Object Oriented Animation on the World Wide Web

Patrick P. Cai and Azer Bestavros

Computer Science Department

Boston University

Boston, MA 02215

patcai@csa.bu.edu and best@csa.bu.edu

Abstract

This report describes our attempt to add animation as another data type to be used on the World Wide Web. Our current network infrastructure, the Internet, is incapable of carrying video and audio streams for them to be used on the web for presentation purposes. In contrast, object-oriented animation proves to be efficient in terms of network resource requirements. We defined an animation model to support drawing-based and frame-based animation. We also extended the HyperText Markup Language in order to include this animation mode. BU-NCSA Mosanim, a modified version of the NCSA Mosaic for X(v2.5), is available to demonstrate the concept and potentials of animation in presentations and interactive game playing over the web.

1. Introduction:

The web was originally created as a distributed hypertext system. As it evolves, more data type has become available. However, today's international internetworking infrastructure, the Internet, is incapable of supporting multimedia data types such as video and audio, which require too much network bandwidth. As a result, most web pages are composed of static data types such as text and still images.

As it becomes more commercialized, the demands are more and more to create impressive presentation over the web. Clearly, with our current networking infrastructure, neither video nor audio is considered as a promising candidate to be used for this purpose. In contrast, object-oriented animation requires network bandwidth comparable to the normal still images. With an underlying synchronization mechanism, object-oriented animation can be used for creating powerful presentation and even interactive games over the web.

In this report, we describe our attempt at adapting object-oriented animation to the web. We extended the HyperText Markup Language(HTML) in order to describe objects used for animation. We also made modifications to the NCSA Mosaic for X-Window(v2.5) in order to interpret the new HTML tags and show animation. The result is the BU-NCSA Mosanim(Mosaic with Animation). Mosanim is distributed to the Internet community as a freeware.

The rest of this report is organized as following: in section 2 we'll mention previous works in the area; in section 3 we'll describe our object-oriented animation model; in section 4 we'll discuss our extension for

HTML; in section 5 we'll sketch our implementation with BU-NCSA Mosanim; and in section 6, we'll discuss future works to turn this novel idea into a more usable tool for communications over the Internet.

2. Previous works:

There have been discussions among the web community about defining/choosing a proper graphical composition language for timed-presentations and virtual reality over the net. CGM(Computer Graphics Metafile), ISO/IEC 8632 or ODA(Office Document Architecture Standard), HyTime, MHEG, HyperODA, IMA, OMG and many other formats are being considered. CGM and ODA are inappropriate for our purpose because they do not have the concept of time and thus lack the ability to synchronize presentations. HyTime and some other languages are capable of expressing various relationships between time intervals, such as preceding, following and overlapping. However, none of the above languages explicitly take into consideration of the specific needs of animation.

It must be admitted that it is an easy task to pick a standard. However, it is not our intention to set a standard by presenting our animation model and HTML extension. By implementing a relatively simple model, we'll be able to understand various issues animation brings to the web. Hopefully our effort will provide experimental data when a standard is finally decided.

3. The Object-Oriented Animation Model:

The objectives of our model were the following:

1) to have the ability for describing common graphics objects, such like lines, circles, polygons and pre-scanned images
2) to organize the objects using a hierarchical structure
3) to have a synchronization mechanism so that timed presentation can be achieved
4) to make the object be able to respond to various events so that interaction can be achieved

One underlying assumption is that the network is a scarce resource which should be avoided of using whenever possible. This leads to the basic design of our model: once downloaded, all presentation and interaction should be handled by the web client.

Lines, circles and polygons are the graphics primitives. To have a reasonable graphics expressiveness, pre-scanned images are a good compromise. Pre-scanned images require more network bandwidth and processing power, but they do help make higher quality presentations.

Primitive objects such as lines, circles, polygons and pre-scanned images can be grouped together in a composite object. This hierarchical structure is natural as in the real world objects are composite. For example, a person object is composed of head, body, arms and legs. A composite object maintains a local time-line to synchronize its sub-objects(child-objects). As a result, regardless the action or movements of the person object, its arms and legs always move in the correct direction and sequence. The hierarchical structure helps to define shared attributes for a group of objects. Besides the common time-line, scale and color are two of many other shared attributes.

