones because these files are tailored to fit the program requirements exactly. The Display should be set to 256 colors.

Now lets kick off this tutorial with a really elementary 3 line program.

Program 1

```
// ActiveVRML 1.0 ascii

f = first(import("small.jpg"));
model = f;
```

Give this program an .avr extension so that Internet Explorer can recognize it as an Active VRML program. The first line in the program is a comment line, We don't know whether it is necessary because we have worked on the Alpha version but it seems that if you don't have this comment line, nothing works. For now, we've worked on version 1.0 but later you might have to change the line to // ActiveVRML 2.0 ascii with AVRML version 2.0. After the comment line, you have *f* which is a variable. Now as a seasoned programmer in your chosen language, you know that the environment that you use, gives you several, useful data types. In C we have ints and chars and longs and pointers and structures. DBase has something called fields and records. So in the same way Active VRML has a large number of unique data types, which can be put to good use. We will explain them as and when the need arises. In the **first** function we have **import**. Now **import** is a built in library function of which Active VRML has a large store. It says we want to 'import' small.jpg which it understands to be nothing more than a picture file. The problem with **import** is that unlike a C/C++ function, **import** does not return one value, it returns multiple values. For example , because **import** is dealing with a jpg file, along with the image, it will return it's height it's width, and the resolution of the image. Right now, we are only interested in the image part of **import**, and not in any of the other aspects, so we use a function called **first** which will pick up only the first of these multiple values.

So *f* is now a data type, known as an image data type, which Active VRML understands. The semi colon at the end is a must. If you know C, you'll remember that main always got called first, that was the starting point. Here let us assume that the starting point is **model**. So we say **model** is equal to *f*. Next you type in **repl a1.avr** , i.e. the name of the program. As soon as you press enter you'll get a message and then a greater than sign (>). There you say **# add**, leave a space and then you type in the title of the window. We specified **m** as the title and enclosed it within in **double inverted commas**. We then type in a **comma** and then **model** with two **semicolons** at the end of the line.

i.e.

d:\Microsoft Internet\A-VRML>repl a1.avr{Enter}
Welcome to ActiveVRML version 1.0
Using a1.avr ....
># add "m",model;; {Enter}
>

When you press enter, repl calls an ActiveX Object which can run VRML programs. The first word that Active VRML sees is **model**. Now because it next notices *f* and since *f* stands for the image small.jpg , you will see a small window with **m** as its title and inside this window you will see the picture. Which now means that we can view pictures without too many problems. Then in the second program we will display the sample .gif file, in the third one we will display a .bmp file, in the fourth you can hear a .wav file.

**If the program does not end normally press Ctrl+C at the greater than sign.*

Program 2

```
// ActiveVRML 1.0 ascii

f = first(import("active.gif"));
model = f;
```

Program 3

```
// ActiveVRML 1.0 ascii

f=first(import("setup.bmp"));
model=f;
```

Program 4

```
//ActiveVRML 1.0 ascii

f = first(import("a1.wav"));
model = f;
```

So all that these first four programs have demonstrated is that Active VRML reads and can understand common file formats. You can display them all on your screen. After all this, you'll probably turn around and say "so what, any browser worthy it's name can do that" , but then, we had said we were going to start from scratch.

Program 5

```
//ActiveVRML 1.0 ascii

f = first(import("small.jpg"));
g = first(import("active.gif"));
model = f over g;
```

Now in program 5 we have got 2 images, *f* and *g*. In the last line we say **model** = *f* over *g*. So now the first thing the program will do is display *g* and then overlay the image with *f*. If you had said *g* over *f* you would have had a completely different result. So this proves that using Active VRML we can superimpose one image over the other. If you didn't understand the statement that we just made, don't panic, we will elaborate on it later and you will grasp it over a period of time.

When you write C/C++ code, you do all the grunt work yourself. You are the one in charge, and you specify every action in the code you write. Active VRML however, it is meant for the lazier ones among us , it's meant for the thinkers, for people who don't have the time to sit down and write reams of code. While programming in Active VRML you simply sit down and think of the rules. You then state the rules in your program and Active VRML figures out what to do with them. We have called this concept event-driven programming, but a more appropriate name might be couch potato programming !!

