An Object Oriented Design of a Graphics Animation System

William Lorensen
Graphics Engineer
Information System Operation
Corporate Research and Development
Bldg KW Room D209A
8*235-8995

Abstract

The software literature has many papers expounding the benefits of object oriented systems for developing complex software systems. Some feel that object oriented programming will be in the 80's what structured programming was in the 70's. This paper investigates the application of the object oriented philosophy to the design of a 3D computer graphics animation system currently under development at the General Electric Corporate Research and Development Center. After a general description of object oriented systems, the paper offers a design methodology which can be used regardless of the object oriented capabilities of the implementation language. This methodology is used to design the animation system itself. The proposed system will provide an animation script interface to a suite of analysis, modelling, rendering, display and filming software and a sequence database to track the progress of the animation through its various stages. The animation system will produce high quality film and video animation of industrial applications such as robotics, structural analysis, molecular modelling and factory simulation. The system is targeted for users who are familiar with their own analysis but not with the intricacies of computer graphics displays and software. A flexible system is proposed which allows the integration of new analysis, modelling and rendering software.

1. Introduction.

Two areas of Computer Science and Computer Graphics are receiving considerable attention in recent literature. Computer Scientists are touting the benefits of object oriented systems, promising benefits in system design that will surpass those obtained using structured programming concepts. Computer Graphics researchers are actively pursuing the goal of realism in the production of high quality, three dimensional animation systems. In following sections, this paper will first review past work in object oriented systems and 3D animation. Then, a design methodology is presented which applies the object oriented philosophy to a 3D animation system currently being designed at the General Electric Corporate Research and Development Center. Functional requirements for the animation system are defined followed by a high level system design and a detailed design of one of the major modules of the system. Finally, an implementation is proposed using C, a portable systems/application language.

2. Object Oriented Systems.

This section describes the characteristics of object oriented systems and then reviews some systems which use these concepts.

This paper was originally presented at the Fifth General Electric Software Engineering Conference, April 30-May 3, 1984, Daytona Beach, Florida.

2.1 Characteristics of Object Oriented Systems.

Object oriented systems rely heavily on the software engineering concepts of

1. Information hiding: Details of a system which do not affect other parts of the system are not visible from the outside.
2. Abstraction: Entities of a system are grouped according to common properties and operations.
3. Modularization: Parts of a system which have localized behavior are grouped together with well defined interfaces.

In the context of object oriented systems, these three principles complement rather than compete with each other. No compromises are required to apply all three to their limit.

Object oriented systems are characterized by abstract constructs called objects which contain data and procedures to manipulate that data. The data describe the local state of the object and are only accessible to the outside world through the object's procedures. These procedures are called methodsand are activated when the object receives a message. Messages are the only means by which objects can communicate with each other and provide a uniform mechanism for inter-object communication. An object is created by making a copy or insranceof a particular class of objects. Classes are the abstractions of these systems. Not only do they define the data structures associated with the class but also the methods for manipulating the class. When an object is created by instancing a class, the object becomes the data and the methods to manipulate the data. New classes can be created by sharing the description of another class and modifying or adding to it with data and methods. This mechanism is called ***inheritance.***

A final important point: only the objects know about their data structures and methods. Nothing outside the object can directly access these structures. If a new algortihm is required to perform a task, the data structure may require change but only the object itself is effected by this change. This Property applies the software engineering notions of information hiding and modularity to data as well as procedures.

2.2 Previous Work.

The first object oriented system referenced by most papers is Ivan Sutherland's PhD thesis [1]. This was a general purpose graphics system for interactive creation and editing of pictures on a graphics display. Geometric transformations were applied to master (class) definitions of objects resulting in an instance of the geometric object. Although the concept of an object oriented system was not defined in 1963, Sutherland's user-interface had many properties in common with such systems.