The hierarchical organization of objects gives rise to synchronization. Each composite object synchronizes its sub-objects. All of the top level objects are contained in a special object defined as the "world". When the world is rendered, it ask all of its sub-objects to render themselves. This process goes recursively, until the primitive objects are reached. When an object is asked to render itself, it receives information about all of the shared attributes, plus its parent's world position and world rotation angle. An object knows its local position and rotation angle within its parent's coordinates. Thus, it is able to figure out its world position and angle to render itself and to pass to its sub-objects.

In order to describe motion, translation specifications and rotation specifications are needed. The translation specification describes how fast and in what direction an object moves horizontally and vertically within its local coordinates. The rotation specification describes how fast an object rotates

within its local coordinates, either clockwise or counter-clockwise. In order to translate the local position into the world position, an object contains a joint point which is in its parent's coordinate. An object joins its parent at the joint point. Each object must keep track of its initial position, initial angle, and initial time. Every time it is rendered, it obtains its current time(either real time or simulated time) and sets its current position and current angle, based on its translation and rotation specifications.

Our animation model is also event-driven. The idea is to make individual objects be aware of certain events and be able to handle them. Event is a broad definition of key presses, mouse clicks and expiration of intended periods in the translation/rotation specifications. To make it more useful for interactive games, it would be possible to send events among objects, so that object collisions may trigger object action or effects, such as explosion. An event list is composed of a list of events and corresponding states to enter if the events occur.

In summary, our object model has the following data structures:

- Object
- Object list
- Line, Circle, Polygon, Pre-scanned and Compose-objects (derived Objects)
- Translation specification
- Rotation specification
- State
- Event
- Event list
- Point
- Point list

These interactive entities relate as such: each object is associated with an initial state, which specifies the movement of the object by translation/rotation specifications. A state is associated with a list of events, the occurrence of which will cause the object to switch to other states. Objects can be graphics primitives or pre-scanned images. Objects can be grouped in an object list to form a composed object. Points and Point lists are used in constructing various interactive entities such as polygons.

An example is to simulate the leg movements when a person walks. Let's first consider the right leg. Since it is local with the whole "person" object, it does not have any translation movements. Initially, it rotates counter-clockwise at the speed of 4 degrees/second. We associate this rotation specification with state 1. After it rotates for 5 seconds, we want it to change direction to rotate clockwise. So we define state 2 with a clockwise rotation specification. And then we must associate with state 1 an event list containing one event -- rotation period expiration, and the corresponding next state is state 2. We should do the same with state 2, but make the next state in the event list to be state 1. After we associate state 1 with object right leg, the leg will swing back and forth with a period of 10 seconds.

4. HTML Extension for Animation:

To make animation a new data type on the web, the web client must be able to interpret the animation definition and build the data structure for the specified animation dynamically. The natural choice of the animation specification is HTML, since all other web data types are specified in this language. However, our current HTML extension can be much improved to make the creation of animation an easier task.

Our current HTML extension relates tightly to our object model. One tag is used for each of the above data structures. An example is shown below. A complete definition of the extension can be found in Appendix B.

