Paper review: A language to model animation out of behavior-embedded graphical components

Shahid Alam

Department of Systems and Computer Engineering Carleton University, Student Member, IEEE

Abstract— Paper reviewed here proposes a scheme for 3D animation. For describing this scheme a language has been implemented, which is nicknamed 'V'. This paper describes entities in two states. Each entity has attributes and by changing the values of these attributes, the behavior of these entities change. These entities can be described as nested components which are structurally and behaviorally complete. The language 'V' is used to define these components in a scene and associates a behavior script with these components. Since 'V' is a procedural language, so it's very easy to describe procedural animation. It provides language constructs to easily build animation of structured scenes having well defined geometry's and algorithm-based distributed behavior. Entire animation can be organized in terms of a hierarchy of independently graphical components. This language can serve as the basis for the development of component based animation system.

Index Terms— Animation, computer language, mechanism, component, modeling

1. INTRODUCTION

This paper was published in the journal of visualization and computer animation in 2002. Name of the authors are Prabir K. Pal and Biswajit Sarkar. Main idea behind this paper is that everything is defined as entities and these entities either animate or inanimate with respect to time. They have certain behaviors which change with respect to time. Behavior or attributes can be represented in parameters. Since these entities are too complex to be represented as one entity. So they are further divided into sub entities, called components. These components are structurally and behaviorally independent and complete.

A scheme has been proposed for 3D animation. This scheme is being described in a language nicknamed ‘V’. This language defines structure, visual attributes and behavior for each component. The behavior attributes can be in the form of programming scripts. These are referred to as embedded behavior. A component when appears in a scene behaves according to the built in embedded behavior. These components can be nested and higher level component can control other nested component through its parameters. Components have there own advantages like reuse, alteration, maintenance, debugging, development etc. Each component has associated with it a behavior. The shape, size, position, color or texture of each component changes over time. This behavior can be controlled by finite number of parameters.

Two examples are described to explain the use of 'V' in modeling animation. One is pedestal fan other is a walking man. The speed of the fan can be controlled by changing the parameter speed. A simple walking man can be described by two parameters, speed and his direction. As a complex geometric model can be built using simple geometric primitives, so a complex scene can be generated by setting the appropriate parameters of each of it’s component. Each component has associated with it a program script. A component’s script can assign parameters to its child component’s script. In this way a hierarchy of components can be built. These components are invoked in a nested manner to build entire animation in steps.

In language ‘V’ algebra like object expressions are used to build animation. They define composite (static) structures that are based on geometric primitives to build objects. And to animate those objects they use mechanism (variable structure), which have attributes that can be changed at run time to produce animation. Mechanisms and composite structures are the main building blocks for modeling animation.

‘V’ is a procedural language i.e it produces procedural animation. The positioning and orientation of the components is with respect to the reference coordinate system. The basic building block for building these components is a composite structure as described above. A complex 3D model can be built using these basic building blocks, in an object expression.

A composite object or a component cannot be changed internally. i.e it’s position and orientation cannot be changed with respect to the local coordinate system. Also it’s shape, size, color cannot be changed dynamically. In a mechanism some of its attributes can be changed dynamically. This is known as degree of freedom. Behavior of mechanism depends on the values assigned to it’s attributes in each successive frame. Embedded or built-in behavior is defined when the program script associated with the mechanism changes these attributes. There can be two kinds of behavior. Fixed behavior
is when program script changes attributes without any external influence. Parametrized behavior is when program script changes attributes with an external influence (arguments of the mechanism).

2. **Overview of Related Efforts**

The following related developments are mentioned in this paper. They are listed here with a very brief introduction. Later language ‘V’ is compared with some of these systems and one other language that is not mentioned in this paper, but is found very similar to ‘V’.

2.1. **ASAS** (1982) - The Actor / Scriptor Animation System developed by Reynolds
   i. An actor is a complex geometric object with its own animation code that is executed once per frame.
   ii. Actors can communicate with each other to synchronize their behavior.

2.2. **Mira Animation System** (1983) – reported by Thalmann et al.
   i. Successfully used to produce short animation movies
   ii. Standard Pascal was extended with special graphical types (figures, actors and camera)
   iii. They contain procedural elements to describe how to draw and animate
   iv. Graphical components of ‘V’ strongly resembles these data types

2.3. **Alice** (1995) – List of action routines executed in each frame of animation, for changing the internal states of the objects.