Program 6

```
// ActiveVRML 1.0 ascii

model = time;
```

To start with, we say **model** = *time* where *time* is a reserved word. Every aspect of Active VRML programming has a time component. To prove it, we'd like you to run this program. It will keep displaying the time in seconds. The time will start with the value 0 and then instead of displaying 1,2,3,4,5....., it will display 1.12039, 1.6575, 1.9989, 2.0034....., but this still gives you the time in seconds. So by now, you must have noticed that wherever you use *time*, it's value keeps changing. In other words, the variable *time* will always contain the current time. You'll also realize that unless you use ALT+F4, your program wouldn't stop working !!. In other words it starts to execute a infinite loop.

Now lets proceed to program 7.

Program 7

```
// ActiveVRML 1.0 ascii

a=simpleText("hello");
b=renderedText(a);
c = first(b);
model = c;
```

We've mentioned data types before. In this program hello is a string data type, a concept Active VRML understands. **simpleText** is once again a built in function It takes a string and converts it into text. Now what you have to understand is that when you say **model** = *c*, everything starts to go backwards, i.e. the statements are executed bottom line first, But for the sake of comprehensibility, we'll explain it all to you the other way round. **renderedText** asks for text from you, and it then converts it into an image.

Therefore *b* is now an image. Remember that the image not only returns images, but also the width, height and the resolution of the specified image, as and when demanded. Thus *b* is more than just an image. *c* is where we remove the image component and when we say **model** = *c* , we will see hello displayed in the center of our screen. The message will be displayed in white.

If you don't like to write programs over a number of lines, you can do it as we have done in program 8 , and shorten everything into one line.

Program 8

```
// ActiveVRML 1.0 ascii

f = first(renderedText(simpleText("hello")));
model = f;
```

Program 9

```
// ActiveVRML 1.0 ascii

a=simpleText("hello");
b=textColor(red,a);
f=first(renderedText(b));
model = f over solidColorImage(blue);
```

In program no. 9 , *a* once again represents the text, which in this case is "hello". There is also a function called **textColor**, which asks for a color, either red, green or blue or a combination of these primary colors. *a*, which is your text was white in color by default, *b*, is also text and is of the same data type as *a*, but now it's red in color. So, when you say *f* , *f* will still be the image hello colored red, displayed over **solidColorImage** (which gives you a colored image) which is blue at the moment. Therefore you will get the red image i.e. hello, over a blue background.

Program 10

```
// ActiveVRML 1.0 ascii

c = colorHsl(20,15,40);
a = simpleText("hello");
b = textColor(c,a);
d = renderedText(b);
model = first(d);
```

If you know Windows programming, you'll remember that all colors are composed of the Primary colors, i.e. Red, Green and Blue (RGB), or a mixture of them. Here **colorHsl** will display a combination of red green and blue. By declaring *c* to be a color, we are in effect stating that we will not be utilizing the inbuilt colors of Active VRML which identifies the colors as numbers from 0 - 255. E.g. **c = colorHsl(255,0,0)** will give me red text, where 255 stands for the amount of red.

This program too, can be accommodated on one line.

Program 11

```
// ActiveVRML 1.0 ascii
```

```
f = first(renderedText(textColor(colorHsl(20,15,40),simpleText("hello"))));
model = f;
```

Now Program 12 is actually the first VRML program.

Program 12

```
// ActiveVRML 1.0 ascii

c = colorHsl(time,time*2,time/3);
f = first(renderedText(textColor(c,simpleText("hello"))));
model = f over solidColorImage(blue);
```

In Active VRML, whenever you start looking at the code , you start at the word **model**. Here **model** is equal to *f*. So the code for *f* is executed first, creating an image "hello", with the color *c* . The variable *c* has the time component in it . Now this time component, as we've told you before, keeps changing continuously, and therefore the color too will keep changing indefinitely. So you'll see hello displayed in different colors on the screen. What we are trying to say here, and this is important, is that unless you don't artificially halt the program by pressing ALT+F4, the program does not stop executing. and all the while the value of time keeps changing. This is exactly where VRML scores over C++. Its just like having a infinite for loop in your program. Like the Energizer bunny, the code just keeps going on and on and on..... The major difference here is that you have a variable called *time* which keeps changing infinitely. This is what makes programming in Active VRML so very different from conventional programming.

Now lets look at the next example. Program no. 13.

Program 13

```
// ActiveVRML 1.0 ascii

f = first(import("big.jpg"));
t = first(renderedText(simpleText("hello..")));
g = transformImage(scale2(10),t);
model = g over f;
```

Here we say **model** = *g* over *f*, making *f* the background. But before we attempt to explain the variable *g* , we'll take a quick look at *t*. *t* simply represents the image of the string hello. Next, in *g*, we say **transformImage**. **transformImage** asks for something called a transform which is nothing but a data type which is **scale2** in this program snippet. The variable *t* is image "hello". In *g* we say **scale2(10)** which means, scale the image 10 times. Here the image happens to be a string but it could be small bitmap, in fact, it could be anything at all. The specified image (hello) will now be magnified to 10 times it original size.