The Smalltalk effort [2] at Xerox's Palo Alto Research Center (PARC) is the most familiar object oriented system. This system uses the concepts of objects, messages and classes to produce a programming system and a user interface. It differs from other object oriented systems in that it does not have any conventional typing and procedure constructs which might violate the rigorous application of objects and message passing. The only construct in the system is an object. Even program flow is controlled with this construct. Each class has a number of methods it uses to process messages. All processing takes place inside the objects. New classes can be defined by adding data and methods to other classes called super classes. When a message is received by a Smalltalk object and an associated method is found, it is executed; otherwise, the message is passed to the object's super class. This process proceeds until the message is either recognized or rejected. This hierarchical inheritance property allows the Smalltalk system to rely heavily on previous software and to build incremental systems without substantial software developmen!. The most familiar objects in Smalltalk are the windows one sees on a Smalltalk screen.

The Flavor System [3] of Symbolic's Lisp Machine is an implementation of objects in a dialect of Lisp. This system extends the hierarchical inheritance concepts of Smalltalk classes, allowing non-hierarchical combinations of classes called ***flavors.*** When a new flavor is created it can inherit the attributes of multiple flavors. Methods for handling messages are defined as combinations of methods from the other flavors. Conflicts can arise when two combined flavors process the same message in different ways but the Flavor concept resolves these conflicts in a uniform, prescribed manner, The Lisp Machine Window System [4] is a practical implementation of Flavors. The Window System manages communication between processes and the user. The user communicates to processes through windows via a keyboard and a pointer device (mouse). M e for windows have been classified into several flavors, For instance, all windows use the basic flavor ***minimum-windon..*** It contains the minimum functionality that a window must have to behave as a window. The ***window*** flavor adds more sophistication. It is the ***minimum-windon.*** plus methods (called mixins) to handle stream input, to draw borders, to draw labels and to do graphics. When a window is instanced, the process specifies the flavor and any initial values for internal states. Lisp returns an object descriptor of the particular window created. To communicate with the window, the process sends messages to the window object. The messages prescribe a uniform syntax and semantics for objecl communication. For example, to draw a vector in a particular window, the process sends the message *:draw-line* to the window and a vector will appear. In the following excerpt from a Lisp program, a window is created and graphics operations are performed in that window:

```
;first create a window of flavor 'window'
;    initialize instance variables as follows:
           borders are 4 lines thick
,          edges are from the mouse
           a label is displayed
,          expose it

(setq my-window (make-window 'tv:window
                 ':borders 4
                 ':edges-from ':mouse

                               ':label "my very own"

                               ':expose-p t ))

;draw some graphics in the window

; a line from (10,10) to (50,70)
 (send my-window ':draw-line 10 10 50 70)
;a circle at (50,50) of radius 10
```

```
(send my-window ':draw-circle 50 50 10)
;a circle at (50,50) of radius 5 using exclusive or
(send my-window ':draw-circle 50 50 5 tv:alu-xor)
```

The above window can respond to the graphics messages because the flavor **window** includes the graphics-mixin. In summary, the user has complete control over the characteristics of the window and can easily communicate with multiple windows by sending messages to the appropriate objects. The cursors, menu system and keyboard are also handled via the Flavor concept.

3. Object Oriented Design.

Having described object oriented systems, it is appropriate to see how the principles and properties of these systems can be applied to the software design process. The examples of previous sections illustrated implementations of object oriented systems. This section concentrates on the design process without regard to actual implementation.

The methodology outlined here is appropriate at both the preliminary and detailed design stages of the software engineering life cycle. The design proceeds from a functional requirement which describes the overall goals and capabilities of the system.

3.1 Booch's Methodology.

Grady Booch [5] outlines one approach for designing systems with objects in mind. His procedure consists of three steps:

1. Define the problem.
   This is the conventional process to describe the nature and scope of the system to be built. Iterations with the other steps are required as new aspects of the system are discovered.
2. Develop an informal strategy.
   A natural English description is used to describe the operations and objects of the system.
3. Formalize the strategy.
   Using simple rules and the informal description of the system, identify the objects (nouns), attributes (adjectives) and operations (verbs) required.