2.3. **FRAN** (1999) – by Elliot [1]
   i. This is a declarative language
   ii. Well suited for 3D geometry and animation

2.4. **Autonomous components** (2002) – by Huizing and Barenbrug
   i. Each graphical object has an autonomous behavior
   ii. Same concept as in ‘V’, but they delve deep into physical laws of gravity, inertia, collision etc.

   i. X3D is an extension to VRML and adds humanoid animation, NURBS etc.
   ii. It's used for building 3D models on the web
   iii. Needs explicit event routing scheme
   iv. ‘V’ has implicit event routing

   i. Maya embedded language (MEL) for display and animation
   ii. Good for creation and navigation of virtual worlds
   iii. Lack language constructs

   i. Created by Attitude Software, purchased by Adobe in 1999 and discontinued in December 2004
   ii. Animation through code written in JavaScript, which can be attached to individual objects
   iii. No more supported

3. **Modeling in ‘V’**

To explain modeling in ‘V’ an example of a pedestal fan has been selected. The pedestal fan has a motor and three blades mounted on a shaft. Whole thing is mounted on a wide base as shown in figure 1. Following is a list of a script for modeling the animation of fan in ‘V’. We want to spin the blades and also turn the fan.

**COMPOSITE motor**

```plaintext
/{
   OBJECT motorShape;
   TRANSFORMATION orientMotor;
   COLOR lightYellow;

   motorShape = SURFACE_OF_REV ((0,0), (4,0), (13,1),
   (15,5), (15,15), (13,35),
   (12,40), (9,45), (5,50) (0,50));

   orientMotor = RX (90) * TZ(-18);
   lightYellow = RGB (1.0, 1.0, 0.2);
   motor = orientMotor : lightYellow : motorShape;
}
```

**COMPOSITE stand**

```plaintext
/{
   OBJECT base, supportRod;
   TRANSFORMATION placeStand;
   COLOR lightBlue;

   base = SURFACE_OF_REV ((0,0), (40,0), (40,5), (35,5),
   (32.5,15), (25,18) (12.5,24),
   (7.5,27 , (7,30), (0,30));

   supportRod = CYL (5, 150);
   placeStand = TZ (-150);
   lightBlue = RGB (0.4, 0.4, 1.0);

   stand = placeStand : lightBlue : [supportRod + base];
}
```
Figure 1. Animation of pedestal fan that turns while its blades are spinning

COMPOSITE bladeAssembly
{
OBJECT blade, shaft;
TRANSFORMATION orientShaft, orientBade, orientAssembly;
COLOR lightGreen, darkRed;
orientShaft = RY (90) * TZ (-5.5);
orientBlade = RY (-80);
orientAssembly = RZ (90);
lightGreen = RGB (0.2, 1.0, 0.2);
shaft = orientShaft : darkRed : CYL (4,6);
blade = orientBlade : lightGreen : PRISM (1, (2,2),
(14,30), (15,40), (9,50), (0,49),
(-10,45), (-15,35), (-2,2));
//
// 3 copies of blade rotated 120 degrees
//
bladeAssembly = orientAssembly : [shaft + REPEAT (blade, RX (120), 3)];
}

MECHANISM fan (IN theta, IN alpha)
{
TRANSFORMATION placeBladeAssembly, spin, turn;
placeBladeAssembly = TY (24);
spin = RY (theta);
turn = RZ (alpha);
//
// Fan contains a stand and a motor with a blade assembly which rotates
// alpha degrees
//
fan = stand + turn : [motor + placeBladeAssembly : spin : bladeAssembly];
} fan1 (0, -60);

//
// Fan mechanism is instantiated with initial values
// and it animates by changing the values of alpha and theta
//
SCENE fan1;

The first is a COMPOSITE structure of a motor. It has an object motorShape, which is used to define the shape of the motor. The motor has yellow color. RX is used to rotate and TZ is used to translate the motor. Both of them are responsible for rotating and turning the motor respectively. Second is the COMPOSITE structure of the stand. It has two objects base and supportRod. Its color is light blue. Stand is turned using TZ. Third is a COMPOSITE structure of three blade assemblies. The shaft rotates in Y (RY) direction and turns in Z (TZ) direction. Blades rotate in Y (RY) direction. The assembly rotates in Z (RZ) direction. The mechanism is used for describing the behavior of the fan. In this example no behavior script is embedded in the fan's mechanism. The fan MECHANISM takes two IN parameters theta and alpha. Theta and alpha are the angles responsible for spinning the blades and turning the fan respectively. Do we need more control rather than just controlling the two angles for spinning and turning? Fan is of no use if you can't change its speed. Next section gives an example to change the speed of the fan.