Program 14

```
// ActiveVRML 1.0 ascii

f = first(import("big.jpg"));
t = first(renderedText(simpleText("hello..")));
g = transformImage(scale2(time),t);
```

```
model = g over f;
```

In this program, the very same image is being scaled. Earlier we had given a constant number , now we say that the image should be scaled with time as the variable. Because of this the multiplication factor will keep changing, so the image will start small and just keep growing bigger and bigger and bigger........

Program 15

```
// ActiveVRML 1.0 ascii

f = first(import("big.jpg"));
h= first(import("circles.bmp"));
g = transformImage(scale2(10),h);
model = g over f;
```

In Program no. 15 we demonstrate that you can use any image and scale it. You don't have to create the image yourself, you can use any image, saved in any common file format (e.g. .bmp, .gif). Preferably use the samples provided by us.

Program 16

```
// ActiveVRML 1.0 ascii

f = first(import("big.jpg"));
g = transformImage(rotate2(0.5),f);
model = g;
```

In this program we illustrate that it is possible for you to rotate images. We'd like you to keep changing the values in g so that you gain a better understanding of 3D space, which is what VRML is all about anyway.

Program 17

```
// ActiveVRML 1.0 ascii

f = first(import("small.jpg"));
g = crop(point2Xy(0,0),point2Xy(10,10),f);
model = g;
```

This is the first example of real multimedia. Go through the program and don't get discouraged by the many confusing options available. One of these is something called **crop**. **Crop** asks for a starting and an ending coordinate in a form similar to a Point structure, i.e. an x and a y . Here we specify the starting point as 0,0 . In the image small.jpg , the center of the picture is considered to be the coordinates 0,0. In this program the points x and y are both positive and thus comprise the upper left hand corner of the image. So we'll end up with only 1/4 th of the original picture.

Don't lose hope yet !!! This is not as confusing as it first seems. Do the next two programs which elaborate on these concepts.

Program 18

```
// ActiveVRML 1.0 ascii

f = first(import("small.jpg"));
g = crop(point2Xy(-10,-10),point2Xy(0,0),f);
model = g;
```

Program 19

```
// ActiveVRML 1.0 ascii

f = first(import("small.jpg"));
g = crop(point2Xy(-10,-10),point2Xy(10,10),f);
model = g;
```

The reason we specify the points 10,10 is because the picture width and height do not exceed this limit.

Program 20

```
// ActiveVRML 1.0 ascii

f,h = import ("small.jpg");
g = crop(point2Xy(0,0),h,f);
model = g;
```

Now in this program, the **import** statement does not have the function **first**. So in this program, *h* stands for the dimensions of the image., What we failed to mention earlier is that the function **point2Xy** returns a point, a data type which essentially contains two numbers, an x and a y. So when we specify **crop** in program no. 20, we are not giving a very large number for the second half, we are simply specifying the value of *h*. So we see the same image with the same size as before.

In program no.21 we demonstrate what VRML is really all about.

Program 21

```
// ActiveVRML 1.0 ascii

w = 0.001 * time;
f=first(import("small.jpg"));
b=crop(point2Xy(-w,-w),point2Xy(w,w),f);
model=b;
```

Here we create a variable *w* which represents a number that will keep changing. In *w* we keep dividing the value of *time* by 1000. If you're curious enough you can see the value of *w* by simply saying **model** = *w* (Program 22). Next you'll notice that while cropping we have chosen the minimum and the maximum size of the image, by using the plus (+) and the minus (-) signs. When you run this program, the image will starts off by being small then it keeps growing larger and larger.

Program 22

```
// ActiveVRML 1.0 ascii
```

```
w=0.004+0.04*abs(sin(time));
model=w;
```

Program 23

```
// ActiveVRML 1.0 ascii

w=0.004+0.04*abs(sin(time));
f=first(import("small.jpg"));
g=crop(point2Xy(-w,-w),point2Xy(w,w),f);
model=g;
```

Now what happens is that **sin** returns a value, in the range -1 to 1. Now because of this, the value *w* and the image associated with it oscillate. So e.g. 21 and 23 are pretty much the same, the only difference being that in program no. 23, the image starts out small and then grows larger and then shrinks once again, while in 21 the image just keeps getting larger.