Ada @ is used to describe the visible interfaces to each object and the explicit operations which can be applied to the objects. This process is successively applied to refinements of the system. Booch uses this methodology to solve five design problems ranging from leaf counting on trees to a heads-up display system for fighter pilots.

® *Ada* is *a trademark* of *the Department* of *Defense.*

At first glance, Booch's approach seems attractive. Just write down a description of the situation to be modelled, underline the nouns (these become objects), then the verbs (these become operations) and translate the design into Ada. But he has not described a design methodology. He has given guidelines for translating a design into an object oriented implementation. In fact, his guidelines are useful for that step. However, moving from the problem definition step to the informal strategy is the most difficult step of the process.

3.2 A New Methodology.

This section presents an alternative object oriented methodology which is intended to define the overall design of the system to be developed. Once this design has been established, the techniques of the previous section can be used to describe the strategy and translate it into an implementation. The new method does not extract objects from the design but builds the design based on abstractions of the objects themselves. Therefore, the primary effort in this approach is the definition and characterization of the abstractions. The following steps are involved:

1. Identify the data abstractions for each subsystem.
   These data abstractions will be the objects of the system.
2. Identify the attributes for each abstraction.
   These will be the instance variables for each object.
3. Identify the operations for each abstraction.
   These will be the methods for each object.
4. Identify the communication between objects.
   This step will define the messages which objects can send to each other.

This process must be repeated at each level of abstraction. Through successive refinements of the design, our view of the system will change depending on our needs at the moment. Each level of abstraction will be implemented at a lower level until we reach a point where the abstraction corresponds to a primitive element in our design. Each level of abstraction should provide some uniqueness that cannot be expressed at the next lowest level. After a review of current graphics animation work, this methodology will be applied to the design of the animation language.

4. Computer Graphics Animation Systems.

Computer graphics representations have progressed from the early use of lines to produce wire frame images of three dimensional models. Extensive research over the past ten years has seen a progression from these crude mathematical renderings, through simple shaded presentations, up to the current state-of-the-art which includes more realism in the results. The realism is the result of success in modelling more realistically the models' environment. Transparency, translucency, shadows, illumination models and surface properties are a few of the areas where research has produced algorithms resulting in more acceptable synthetic images. A browse through the proceedings of the ACM's Special Interest Group in Graphics, SIGGRAPH, provides a historical review of the increasing sophistication of advanced image synthesis. The current trend in computer graphics is to apply these advanced techniques toward the production of quality animation.

4.1 Previous Work.

Although dozens of films produced using computer graphics have appeared over the years, the literature has concentrated on the algorithms used to produce the images, not on the animation systems themselves. This is probably because few animation systems exist. The films are used as a vehicle for illustrating the results of the algorithms or to make an artistic statement. Most of the film production is done by running a sequence of unrelated programs through the control of command files. The major effort has been on image quality and realism, not on the animation interface itself. The current leaders in this type of animation include the New York Institute of Technology (NYIT), Ohio State University, Lucas Films (of Star Wars fame), Cornell and Information International (III). Recently a new company, Digital Productions, was formed [6] to create 3D animation using a Cray supercomputer.

A few of the animation systems that have been described are included in this section. Reynolds[7] has worked for several years on the Actor/Scriptor Animation System, ASAS. Originally started as a Master's Thesis at MIT, ASAS was used at Information International Inc., III, to produce sequences for the Disney movie TRON [8]. ASAS is implemented in Lisp and relies heavily on object oriented concepts. ASAS *actors* are the participants in the animation. They communicate by sending and receiving messages. Once an actor is started (instanced) it remains a part of the animation until it stops itself or is stopped by another actor. Actors can also be given *cues* as to when they should appear or disappear. The processing of the cues is contained within the actors themselves. Characteristic of other Lisp-based systems, ASAS can use all of the power of the Lisp interpreter, only having to extend the capabilities of Lisp through new functions and forms.