3.1. Mechanisms with Embedded Behavior
Here is an example of a behavior script embedded in the mechanism. It just takes one parameter which is fan's speed.

MECHANISM fan (IN speed)
{

//
// Can be changed locally
//
VARIABLE theta = 0, alpha = 0; thetaStep = 20
alphaStep = 0.5, dir = 1;
TRANSFORMATION placeBladeAssembly, spin, turn;

BEHAVIOR-SCRIPT
{
theta = theta + speed * thetaStep;
if (theta > 360) theta = theta – 360;
if (alpha + alphaStep > 60)
dir = -1;
elseif (alpha-alphaStep < -60)
    dir = 1;
    alpha = alpha + dir * alphaStep;
}

placeBladeAssembly = TY (24);
spin = RY (theta);
turn = RZ (alpha);
fan = stand + turn : [motor + placeBladeAssembly : spin : baldeAssembly];
);

Then is the MECHANISM that creates and control three buttons using the above button MECHANISM:

MECHANISM buttonAssembly (OUT speed)
{
    VARIABLE pressed1, pressed2, pressed3;
    Button button1 (pressed1), button2 (pressed2), button3 (pressed3);

    BEHAVIOR-SCRIPT
    {
        if (pressed1 == 1)
        {
            speed = 1.0;
            pressed1 = 0;
        }
        else if (pressed2 == 1)
        {
            speed = 0.5;
            pressed2 = 0;
        }
        else if (pressed3 == 1)
        {
            speed = 0;
            pressed3 = 0;
        }
    }

    buttonAssembly = button1 + TZ (-5.0) : button2 + TZ (-10.0) : button3;
};

Now the button instances are controlling the speed. If you notice there are two different scripts used in above MECHANISMS. The first one is SELECTION-SCRIPT for selecting the button. It has ‘pressed’ as INOUT parameter. Second script is BEHAVIOR-SCRIPT which is reading the buttons and controlling the speed of the fan. Since there are 3 different speeds so we need to instantiate 3 buttons. These buttons are passed three VARIABLE parameters so that these instants of buttons can change these parameters to influence the speed change, which is an OUT parameter of this MECHANISM. These scripts are embedded in MECHANISM fan. So wherever it appears it behaves with this behavior.

There are three classifications of parameters / arguments IN, OUT and INOUT as described below:

3.2. IN parameters
They provide control over the behavior of the mechanism. In a hierarchy of mechanisms a (as the case in language ‘V’) higher level mechanism can change this parameter to affect the behavior of the lower level mechanism.

3.3. OUT parameters
A higher level mechanism can receive notification from lower level mechanism of any special event by using these arguments.

3.4. INOUT parameters
Since this provides both the above functionality's in one, so it let the lower level mechanism to influence the behavior of its higher level mechanism and also the whole environment in general.

4. RIDERS
In computer animation often an object is moved from one frame of reference to another. In assembly lines when an object on the conveyer belt is picked up by a robot as shown in figure 2. It moves from the one reference, conveyer belt to another reference, robot. First it was moving with the conveyer belt, but now it needs to move with the robot. So to achieve this, the object needs to be transferred from one frame of reference to another frame of reference. In ‘V’ this has been achieved by using riders. They act as a placeholder for objects. To further understand following is an example of there use.

RIDER rGrip;

// pellet is a composite object which is placed in the grip of the robot
// as is shown in Figure 2. It will replace the already placed object in the robot grip
rGrip->pellet;

// This will make it release / disappear form the robot grip
rGrip->null;

We can also remove the object from one RIDER and place it on another RIDER as:
Figure 2. Animation of a 4-degrees-of-freedom robot picking up cylindrical (nuclear) fuel pellets from a conveyer and placing them in rows in a tray on another conveyer. Riders are used to transfer a pellet from the conveyer to the robot gripper and finally to the tray.

RIDER can also be passed as an argument IN, OUT and INOUT to a MECHANISM.

Rider as IN is only available to be assigned by the higher level MECHANISM. Rider as OUT is only available to be manipulated inside the MECHANISM that receives it. Rider as INOUT is available to both the MECHANISMS that receive it and the higher level.

