# A Classification of File Formats for Animation

Gene S. Lee
Department of Computer Science
University of British Columbia
Vancouver, BC Canada V6T 1Z4
gslee@cs.ubc.ca ! gslee@alum.mit.edu

**Abstract**

This paper proposes a classification of existing *file formats* for computer animation. File formats for computer animation store and describe the evolution of virtual scenes over time. They permit applications, which produce animating scenes, to share and to exchange information. File Formats are classified by the information they store and by what means they store the information.

## 1 Introduction

File formats establish specifications to store information onto data files. The information is arranged methodologically for development and easy retrieval. File formats for animation describe the contents or results of a time-varying scene. Computers playback the animations which the file formats describe. A time-varying scene consists of many scene elements that change state over time. Scene elements represent visible entities, such as shapes and figures, and invisible entities, such as behavioral models and temporal strategies. Each is equally important to control and to specify scene changes. The specification for a file format for animation supports data structures and rules to describe the contents of the scene and to communicate the interplay between scene elements that produce scene changes.

Many existing file formats for computer graphics represent only the static elements of a scene. These formats, known as *modeling* formats, primarily store geometric data. Many modeling formats, such as VRML[4], 3DMF[2], IGES[11], and DXF[3], are widely employed by many applications in computer graphics, computer-aided design, and computer simulation. These popular formats promote extensive sharing and reuse of information. Applications adhere to the specifications of modeling formats to read and to write scene information, but employ their own means and data structures to visualize and to manipulate the scene information.

There are few file formats specifically designed for computer animation. Of those that exist, almost none are popular and few support a specification that can describe a wide variety of scenes. Popular formats for animation, such as Quicktime[9] and MPEG[7], are common, but serve limited use. They simply store and display the resultant frames of an animation. The formats permit sharing and reuse of only the frames, not the elements of the scenes that define the animation. Most applications for animation design their own file formats that support only their data structures and methods for animation. To exchange information with other applications, they must maintain individual translators for each file format that other applications support.

It is the aim of this paper to present a classification for existing file format for animation. The classification identifies major format types and describes the strengths and weaknesses of each.

This paper consists of five sections. Section two provides an overview of the class hierarchy that classifies file formats for animation into types. Sections three and four describe the basic design of each type. Each section identifies a separate category of file formats and cites various examples. Citations reference "real" formats or graphics systems with infrastructures that readily support a particular file format type.[1] Section five compares file formats types that describe the contents of animated scene. Types are compared by their design and by their features. Section six..

[0] *Presented at the Western Symposium on Computer Graphics, April 1998, Whistler, Canada*

[1] Most graphics systems developed by academics do not describe publicly the file format they employ to save information. Therefore, their file formats are presumed to be a simple extrapolation of their design.

## 2 An Overview

File formats for animation separate into two major types: *frame* formats and *content* formats. Frame formats store images of scenes while content formats store descriptions of scenes. Each type further decomposes into several subtypes. Collectively, all these types form a class hierarchy, as shown in Figure 1,

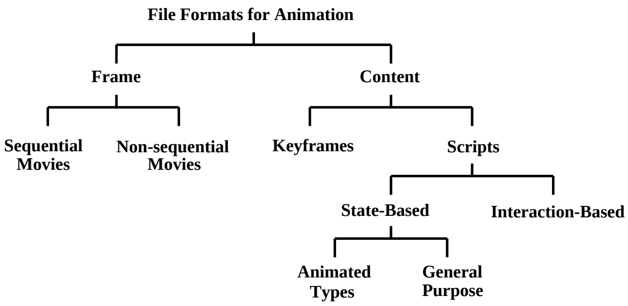


Figure 1: A Classification of File Formats for Animation

Frame formats divide into two subtypes, *sequential movies* and *non-sequential movies*. The former stores a linear progression of images while the latter stores a collage of images. Similarly, content formats partition into two differing subtypes: *keyframes* and *scripts*. Keyframes stores sequences of key values that establish state values while scripts stores sequences of expressions that evaluate to state values.

Scripts further partition into *state-based* and *interaction-based* formats. State-based formats, consisting of *animated types* and *general purpose* formats, explicitly manipulate the states of scene elements. Expressions intermix the parameters of scene elements with special operators to manage the evolution of state values. Interaction-based formats identify relationships between scene elements. Related scene elements change state as they exchange information.

## 3 Frames

Frame-based file formats for animation are common, and many are popular. They are ideal for storing images of great complexity. Many applications exist to read, write, and convert existing movies formats.

### 3.1 Sequential Movies

Sequential movie formats, such as MPEG[7], AVI, and Quicktime[9], compress sequences of frames to form a linear playback of animated scenes. They take advantage of scene coherence between animated frames to decrease storage costs without sacrificing much fidelity. Sequential movies are handy to use as storage devices for successive images, but impractical for use as a basic storage device for computer animation. They store only frames and support few editing operations which can modify and describe the contents of a frame. It is impossible to share and reuse the elements which the frames visualize.