The MIRA System [9] also extends a computer language, in this case Pascal. Abstract graphical types are defined which describe the participants in the animation. An animation is described by a sequence of scenes. Each scene has a name and is a sequence of statements manipulating *actors*, *cameras* and *decor*. Decor includes graphical objects which do not change within a scene.

The efforts of a commercial computer graphics animation company, Pacific Data Images are described in [10]. The company produces commercial computer-generated animation for broadcast television. Few specifics of the system are given but the overall production process is outlined. This systems differs from the other two in that an animation language interpreter was written in a high level language, C, rather than extending the language itself.

Finally, [11] describes a procedural based animation system. Procedural systems are akin to object oriented systems in that a procedurally modelled object is entirely described by its procedure and parameters.

5. The CRD Animation System.

Industrial computer graphics applications share some characteristics with those of the university and commercial communities. Software for modelling and display (rendering) is common to both environments. However, university and commercial systems assume some artistic talent to communicate a message through the animation. On the other hand, the industrial environment is driven by analyses of modelled phenomenon. For example, an artistic interpretation of a robot in a work environment may show the robot go through apparent realistic motions, whereas an industrial robot motion must be predicted by sophisticated kinematic analysis. After all, the intent of such an animation is not to produce a pretty film but to gain insight into the interactions of the robot with its work environment. Also, if the animation is used as a marketing tool, prospective customers will not be impressed by an artist's interpretation of how the robot behaves but need to understand how the actual robot will perform. This is the key difference between this system and those described previously: the reliance on analysis rather than art. This also illustrates a problem that this animation system must address: the analysis software often already exists with its own user interface and data bases.

The proposed system will provide an automated graphics animation capability for the efficient creation, control and management of 3D computer generated animation sequences. The graphics animation system will completely automate the creation of high quality film and video showing the results of complex research, experiments and other computer generated analyses. Using an English-like script language as the user interface, the CRD animation system will provide automatic control of analysis, modelling, rendering, display and filming processes. Interfaces will be developed to analysis programs in the areas of structural analysis, molecular modelling, robotics and factory simulation. An open-ended design is proposed to simplify interfacing to existing and future CRD/external software. A generic animation language, frame and sequence databases and sequence editor are key system features.

5.1 **The** Animation Process.

Several steps are required to produce an animation. It is useful to review the process to see what steps can benefit from the proposed animation system.

1. State the intent of the animation.
   Every film is made with some purpose in mind. It could be to verify or understand some mathematical algorithm in relation to a physical phenomenon, to explain an abstract concept to an audience or to market a product.

2. Create a story and write a script.
   Creation of the story is a mental process that cannot be assisted but the description of the story as a script is useful for documenting, changing and creating a final product.

3. Run any simulations.
   The proposed system assumes that most animation will depend on some sort of computer model. Analysis runs must be made to provide the simulation results for the animation.

4. Create geometric models of participants.
   Some analyses have geometric models associated with them while others do not. For instance, a structural engineer performing a stress analysis of a turbine models the turbine with finite elements before the actual analysis is performed. Here, the model and analysis are tightly coupled: both analysis and display require the same model. However, in a molecular mechanics calculation, simple cartesian points model the atoms and connectivity relationships model the bonds. In this case, more sophisticated geometric models are required for the rendering process: spheres and cylinders.

5. Render the geometric models.
   This step involves applying computer graphics algorithms to take the computer geometric model, surface properties, lighting, etc. and producing images for display.

6. Preview the animation.
   Fast, interactive preview facilities catch conceptual and mechanical errors in the animation.

7. Edit the completed frames.
   Titles, credits and special effects such as dissolves and fades add a professional touch to the completed sequence.