Program 24

```
// ActiveVRML 1.0 ascii

f = first(import("small.jpg"));
g = tile(point2Xy(0,0),point2Xy(5,5),f);
model = f;
```

This program in essence demonstrates what **tile** can do

Program 25

```
// ActiveVRML 1.0 ascii

w = 0.001*time;
f = first(import("small.jpg"));
g =tile(point2Xy(0,0),point2Xy(w,w),f);
model = g;
```

Program 25 is the same as 21, but the only difference now is, that instead of using **crop** we are using **tile** and we have the time factor involved in it..

Program 26

```
// ActiveVRML 1.0 ascii

w = 0.001*time;
f = first(import("small.jpg"));
g =tile(point2Xy(-w,-w),point2Xy(w,w),f);
model = g;
```

In Program 25 and Program 26 , we’re using **tile** instead of **crop**, unlike our earlier programs which only used **crop**.

Program 27

```
// ActiveVRML 1.0 ascii

w = 0.004 + 0.04*abs(sin(time));
f = first(import("small.jpg"));
g = tile(point2Xy(-w,-w),point2Xy(w,w),f);
model = g;
```

Program 28

```
// ActiveVRML 1.0 ascii

w = 0.004+0.04*abs(sin(time));
redimg=solidColorImage(red);
f = first(import("small.jpg"));
b=crop(point2Xy(-w,-w),point2Xy(w,w),f);
model = b over redimg;
```

This program is basically the same as the earlier ones, the only difference being that instead of **crop** we're using **tile**. Having a background enhances your animation and here we show how you can go about doing that.

Program 29

```
// ActiveVRML 1.0 ascii

w = 0.004+0.004*abs(sin(time));
redimg = solidColorImage(red);
w2 = w*2;
blueimg = solidColorImage(blue);
cblue = crop(point2Xy(-w,-w),point2Xy(w,w),blueimg);
f=tile(point2Xy(-w2,-w2),point2Xy(w2,w2),cblue);
model = f over redimg;
```

In program 29 we have decided not to deal with ready-made images any more, now we'll create our own images, using combinations of the primary colors . Using **crop** and **tile**, we'll give our image a certain effect. This program has not been written by us, it is taken directly from one of the many downloadable samples that are packaged with Active VRML.

Program 30

```
// ActiveVRML 1.0 ascii

f = first (import("big.jpg"));
tr = translate(0,cos(time *0.3) * 0.01);
g = transformImage(tr,f);
model = g;
```

Whenever you want an image to move, you'll need to give an x coordinate and a y coordinate. So at the moment *f* represents an image (big.jpg) .The variable *tr* is a translate with two numbers in it, an x and a y. One of the things that should be noted is that the first x is equal to 0 , while the second one keeps changing. So now, when I say **transformImage**, what I am actually doing is adding *f* to the values of x and y. But in *tr* we have initialized x to 0 and because only the value of y keeps changing, the image moves vertically. **cos** is just like **sin**, both of which are cyclic in nature .

Program 31

```
// ActiveVRML 1.0 ascii

f = first (import("big.jpg"));
tr = translate(cos(time *0.3) * 0.01,0);
g = transformImage(tr,f);
model = g;
```

Program 32

```
// ActiveVRML 1.0 ascii

f = first (import("big.jpg"));
tr = translate(sin(time) * 0.01, cos(time*0.3) * 0.01);
g = transformImage(tr,f);
model = g;
```

In program no. 31 all that we are doing is keeping the value of y as 0 and changing the value of x. In program no. 32, we have changed both x and y so the image will start moving both horizontally and vertically.

Try out the next program.

Program 33

```
// ActiveVRML 1.0 ascii
w = sin(time)* 0.01;
f = first (import("big.jpg"));
g = transformImage(rotate2(w),f);
model = g;
```

So far all our programs have been rather small and though you're probably excited by this technology, you still haven't experienced it in its entirety. For VRML is not as wonderful as it first seems to be, in fact there are times when good old C/C++ is more appropriate. To demonstrate this assertion of ours, we'll show you a real world example of VRML, a game, and compare it to other applications written in C/C++. You'll realize that you might start to bald prematurely if you try to write some kinds of programs in VRML.