### 3.2 Non-sequential Movies

Non-sequential movie formats, such as Quicktime-VR[5] and Lightfields[14], store sequences of images for non-linear playback of scene flybys. During (interactive) playback, a compositing process blends multiple images into one to feign the appearance of seamless camera movements. Non-sequential formats are useful for rapid playback of complex scenes that animate only a camera. Like sequential movie format, they are impractical for use as a basic storage device for computer animation. The images store content information for only a limited class for animated scenes.

## 4 Content

Content-based file formats describe the contents and behaviors of animated scenes. They store information that is easy to playback and relatively easy to modify. Modifications to scene elements produce new animations. Content-based file formats are expressive, but not widely accepted. Unlike frame-based formats, there are no standard content-based formats. Almost every application in industry and in academia that performs animation supports a different file format that is either based on *keyframes* or *scripts*. With each type, there exist many different ways to store and to organize information.

The graphics community is still searching for the best data structures and design requirements for content-based formats. Ideally, a file format that permits a clean separation between the elements that produce data changes, such as the "who", "what", and "when", is probably best[13]. Who indicates what value are changing; what indicates how values change; and when indicates what conditions are creating change.

### 4.1 Keyframe Formats

Keyframe formats, such as SDL[1] (see Figure 2), consist of "time,value" pairs that describe the evolution of data values over time. The keyframe pairs are low-level abstractions that perform simple state changes. Keyframe formats are ideal for explicitly guiding an animation from one state to the next.

Unfortunately, keyframes formats are inefficient and not suitable for all classes of animations. Keyframe files grow large when specifying scenes of

```
DEFINITION /* ALIAS SDL File Version 8.2 */

/* keyframe curve */
pc plane.X_Translate ( in = PRE_CONSTANT, ...,
   cvs = ( pv( 0.0,0.0, ... ), pv( 20.0,2.0, ... )));

/* define geometry */
patch plane ( ... )

MODEL
{
   trn(animate( plane.X_Translate, ... ), .. )
   inst plane();
}
```

Figure 2: An SDL file that translate a "plane" via two keyframes.

```
/* CORY Script File */

scene_l := scene {
...
cues = ( cue1, ... ) };

cue1 := cue {
   start = 5.0;
   duration = 20.0;
   start_action = ( ... )
   tick_action = ( HAT position = ... )
   end_action = ( ... )
};
```

Figure 3: A CORY file that animates HAT's position with an animated type.

great complexity. Every state change must be handled individually because keyframe formats do not support readily higher level abstractions to build hierarchies or to coordinate multiple state changes. Keyframe files can not specify or control adaptive motions[20] or reactive behaviors[6]. They do not accept relations - a highly desirable operation for the manipulation of detail - between sequences of related state changes. Keyframes assign a fixed value and time to every scene parameter within an animation. It is practically impossible to devise an animation which reacts to run-time conditions, such as unpredictable collisions between scene objects.

### 4.2 Script Formats

Script formats resemble high-level programming languages. They employ rules and syntax to combine instructions and control structures. They form expressions which special evaluators interpret to change the states of scene elements over time. *State-based* scripts explicitly control the state changes, while *interaction-based* scripts produce interactions among scene elements that lead to the state changes. Both types can describe animations that adapt or react to run-time conditions, and both types interact well with simulators. Simulators employ numerical methods to (precisely) calculate scene changes. However, both script types are the most difficult of animation formats to develop for widespread use. Notations are either too restrictive or too general for many users.

#### 4.2.1 State-based Formats

The expressions of state-based file formats update state variables directly. *Type-based* formats embed time-varying variables, called *animated types*, into expressions that control or represent state values. As the animated types change state, the expressions follow suit. *General purpose* formats embed state variables into expressions. State changes occur are evaluators interpret the expressions.

```
/* OBLIQ-3D Script File */

let Slowly = proc (endpoint, s, d)
   let m = meth (self, startpoint, reltime )
   if d is 0.0 then .. else ... end
end;
...
let anim = PointProp_NewSync ( ... );
root.add( SphereGO_New( anim, ... ) );
anim.getBeh().addRequest( Slowly(...) );
```

Figure 4: An OBLIQ-3D file animates a sphere slowly.

##### 4.2.1.1 Type-Based

Type-based file formats, such as Swish[10], Hash[8], and Cory[15] (see Figure 3), parameterize collections of animated types. The parameter values decide how the animated types behave and what values they attain. Type-based formats are best for rapid development. Many provide animated types which control animation at a high level of abstraction. Scenes with complex motions, such as walking, are describable with only a few parameter values. However, similar to keyframe formats, type-based formats are not ideal for managing the growth of complex scenes. The formats provide few means to incorporate hierarchical structures or to coordinate the actions of many animated types.