To define the movement for the right leg, as mentioned above, the following HTML extension can be used:

```
<EVENT name="rot_EVENT" type=1>
<!-- rotate clockwise at speed 4 (20/5) degrees/second>
```

```
<ROTSPEC NAME="leg_clockwise" mode=1 speed=-4 anglechange=20 period=5>

<!-- define an EVENT LIST for the initial STATE. Since the NEXT STATE is not yet defined, it will be set later.>
<EVENTLIST NAME="leg_clockwise" EVENT="rot_EVENT">

<!-- associate above defined movements with the initial STATE. When an EVENT in the EVENT LIST occurs, the OBJECT
will enter a new STATE.  Translation spec is null -- use default, no translation movements. >
<STATE NAME="leg_clockwise" scale=1 ROTSPEC="leg_clockwise" EVENTLIST="leg_clockwise" color="green">

<!-- **** after a leg has rotated clockwise for 20 degrees, it will enter this STATE to move backwards **** >
<!-- rotate counter-clockwise at speed 4 (20/5) degrees/second>
<ROTSPEC NAME="leg_counterclockwise" mode=1 speed=4 anglechange=-40 period=5>

<EVENTLIST NAME="leg_counterclockwise" EVENT="rot_EVENT">

<!-- associate above defined movements with the this STATE.  When an EVENT in the EVENT LIST occurs, the OBJECT
will enter a new STATE >
<STATE NAME="leg_counterclockwise" scale=1 ROTSPEC="leg_counterclockwise"
EVENTLIST="leg_counterclockwise" color="white">

<!-- Now we have the STATEs defined, we can specify the NEXT STATEs for the EVENTs in the EVENT LISTs>
<EVENTLIST NAME="leg_clockwise" EVENT="rot_EVENT"  NEXTSTATE="leg_counterclockwise">
<EVENTLIST NAME="leg_counterclockwise" EVENT="rot_EVENT" NEXTSTATE="leg_clockwise">
```

The right leg itself is defined as a polygon. The following extension can be used:

```
<!-- *** Build the right leg >
<POINT NAME="pt1" x=-3 y=0>
<POINT NAME="pt2" x=2 y=-0>
<POINT NAME="pt3" x=2 y=20>
<POINT NAME="pt4" x=-3 y=20>

<POINTLIST NAME="pl1" POINT="pt1">
<POINTLIST NAME="pl2" POINT="pt2" NEXTLIST="pl1">
<POINTLIST NAME="pl3" POINT="pt3" NEXTLIST="pl2">
<POINTLIST NAME="pl4" POINT="pt4" NEXTLIST="pl3">

<!-- create OBJECT right leg, which is a polygon with initial STATE "leg_counterclockwise".>
<OBJECT NAME="right_leg" joint="jt" angle=10 rotateorigin="start"   STATE="leg_counterclockwise" type="polygon"
POINTLIST="pl4">
```

When creating animation, the demanding task is not to design the objects, but to define the state changes. It would be helpful to bear in mind a time line, and think clearly how each object would move in each time interval. Then for each and every interval, define corresponding states and link them together by event lists.

In the future, this HTML extension should be simplified to improve usability. In order to synchronize objects, it is preferred to use the following style:

```
<while in next 10 seconds>
<object PERSON moves from TREE to STOP_SIGN>
<object CAR moves from PARKING_LOT to STOP_SIGN>
</while>
```

The above statements create a scene, in which object "PERSON" and "CAR" both move to and arrive at "STOP_SIGN" in 10 seconds. Synchronization is more straight-forward in this fashion.

5. Implementation with the NCSA Mosaic:

NCSA Mosaic is chosen because its source is readily available. The source code is fairly structured, which makes modification relatively simple.

The modification is done in three areas: to make it understand the HTML extension, to build the animation data structure and to render it. The first two areas are straight-forward, because the object model has been clearly thought of and defined. It took a little more effort to get pleasing display results.

Some techniques used include: off-screen drawing to eliminate flickering, timer-callback functions to refresh the screen periodically, setting up a clipmask to display images with transparent colors on a variety of displays.

BU-NCSA Mosanim is freely distributed to the Internet community. Currently, it is known to compile on platforms including SPARC running Solaris, SGI running Irix and DEC Alpha. More information about Mosanim can be found at

"http://cs-www.bu.edu/students/grads/patcai/demo/demo.html".

6. Future work:

The fact that the web is exploding is partly due to its ease of use. The efforts required to create HTML documents are moderate. However, at the current stage, it is not as easy to create an animation object using the above HTML extension. As mentioned earlier, the language for synchronizing objects can be much improved. Other possible improvements of the extension includes:

1) to allow multiple points in a point list
2) to allow multiple objects in an object list
3) to allow multiple events in an event list
4) to allow multiple frames in a frame list
5) to make the language extension more user-friendly, by employing natural language processing techniques

We plan to include additional object types in our animation model, such as audio. The goal is to achieve synchronized multimedia presentations over the Internet.