This VRML game is really very simple. There will be one image as the background and another, smaller image moving around in the foreground. The object of the game is to click on the moving image and not on the background. If you do manage to click on the smaller image, you gain a point, if you miss, you lose a point. Your current score will be displayed on the screen. Once you finish the first level, there will be another, tougher level, where a third image, which moves even faster, will appear. If you succeed in clicking on it you get 5 points.

Program 34

```
// ActiveVRML 1.0 ascii

f = first (import("big.jpg"));
s = 10;
t = numberToString(s,0);
g = transformImage(scale2(10),first(renderedText(simpleText(t))));
model = g over f;
```

Program 35

```
// ActiveVRML 1.0 ascii

f = first (import("big.jpg"));
s = 10;
t = numberToString(s,5);
g = transformImage(scale2(10),first(renderedText(simpleText(t))));
model = g over f;
```

In these programs we have tried to display a number on the screen. We cannot directly display a number in VRML, we have to first convert it into an image. So here we have a function called **numberToString** which will convert the number, a variable, into a string. This is one of the many in-built function present in Active VRML. This function takes 2 parameters, the first one is the numeric variable and the second one specifies the number of decimal spaces. So if we state 0 to be the second parameter, we get no decimal places. If we say 5, we get 5 decimal places. Now that we have a string, we follow the earlier steps to convert the string into an image and in no time at all we've got the number displayed on the screen.

Program 36

```
// ActiveVRML 1.0 ascii

f = first(import("big.jpg"));
g = first(import("bubbles.bmp"));
model = f until leftButtonDown => g;
```

As we've mentioned earlier, we would like to handle events (user input). We've noticed that VRML's way of handling events differs quite a bit from the usual way events are handled. In this program we say **model** = *f* which is big.jpg. Then we type in **until**, which is a reserved word . This is one of the reasons we prefer programming environments like VC++ 4.0 which change the color of reserved words, something Active VRML doesn't do, yet. After **until** , we type in **leftButtonDown** which is a reserved word that stands for the left mouse button. The **=>** (equal to greater than) sign is the syntax used with this word. Next we type in *g* which represents the image bubbles.bmp. So when you run this program ,you will see the file big.jpg, and when you click with the left mouse button, you will see bubbles.bmp (*g*).So now we have proof that we can take user input and also change the images on the screen. This program also demonstrates that *g* is only activated when the left mouse button is pressed. Thus **model** is now equal to *g* and not *f*.

Program 37

```
//ActiveVRML 1.0 ascii
```

```
f = first(import("bubbles.bmp"));
g = first( import("big.jpg"));
h = first( import("circles.bmp"));
model = f until leftButtonDown => ( g until leftButtonDown => h);
```

In this program we add to the previous **model** statement and in brackets say, **g until leftButtonDown =>h.** So now before the left mouse button is pressed, the image will be *f*, which is the first image. Now if you click again, the image *g* (big.jpg) will be displayed . If you click again the image *h* (circles.bmp) will be displayed. If you click once again, nothing happens. Thats because your program has run out of images to display.

Please remember that VRML programs don't quit, they don't stop automatically, they just keep on going. It's only when you say end, that they realize that they must stop executing. You'll know that they've stopped because the mouse stops flickering about on the screen and when you quit out, repl tells you that the program terminated normally. Change the line **h = first(import("circles.bmp"));** to **end;** and your program will halt.

Program 38 demonstrates this.

Program 38

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("small.jpg"));
d = f until leftButtonDown =>( g until leftButtonDown => end );
model = d;
```

Program 39

```
//ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first( import("big.jpg"));
d = f until leftButtonDown => ( g until leftButtonDown => d);
model = d;
```

So far, in our programs we have not shown you any instances of recursion i.e. uptil now the programs stop once they run out of images. In Program 39 we say, $d = f$. Since **model** is equal to *d* you get the first image on your screen. If you click with the mouse button, the next image *g* is displayed, when you click again , *g* is replaced with *d* (which stands for the image *f*). When we click on the image *f* , it will change back to *g* and so on. So each time the mouse button is clicked, an image is displayed.

Program 40

```
// ActiveVRML 1.0 ascii

f = first( import("bubbles.bmp"));
g = first( import("big.jpg"));
h = first( import("circles.bmp"));
model = f until keyDown(32) => ( g until keyDown(37) => h);
```

In Program 40, all we have done is replace the **leftButtonDown** with **keyDown** , the value **32** obviously stands for the space bar and **37** stands for the left cursor key. So when you press space or the left cursor key the image changes. Using these reserved words you can now 'capture' the right mouse button, the left mouse button, the keyboard keys or the user's attention.

Program 41

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("big.jpg"));
tr = translate(sin(time)*0.03,cos(time*0.3)*0.03);
h = transformImage(tr,f);
model = h over g;
```

Program 41 may be a repetition to some of you because we have taken one of the Active VRML sample programs and broken it down to simplify it. We have also changed the variable names to make it easier for you. Here *f* is an image (bubbles.bmp), and **translate** has been used to move it. Since **model = h over g**, the bubbles will start moving across the background image *g* (big.jpg). In earlier programs, we had purposely chosen smaller values so that the image moved sluggishly, so you wouldn't have a problem figuring out if the program works or not.