##### 4.2.1.2 General Purpose

General purpose formats, such as Obliq-3D[16] (see Figure 4), intermix high level abstractions with imperative or declarative commands to solve the problem of scene description for animation. They, like general

```
#VRML V2.0 utf8

DEF myClr ColorInterpolator {
   key       [ 0.0, 0.5, 1.0 ]
   keyValue [ 1 0 0, 0 1 0, 0 0 1 ] # red, green, blue
}

DEF myClock TimeSensor {
   cycleInterval 10.0 # 10 second animation
   ...
}

Shape {
   appearance Appearance {
      material DEF myMatrl Material { } }
   geometry Sphere { }
}

ROUTE myClock.fraction_changed TO myClr.set_fraction
ROUTE myClr.value_changed TO myMatrl.set_diffuseColor
```

Figure 5: A VRML file that animates the color of a sphere via connections called ROUTES.

purpose programming languages, provide users with great expressive capabilities. However, they often lack rules or guidelines to help users systematically build and organize useful data files. It can be difficult to partition data files into parts and to extract only those parts that are useful. In addition, users can produce varying data files for the same animation.

### 4.2.2 Interaction-based Formats

Interaction-based file formats, such as VRML[4] (shown in Figure 5), ASAS[17], and UGA[19], consist of many elements that represent state and some elements that facilitate communication. The latter elements instruct the former elements to communicate. As communications occur, elements react by changing state. Interaction-based file formats generally produce longer data files than all other formats. All other formats neither handle nor permit controls to moderate communications. They support elements and/or expressions that define and manage communications implicitly.

Similar to other formats, existing interaction-based formats suffer great problems when scene complexity grows. There are few means to moderate or coordinate great numbers of interactions. As shown in Figure 6, the topology of a system becomes very dense and difficult to decipher when there are many concurrently interacting elements. Further complication arise when many interactions are changing dynamically.

However, as noted by Lee[12] and Sullivan[18], software systems, especially dynamic ones, are relatively easy to reuse and to manage when separate elements,

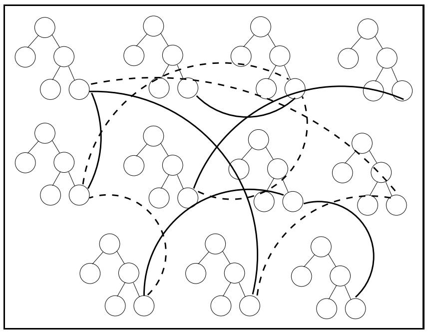

Figure 6: Topology of many interacting elements

not implicit mechanisms control communications between elements. Separate elements are easier to identify, update, and extract. These benefits are essential for managing scenes of great complexity.

## 5 A Comparison

### 5.1 An Illustrative Example

Figure 7 illustrates the difference between the four content-based file formats for animation with four segments of fictitious VRML-like code. The code manages the states of two variables, `key` and `trg`. `Trg` shadows `key` which is set to the sequence `1,2,3`. When `key` changes, `trg` changes. In Figure 7a, keyframes are explicitly assigned to both variables. In Figure 7b, `key` is an animated type with fixed parameter values. `Trg` is key to equal `key` In Figure 7c, an expression inside a loop sets the values of both `trg` and `key`. In Figure 7d, a ROUTE connects `key` to `trg`. The connection propagates changes from `key` to `trg`.

### 5.2 Brief Summary

| | Movies | Keyframes | Scripts |
|---|---|---|---|
| **Foundation** | Frame | Keyframe | Expression |
| **Abstractions** | Low | Low | High |
| **Ease of Use** | Easy | Moderate | Moderate |
| **Complex Mngmnt** | NA | None | Poor |
| **Acceptance** | Wide | Limited | None |
| **Parameters** | Few | Some | Many |

Table 1: Comparison of Existing Formats Animation

Table 5.2 briefly summarizes the uses and capabilities of the three basic types of file formats for animation. Movies files, the most popular of the three, are easy to build and store scenes of unlimited complexity. Keyframe files are popular, but not standardized. They are moderately easy to create and to understand, but arduous to employ when specifying scenes of great

| Keyframe | Script: Type-Based | Script: General Purpose | Script: Interaction |
|---|---|---|---|
| DEF Foo NODE {<br>INT key.keyFs = {1,2,3}<br>}<br><br>DEF Bar NODE {<br>INT trg.keyFs = {1,2,3}<br>} | DEF Foo NODE {<br>ANIM-INT key<br>key.start = 1<br>key.end = 3<br>}<br><br>DEF Bar NODE {<br>INT trg = Foo.key<br>} | DEF Foo NODE {<br>INT32 key<br>}<br><br>DEF Bar NODE {<br>INT32 trg<br>}<br><br>BODY {<br>LOOP T=1 TO 3<br>trg = key = T<br>} | DEF Foo ScalarInterpolator {<br>eventOut Int key<br>Int keyValue = {1,2,3}<br>}<br><br>DEF Bar NODE {<br>eventIn Int32 trg<br>}<br><br>ROUTE Foo.key TO Bar.trg |
| **(A)** | **(B)** | **(C)** | **(D)** |