5. Case Studies

Four case studies (belt drive, loader, walking machine, group of mobile robots) are described in the paper but only one is described here, because it describes very interesting group animation of robots which can be compared to the behavior of flocking birds.

5.1. Simulation of Collision-Free Navigation of a Group of Mobile Robots

A group of mobile robots are simulated as shown in figure 3. Each of which is chasing the moving target and also avoiding collision with each other and stationery obstacles. The environment provide an array of freeRange[8] and the goal coordinates goalX and goalY. The robots gives there position mrlX and mrlY to the environment. The array freeRange is the value of eight equal angles (45 degree apart) which are the sensor readings of the robot indicating the distance from the nearest obstacle. We can simulate the behavior of flocking birds if these readings are replaced by the distance of neighboring birds.

MECHANISM mRob (IN freeRange[8], INOUT {mrlX, mrlY}, IN {goalX, goalY})

{ BEHAVIOR-SCRIPT

  // Determine next step of motion of this mobile robots using the knowledge of freeRange[] and goal and update mrlX and mrlY accordingly

  BEHAVIOR-SCRIPT

  // Compute freeRange[8] for each robot using
  // obsRad[nObs], obslX[nObs], obslY[nObs], mrlX[nRob] and mrlY[nRob]; Redefine some or all elements of goalX[nRob] and goalY[nRob] if so desired

  BEHAVIOR-SCRIPT

  // Geometric description of environ in terms of mRobot[i] located at mrlX[i], mrlY[i], i = 1 to nRob and obstacles at obs[j], j = 1 to nObs
  
}
Figure 3. A group of mobile robots chasing a moving goal through cylindrical obstacles
The robots and obstacles are placed in the scene according to their initial positions. After that each robot determines its position using the inputs freeRange[8] and goalX and goalY. When it decides its motion step, it also updates its mrlX and mrlY which are passed back to the environ MECHANISM. The environ MECHANISM computes the freeRange[8] for each robot only once using the last values of mrlX and mrlY from the mRob MECHANISM.

It has been claimed in the paper that each robot moves concurrently and independently from other robots. Also if there movement is bounded, then no one robot will bump with another robot. But it's not clear what algorithm they are using to keep robot movement bounded.

It's not specifically mentioned in the paper but this is what we assume about their algorithm for bounding the robots. Since the location (mrlX, mrlY) and size of each robot is known and also the shape of each object is geometrical. So whenever one of the robot's position and size starts to coincide with another robot's size and position, it changes its next motion step such that it avoids collision. Similarly they avoid collision with the obstacles. This is compared with Reynolds [5] behavioral distributed model of flocks. Reynolds emphasized the principle of programming an individual to generate the behavior of a group, in the study of collective motion of flocks, herds and schools.

6. Implementation of ‘V’
The language used in implementing ‘V’ is C with OpenGL [11] library. OpenGL is an open source low level library written for 3D graphics rendering. Operating systems on which ‘V’ has been implemented are IRIX and Windows 9x and has been developed for Silicon Graphics Workstation and PC.

Scripts are supported by an interpreter. For reusability of components global variables are not used. Parameters are passed as arguments of type IN, OUT and INOUT. Components used them to communicate with each other. As higher level components can talk to the lower level components, vice versa and both ways.

A scene is divided into components and as shown in the Figure 4. First the lowest level component is drawn, then next higher level component and this keeps on going until the highest level component is rendered.

Components run as a single program unit and animation runs as a sequential program, because all the graphical components use the same matrix stack (in the context of OpenGL, either Model View or Projection). Events are processed synchronously by all the scripts with the exception of SELECTION-SCRIPT. Table 1 gives a comparison of the graphical components implemented in this system with the software components in general. Table 2 gives a comparison of language ‘V’ with other languages including VRM / X3D, MEL (Maya embedded language) by Maya and AL (Animation language) developed at Ohio State University maintained by Stephen F. May.

Table 1. Comparison of graphical components with software components of component-based systems. Only basic properties are compared here.