Program 42

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("big.jpg"));
tr = translate(sin(time)*0.03,cos(time*0.3)*0.03);
c=let
        h = transformImage(tr,f);
  in
        h over g;
model = c;
```

Programs 42 and 41 both work exactly in the same way. 42 is written in the fashion Microsoft wants us to write code and 41 is our way of writing code. You've probably noticed that c is equal to a new, reserved word, **let in**. Apparently after the let you can have as many expressions or statements as you like, but in **in** you can have only one. All that Active VRML does is execute the code that appears after **let** and it then puts the result into the statement following **in**. The code will execute as normal and the result will be the image *h* over *g*. *g* is a global variable and has not been changed by us, on the other hand *h* has been changed radically in the **let** statements and it becomes a new image after the transform. In larger programs it's advisable to use **let in** to organize the code.

Compare the workings of both 41 and 42, which give identical results and you'll understand the **let in** concept.

Program 43

```
// ActiveVRML 1.0 ascii

h = first(import("big.jpg"));
newscore(n) = (n) until leftButtonDown=>function(x).newscore(n+1);
score = newscore(0);
s = numberToString(score,0);
f=first(renderedText(simpleText(s)));
g = transformImage(scale2(2),f);
model=g over h;
```

When reading code it's best to start from the bottom and then go up, line by line. In program 43 , the last line is **model = g**. If you look at *g* , it is the image *f* which has been scaled. The variable *f* is the imageform of the string *s*. Now *s* is equal to **numberToString(score,0)** which means that *s* is a number with no decimal spaces, which is held by *score*. *Score* then looks at **newscore** and which is equal to 0 . This is because when we say **newscore(0)** the zero in function **newscore** is given to *n* in the function **newscore(n)** . The answer for **newcore(n)** is *n* so the **newscore(0)** will give 0 as the result. Now if **newscore** was called with 6, the answer will also be 6. After **newscore** , we have **until leftButtonDown** which will not be activated unless the left mouse button is pressed. When the LeftMouseButton is pressed, the function after **functionx** (which is a reserved word) is called. So the function that will be executed will be **newscore(n+1)** (which is the function after **functionx**) . Till now, Active VRML knows that *n* is equal to 0. So when we click with the mouse , the line reads **newscore(0+1)** (i.e. **newscore (n+1)**. Now **newscore(n)** equal to *n*, so **newscore(n+1)** will be equal to *n+1*. This is is the same as saying **newscore(0+1)** = 1. This 1 will be given to *score* and hence you will see 1 displayed on the screen. If you click again with the Left Mouse Button, **newscore** will be called again. But this time, Active VRML remembers that *n* is equal to 1. So **newscore** gets called with the value 2 and the whole process repeats infinitely.

Earlier on **newscore** returned the value 0, now it returns 1. So with every mouse click the *score* will keep increasing. Everything is re-evaluated each time you press the left mouse button and therefore you will see the numbers 0,1,2,3,4,5.... being displayed as images in the background.

Confused ?!, Well, the first time I saw this, I thought I'd stumbled upon a secret document from the X Files !!. The real reason behind this confusion is that the whole process is dynamic.

Remember it is a new way of writing programs but it is worth learning this new way of writing programs

Now if you don't like it like this, you want to write it the right way, that is what 44 does.

Program 44

```
// ActiveVRML 1.0 ascii
h = first(import("big.jpg"));
c =let
        newscore(n) = (n) until leftButtonDown=>function(x).newscore(n+1);
        score = newscore(0);
        s = numberToString(score,0);
        g = transformImage(scale2(2),first(renderedText(simpleText(s))));
   in
        g over h;
```

```
model = c;
```

Program 45

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("big.jpg"));
tr = translate(sin(time) * 0.03, cos(time*0.3) * 0.03);
c =
let
  h = transformImage(tr, f);

  newscore(n) = (n) until leftButtonDown =>function(x).newscore(n + 1);
  score = newscore(0);
  s= numberToString(score, 0);
  t=transformImage(scale2(2),first(renderedText(simpleText(s))));
  in
    (t over h over g);
model = c;
```

In program 45 we say **model = c**. If you've being paying attention you will have noticed that *c* is made up of 3 different images, i.e. *t, h, j* . *t* is what we need to look at first and it is nothing but the score while *h* is nothing but the **transformImage**. What happens now is that the image bubble.bmp starts moving around on the background (big.jpg) and no matter where you click, you will be rewarded with 1 point..