We also plan to allow user-interaction by making the objects capable of responding to key/mouse events. Currently, the only available event types are period expiration in translation/rotation specifications. To turn Mosanim into a vehicle for interactive games, key and mouse events must be supported.

Needless to say, we'll ensure that Mosanim is compatible with all popular hardware platforms which runs X-Window. If time permits, we might be able to extend our work into the personal computer platforms.

Bibliography

1. Campbell-Grant, I.R. and Robinson P.J., An Introduction To ISO DIS 8613, "Office Document Architecture" and Its Application To Computer Graphics, Computer Graphics Vol 11 No 4 pp 325-241, 1987.
2. Erfle, R., "HyTime as the Multimedia Document Model of Choice", 0-8186-5530-5/94.
3. Balaguer, J.F. and Gobbetti, E., "Special Issue on Real World Virtual Environments", Center for Advanced Studies, Research and Development in Sardinia, Italy.
4. Turner, Gobbetti, R., E., Balaguer, J.F., Mangili, A., Thalmann D. and Thalmann N.M., "An Object-Oriented Methodology Using Dynamic Variables for Animation and Scientific Visualization".
5. Gobbetti E., Balaguer J.F., and Thalmann, D., "An Architecture for Interaction In Synthetic Worlds", Computer Graphics Laboratory, Swiss Federal Institute of Technology.
6. Gobbetti E., Balaguer J.F. and Mangili A. and Turner R., "Building An Interactive 3D Animation System".

Appendix A -- Object-Oriented Animation Specification

Intractive Entities
- Objects & Object Lists
- States
- Events & Event Lists

The Object Class
- Data Members:
    - Parent
    - Local Time
    - State
    - Elapsed Time since the current state is entered
    - Initial Joint Position
    - Current Joint Position
    - Initial Angle
    - Current Angle
- Methods:
    - render -- virtual
    - move -- virtual
    - check_events
    - change_state
    - draw_line
    - rotate_point
- Derived Classes
    - line
    - circle
    - polygon
    - composed object
    - pre-scanned images

The State Class
- Data Members
    - scale
    - color
    - visible
    - initial time (according to parent object's clock)
    - translation specification
    - rotation specification

Translation Specification
- Data Members
    - mode
    - intended speed vector
    - intended period
    - intended position change

Rotation Specification
- Data Members
    - mode
    - intended speed (angular velocity)

    - intended period
    - intended angle change

The Event Class

- type
    - translation period exhausted
    - rotation period exhausted
    - mouse action
    - key press
    - object action (an object in certain state)
- value

Event List

- pointer to event
- pointer to next state if this event occurs
- pointer to next event in the list

Appendix B -- HTML Extensions used in Mosanim

<animation width=w height=h src=s align=a refresh=r mode=m > -- beginning of animation transcript
- width, height: dimension of the animation window
- src: background image (same as in < img >)
- align: text alignment (same as in < img >)
- refresh: animation window refresh rate. Default is 100.
- mode: mode of animation
    - 0: REAL_TIME_MODE: use "gettimeofday", however, when system is heavily loaded, the motion of an object would become jumpy.
    - 1: SIMULATED_TIME_MODE: use simulated time, which is incremented for 10ks each time the animation window is refreshed.

</animation > -- end of animation transcipt

<point name=n x=xx y=yy> -- set up a point
- name: name of the point to be referred by other statements
- x, y: x and y position. The coordinates depend on the context. Default is (0,0).

<pointlist name=n point=pt nextlist > -- set up a point list
- name: name of the point list to be referred later
- point: name of the point contained in this list. Default is (0,0).
- nextlist: next cell in the linked point list structure. Default is null.

<event name=n type=t value=v > -- set up an event
- name: name of the event to be referred later
- type: type of the event. Default is 1.
    - 0: over_trans_time, the period specified in translation specification has expired.
    - 1: over_rot_time, the period specified in translation specification has expired.
    - key_press(2), mouse_action(3) and object_action(4) are not yet supported
- value: value of the event. Currently not used.