Figure 7: Four different representations of the same animation

complexity. State-based script files employ expressive means to specify the contents of an animation. The establish rules and syntax for coding, but lack rules and patterns for program development (and reuse). User employ their own means to structure the overall design of the script.

## 6 Conclusion

This paper has presented a classification of existing file formats for animation. Of the two major types, frame-based formats are useful only to playback animations or interact with them, but not to modify them. They visualize, not describe the contents of a scene. Content-based animations consists of four basic types, all of which suffer when scene complexity grows. Existing file formats must provide means to coordinate and collate their basic abstractions if they plan to gain widespread acceptance. Current research suggests that a logical scheme to manage complexity for interaction-based formats exists. These results will appear in a future report.

## References


[1] Alias!WaveFront, Inc. *Scene Description Language (SDL) Reference Manual*, 1997.

[2] Apple Computer Inc. *3D Graphics Programming With QuickDraw 3D*. Addison-Wesley, Reading, MA, 1995.

[3] AutoDesk, I. *Drawing Exchange Format (DXF)*, autocad release 1.0 ed., 1982.

[4] Bell, G., Carey, R., and Marrin, C. *VRML 2.0 Specification*, version 2.0 ed., 1996. ISO/IEC CD 14772.

[5] Chen, S. E. QuickTime VR — An Image-Based Approach to Virtual Environment Navigation. In *Computer Graphics* (Aug. 1995), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 29–38.

[6] Elliott, C., and Hudak, P. Functional Reactive Animation. In *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming* (Amsterdam, The Netherlands, June 1997), pp. 263–273.

[7] Gall, D. L. MPEG: A Video Compress Standard for Multimedia Applications. *Communications of the ACM 34*, 4 (1991), 46–58.

[8] Hash Animation Master. URL: www.hash.com.

[9] Hone, R. *QuickTime*, 2nd ed. Prima Publishing, Sept 1994.

[10] Hopwood, S., Sinclair, I., and Suffern, K. G. A Language for Animating Scene Descriptions. In *Creating and Animating the Virtual World (Computer Animation '92, Fourth Workshop on Computer Animation)* (May 1992), N. M. Thalmann and D. Thalmann, Eds., Springer-Verlag Tokyo.

[11] IGES/PDES Organization. *Initial Graphic Exchange Specification (IGES)*, version 5.3 ed., 1996. ANS US PRO/IPO-100-1996.

[12] Lee, G. S. Reusable Interactions for Animation. In *5th International Conference of Software Reuse* (Victoria, Canada, 1998), IEEE Computer.

[13] Lee, G. S., Forsey, D., and Pai, D. RASP - A Visual Simulation Platform. *Submitted for publication to IEEE Transactions on Visualization and Computer Graphics* (1997).

[14] Levoy, M., and Hanrahan, P. Light Field Rendering. In *Computer Graphics* (Aug. 1996), H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 31–42.

[15] McLachlan, D. R. CORY: An Animation Scripting System. Tech. Rep. 85006, Rensselaer Polytechnic Institute, 1985. Master's Thesis, Rensselaer Design Research Center.

[16] Najork, M. A., and Brown, M. H. Obliq-3D: A High-Level, Fast-Turnaround 3D Animation System. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June 1995), 175–193.

[17] Reynolds, C. Computer Animation with Scripts and Actors. In *Computer Graphics* (July 1982), ACM SIGGRAPH, pp. 289–296.

[18] Sullivan, K. J. *Mediators: Easing the Design and Evolution of Integrated Systems*. PhD thesis, Univ. of Washington, 1994. Tech Report 94-08-01.

[19] Zeleznik, R. C., Conner, D. B., Wloka, M. M., Aliaga, D. G., Huang, N. T., Hubbard, P. M., Knep, B., Kaufman, H., Hughes, J. F., and van Dam, A. An Object-Oriented Framework for the Integration of Interactive Animation Techniques. In *Computer Graphics* (1991), ACM SIGGRAPH, pp. 105–111.

[20] Zeltzer, D. Towards An Integrated View of 3-D Computer Character Animation. In *Graphics Interface '85* (1985), Graphics Interface, pp. 105–115.