Won't it be a good idea to see if we can modify the program so that if we click in the background, the score increased by 1, and if we click on the bubble, nothing happens. What this means is that now we want to associate events with one image and not another.

Program 46

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("big.jpg"));

a1,a2 = pickable(g, ["background"]);
a3= andEvent(leftButtonDown, a2);

tr = translate(sin(time) * 0.01, cos(time*0.3) * 0.03);
c =
let
        h = transformImage(tr, f);
        newscore(n) = (n) until ((a3 => function(x).newscore(n - 1)));
        score = newscore(0);
        scoreStr = numberToString(score, 0);
        t=transformImage(scale2(2),first(renderedText(simpleText(scoreStr))));
in
        (t over h over a1);
model = c ;
```

This is a program that associates events with an image . What we want to do is capture the leftmouse click which has occurred on the background image (*g*). The word **pickable** is an inbuilt function available in Active VRML to which we give the image *g* and in square brackets, any name (we have called it *background*). The first return value of the **pickable** function is an image which we store in *a1*. *g* and *a1* are both the same image. The only difference is that *g* has no events associated with it whereas *a1* has events allied with it.

The second return value of the function pickable are the events associated with the image and these events are stored in the variable *a2*. As of now it contains no captured events . To add the leftmousebutton event to its existing events we need a function called **andevents**. In the **andevents** function you give the new event you want to add and the existing event list (which is *a2*). *a2* represents an object which knows about the events associated with *a1*. So we have said **andEvents(leftButtonDown , a2)**.

The output of this function is stored in another variable (*a3*)which stands for the left mouse button. So *a3* senses the left mouse button being clicked on the background image (*g*) because now it knows of all the events that have occurred during the course of the program. The rest of the program, remains the same. The only difference is that in the until, instead of saying **leftButtonDown,** we are now saying *a3*. Now in the **in** statement, it is crucial to remember that the third image being talked about is not g , it is a1. So now when you click on the background, the score will reduce by one, if you click on the bubble which has no events associated with it, nothing will happen. This establishes that the mouse click does not pass through the image onto the background.

Program 47

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("big.jpg"));

a1,a2 = pickable(g, ["background"]);
a3 = andEvent(leftButtonDown, a2);

tr = translate(sin(time) * 0.01, cos(time*0.3) * 0.03);
c =
let
        h = transformImage(tr, f);

        b1,b2=pickable(h,["left"]);
        b3 = andEvent(leftButtonDown,b2);

        newscore(n) = (n) until
        (
        (a3 => function(x).newscore(n - 1)) |
        (b3 => function(x).newscore(n + 1))
        );

        score = newscore(0);
        scoreStr = numberToString(score, 0);
        t=transformImage(scale2(2),first(renderedText(simpleText(scoreStr))));
in
        (t over b1 over a1);
model = c ;
```

Here, we go a step further and try to associate an event with the file bubbles.bmp. But our problem now is, that we have to do this in the **let** statements, because in the **let** statement we have used **transformImage** with the variables *tr* and *f*. So *h* now becomes the image which moves around while *f* remains a static image.

So we do the same thing that we did earlier, that is, use **pickable** with *h* and have two variables, one the image, the other the events. We will use **andevents** to add **leftButtonDown** for the bubbles.bmp file and the results will be stored in the variable *b3*.

Now in the until, we need to trap two events, that is the **leftButtonDown** event for the background and the bubbles.. In the first case, you decrease the score by one and in the other case, you increase the score by one. In the **until** statement, the OR sign means either one of the background or foreground events. This statement is like a normal OR condition in C. Now when I click in the bubble image or in the background, the click is sensed by VRML and it executes the relevant statements.

Now lets ask ourselves a very elementary question, could you have ever done such a thing in C/ C++ ? Because if you try to do what we have done uptil now in VRML in C/C++, they'll strap you into a straight-jacket and you'll spend the rest of your days in a nice quiet country manor, learning how to weave straw hats !!!