<table>
<thead>
<tr>
<th>Graphical Components ‘V’</th>
<th>Software components [10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can't run on different machines</td>
<td>Can run on different machines on the net</td>
</tr>
<tr>
<td>Run as one sequential program, only the scripts (which are non-graphical in nature) can run as separate processes but its recommended to run them as one process on the same machine</td>
<td>Run as separate processes</td>
</tr>
<tr>
<td>Polling for checking for events</td>
<td>Event based programming</td>
</tr>
</tbody>
</table>

7. Comparison with other languages
As compared in Table 2 all the languages have one thing in common and that is the use of scripts to describe animation. It shows that to produce real good animation or to get more control over modeling animation a good animator tool needs to
implement script interpreter beside other visual modeling techniques.

Each language uses some kind of data structure to store and produce animation. The method used in 'V' is different than used in other languages. Animation is stored as a hierarchy of graphical components. So that these components can be used independently and this forms the basis for using this language for component based animation systems. One thing that 'V' strongly lacks is the support for tools and plugins. That is the reason is not being used as a very popular language in the main stream.

Table 2. Comparison of 'V' with other languages

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
<td>SGI, Windows 9x</td>
<td>Platform independent</td>
<td>MAC OS X, Windows, Linux</td>
<td>SGI, Linux</td>
</tr>
<tr>
<td></td>
<td>Implemented in C uses OpenGL</td>
<td>VRML 97 and XML based</td>
<td>Implemented in C++ uses OpenGL</td>
<td>Extension of scheme [8] uses OpenGL</td>
</tr>
<tr>
<td><strong>Type of animation</strong></td>
<td>Procedural animation</td>
<td>Web based modeling and animation, more in the form of presentations</td>
<td>All animations</td>
<td>Procedural and interactive animation</td>
</tr>
<tr>
<td><strong>Script Interpreter</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Data structure for animation</strong></td>
<td>Animation organized as hierarchy of graphical components</td>
<td>Component based in the form of a DAG (direct acyclic graph)</td>
<td>NA</td>
<td>Hierarchical modeling in the form of graphics state (similar to OpenGL)</td>
</tr>
<tr>
<td><strong>Universality in context of modeling &amp; animation</strong></td>
<td>No prove for its use as a general purpose language for all animation</td>
<td>General purpose language</td>
<td>General purpose language but used only with Maya</td>
<td>General purpose language</td>
</tr>
<tr>
<td><strong>Features supported</strong></td>
<td>Geometrical modeling, transformation</td>
<td>Geometrical modeling, lighting, texture mapping, transformation, NURBS</td>
<td>Lot of features</td>
<td>Geometrical modeling, transformation, camera</td>
</tr>
<tr>
<td><strong>Tools support</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Text based authoring tools</td>
<td>Used with Maya</td>
<td>Built in tools for interactions</td>
<td></td>
</tr>
<tr>
<td><strong>Real-time</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Universal rendering</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Runs as one sequential program as one process on one machine</td>
<td>Scene can be rendered out of assets located on network</td>
<td>Also includes the ability to render on an unlimited number of networked machines</td>
<td></td>
</tr>
<tr>
<td><strong>Support for plugins</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
8. CONCLUSION

The paper reviewed proposes and implements a procedural animation language which is nicknamed 'V'. In this language they are using composite structures and mechanisms as building blocks for rendering animation. The algorithms used for controlling the animation / behavior of the graphical components are coded in scripts. These components can be stored and used independently in a separate components library. A scene graph is built out of these components and rendered in a depth first fashion. A behavior script is associated with each animated component. That is also executed when the component is rendered, which determines the sequence of animation.

Following are the some of the pros and cons of using this system for modeling computer animation:

8.1. Pros
i. It is a scripting language following the principle of procedural animation. One can write a procedure to generate any desired behavior of animation.
ii. Procedural animation can be very useful for generating much lifelike motion from relatively little input.
iii. Animation is little different and noteworthy.
iv. It may find use in modeling mechatronic devices (e.g. robots). Although no use case or example has been described for computer games, but it’s claimed that it can be used for computer games.
v. Provide language constructs to easily build animation of structured scenes having well defined geometry's and algorithm-based distributed behavior.
vi. Entire animation can be organized in terms of a hierarchy of independently graphical components.
vii. Serve as the basis for the development of component based animation system.

8.2. Cons
i. Since the future frames are entirely dependent on conditions from previous frames, procedural approaches can suffer from lack of external control over the individual frames.
ii. GUI based interactive animation has not yet been explored and implemented.
iii. Only appropriate for machines and mechanisms.
iv. Have not been tried or appropriate for modeling all kinds of animations.
v. Lack of support for importing external images.
vi. Can’t put too much detail while modeling and rendering a component.

REFERENCES