<eventlist name=n event=e nextstate=ns nextlist=nl > -- set up an event list
- name: name of the eventlist to be referred by other statements
- event: the event whose occurrence will trigger the object to enter a new state. Must be the name of a defined event. Default to null.
- nextstate: the new state to enter if the event occurs. Must be the name of a previously defined state. Default is a static state (no translation or rotation).
- nextlist: the next (event, nextstate) pair. (A state may be associated with more than one events.) Must be null or a defined event list. Default is null.

<transspec name=n mode=m speed=sp poschange=pc period=pd > -- set up a translation specification
- name: name of the eventlist to be referred by other statements
- mode: since the intended speed, intended position change and intended period are inter-dependent, mode defines two of the three are independent variables. Default is 1.
    - 0: speed_time_mode, poschange = speed x time
    - 1: position_time_mode, speed = poschange / time
    - 2: position_speed_mode, period = poschange / speed
- speed: intended speed vector (x, y) which must be defined as a point. Default is (0,0).
- poschange: intended position change, which must be defined as a point. Default is (0,0).
- period: intended period in seconds. Default is a 999999. (integer)

<rotspec name=n mode=m speed=sp anglechange=ac period=pd > -- set up a rotation specification

- name: name of the eventlist to be referred by other statements
- mode: since intended speed, intended position change and intended period are inter-dependent, mode defines which two of the three are independent variables. Default is 1.
    - 0: speed_time_mode, poschange = speed x time
    - 1: position_time_mode, speed = poschange / time
    - 2: position_speed_mode, period = poschange / speed
- speed: intended speed in pixels/second. Default is 0. (integer)
- anglechange: intended angle change in degrees. Default is 0. (integer)
- period: intended period in seconds. Default is a 999999. (integer)

<state name=n transspec=ts rotspec=rs scale=sc color=c visible=v eventlist=el > -- set up a state

- name: name of the state
- transspec: name of a translation specification to be associated with this state. Default is no translation/
- rotspec: name of a rotation specification to be associated with this state. Default is no rotation.
- scale: for drawing-based objects only. If the object in this state has sub-objects, each sub-objects will be shown as "scale" times their original size. Default is 1. (integer)
- color: name of the color of the object. It only applies to simple drawing objects such as line, circle and polygon. Default is "black".
- visible: set to 0 to hide the object and its sub-objects. 1 means visible. Default is 1.
- eventlist: name of an event list to be associated with this state. When any event contained in the event list occurs, the object associated with this state would enter the coresponding next state. Default is null.

<object name=n joint=jt angle rotateorigin=o state=st type=tp [start=st end=e] [center=c radius=r] [pointlist=pl] [objectlist=ol] [framelist=fl] >

- name: name of the object
- joint: specify where the object joins its parent object. Must be a point in parent object's coordinates. The origin of the parent's coordinates is the parent's joint point. Default is (0,0).
- angle: initial angle. Default is 0.
- rotateorigin: origin of rotation, in parent's coordinates.
- state: name of associated state
- type: type of the object
    - "line": line object
    - "circle": filled circle
    - "polygon": filled polygon
    - "compobj": object that contain one or more sub-objects
    - "prescan": object that contains a sequence of pre-scanned images to be used for motion
- start, end: start and end points. For line objects only. Default to (0,0) and (0,0).
- center, radius: center and radius of the circle (only). Default to (0,0) and 10.
- pointlist: a point list for specifying a polygon. The last point will be automatically connected to the first. Default to null.
- objectlist: a list of subobjects(for "compobj" type only). Default to null.
- framelist: a list of frames(individually scanned images) for pre-scanned objects. Default is null.

<objectlist name=n object=o nextlist=n > -- set up an object list

- name: name of the object list
- object: name of the object in the list. Default is null.
- nextlist: next record in the linked object list. Default is null.

<frame name=n src=s >

- name: name of the frame
- src: url for an image. (as defined in < img >) Defult to null.

<framelist name=n frame=f nextlist >

- name: name of the frame list
- frame: name of the frame in the list. Default is null.
- nextlist: name of the next record in the linked frame list. Default is null.