In C/C++ you'll need to create static background. On top of that background you'll need to put a moving image and you'll also have to figure out how to animate the image, how to make it move across the screen, how to make it respond to user input and how to associate an event with one image and not another. In VRML we only specify the rules as we understand them. We simply tell VRML that we have two images, one is the background and one is supposed to move in such and such manner, that if the user clicks on a specified image the variable's values should change. The one who does the moving and the alterations is Active VRML.

Now the next program has three images with it. Same rules apply.

Program 48

```
// ActiveVRML 1.0 ascii

f = first(import("bubbles.bmp"));
g = first(import("big.jpg"));
e = first (import("circles.bmp"));

a1,a2 = pickable(g, ["background"]);
a3 = andEvent(leftButtonDown, a2);

tr = translate(sin(time) * 0.01, cos(time*0.3) * 0.03);
tr1 = translate(cos(time) * 0.01, sin(time*0.3) * 0.03);
c =
let
        h = transformImage(tr, f);
        i = transformImage(tr1,e);

        b1,b2=pickable(h,["left"]);
        b3 = andEvent(leftButtonDown,b2);

        c1,c2 = pickable(i,["right"]);
        c3 = andEvent(leftButtonDown,c2);
```

```
        newscore(n) = (n) until
        (
        (a3 => function(x).newscore(n - 1)) |
        (b3 => function(x).newscore(n + 1))|
        (c3 => function(x).newscore(n + 5))
        );

        score = newscore(0);
        scoreStr = numberToString(score, 0);
        t=transformImage(scale2(2),first(renderedText(simpleText(scoreStr))));
in
        (t over c1 over b1 over a1);
model = c ;
```

Program 49

```
// ActiveVRML 1.0 ascii

b1, p1 = import("big.jpg");
b2, p2 = import("bubbles.bmp");

xltImage(p, im)=transformImage(translate(xComponent(p), yComponent(p)),im);

p, ev = pickable(b1, []);

s3(pt1, pt2) = xltImage(pt1+(pt2-pt1)*(sin(time)+1.0)/2.0,b2) over p;
s2(t1)=p until andEvent(leftButtonDown,ev)=>function((),(t2,vec)).s3(t1,t2);
model=p until andEvent(leftButtonDown,ev)=>function((),(pt,vec)).s2(pt);
```

Program 50

```
// ActiveVRML 1.0 ascii

b1, p1 = import("big.jpg");
b2, p2 = import("bubbles.bmp");

xltImage(p, im)=transformImage(translate(xComponent(p), yComponent(p)),im);

p, ev = pickable(b1, []);

s3(pt1, pt2) = xltImage(pt1+(pt2-pt1)*(sin(time)+1.0)/2.0,b2) over p
until andEvent(leftButtonDown, ev) => function((), (pt, vec)).s2(pt);
s2(t1)=p until andEvent(leftButtonDown,ev)=>function((),(t2,vec)).s3(t1,t2);

model=p until andEvent(leftButtonDown,ev)=>function((),(pt,vec)).s2(pt);
```

49 and 50 are very simple programs. All that they do is, move the image bubbles.bmp between two points, marked by clicking in to different places on the screen. Here the **model** is *b*, which is nothing but the file big.jpg. Then we say **until andEvent leftButtonDown** to the existing events followed by a function name. What this does is, that the first time the user presses the left mouse button, the function *s2* is called, which in turn calls *s3*. *s3* is where your code is actually executed. which draws an image which moves across two points. *pt1* and *pt2* are the two points, when we click with the mouse. 50 does it recursively. Which is the first time you click , it will do it, you click twice again, it will change it.

Program 51 is all sound. Try it out, you will probably understand it.

Program 51

```
// ActiveVRML 1.0 ASCII

f = first(import("a1.wav"));
g = rate(1.5,f);
model = loop(g);
```

The tutorial on VRML 1.0 and VRML 2.0 (Moving Worlds) speaks more on Virtual Reality Modeling Language or learn more about Microsoft's ActiveX Objects . All your comments and suggestions are welcomed . So please feel free to send us your feedback .

This tutorial is a joint effort of

Mr. Vijay Mukhi
Ms. Sonal Kotecha
Mr. Arsalan Zaidi

Vijay Mukhi's Computer Institute
B-13, Everest Building, Tardeo, Bombay 400 034, India.
http://www.neca.com/~vmis
e-mail: vmukhi@giasbm01.vsnl.net.in