8. Shoot the film or video.
   This is the final step in the process and involves selection of frames and exposure of the film or video.

5.2 Major Subsystems.

The subsystem breakdown is chosen to delegate authority for the steps in the animation process and to correspond to software which is already available. This is a conventional, function oriented design since many of the modules already exist or behave as independent subsystems. An anthropomorphic flavor is used throughout the module descriptions so that the correspondence with conventional movie making can be maintained. This appears to be a natural abstraction to use in the animation system. Modules interactions with other modules in the system are also described.

1. Director.
   This module reads a user's story or script and delegates responsibilities to the rest of the modules. It knows who should do what and when but doesn't know how the other modules do things. The user interface to this module is through an Animation Script which can be entered into a file manually or produced via an Interactive Script Generator. The scripts are written in a very generic fashion. The intent is to provide full animation capabilities but to keep the style and semantics uniform. Typically, the other modules in the system understand different languages. The burden of translation is not on the Director but is delegated to modules called Liaisons.

2. Liaisons.
   Liaisons translate the Director's commands into commands their assigned modules can understand. Likewise, they accept input from these modules and send it to the Director in a form the Director can understand.

3. Sequence Clerk.
   This module is a clerk for the Director. It keeps track of the stage of each piece of the animation by storing status information in a sequence data

base. The data base itself is hidden within the module but can be accessed via messages from the other modules. If the Director needs to know the status of any part of the animation it can ask the Sequence Clerk.

4. Analysts.
These modules perform analyses in a variety of scientific fields. They are very intelligent and act as technical consultants to the Director. Although they have their own way of doing things. they can be told when to start and to report when they are done. A sampling of Analysts include ADINA, a non-linear finite element analysis program; ANSYS, a general purpose finite element analysis system; MNDO and Gaussian 80. molecular mechanics programs; and GRASP, the General Robot Arm Simulation Program. The interfaces to these modules are already defined and cannot be changed. Since these Analysts have pre-established interfaces, each one is assigned a Liaison to act as a go-between with the Director. The Liaison module is able to translate the Directors non-technical requests into command sequences which the Analysts understand.

5. Modellers.
These modules provide a means of creating models for the animation. They typically use some geometric primitives to construct complex representations of structures. GEOMOD, Movie.BYU, the Calma Solid Modeller and the Rensselaer Polytechnic Institute (R.P.I.) Build program are a few of the modellers available at CRD. R.P.I. is also working on a new interactive modeller called TIM which will take advantage of high performance vector graphics devices. Except for TIM, these systems also have predefined interfaces and once again the Director talks to them through Liaison modules. Also, since an animation will often contain many different types of models, a Coordinator is required to combine the models into a homogeneous format the Director can control.

6. Coordinator.
The Coordinator can combine geometric representations of many modellers and create homogeneous representations that a given Renderer can understand. It communicates between the Liaisons of Modellers and Renderers.

7. Renderers.
These modules take geometric information and environmental information such as lighting and camera positions and create raster image files for later display and filming. They have varying degrees of sophistication and are always wary that they can be replaced by more sophisticated renderers. Movie.BYU, Synthavision **and** the R.P.I. Super Quadric renderer are examples of such modules. Each system has its own language it understands and its own way of saving images. Liaisons will interact with the Director to control them.

8. Editor.
The Editor takes animation sequences kept by the Sequence Clerk and adds special effects, titling and the like. Finished frames are sent to the Frame Clerk.

9. Frame Clerk.
The Frame Clerk's job is to keep track of where each finished frame is, whether it has been recorded and also to archive frames when they are no longer needed. It knows several places it can keep frames; online disk, magnetic tape, optical video disk and other Vaxes in the network.

10. Recorder.
The Recorder knows how to perform the actual filming of the sequences. It obtains finished raster frames from the Frame Clerk, displays the images in a frame buffer and records the images on film or video. The Recorder knows everything there is to know about the operation of cameras, displays and video disk equipment.

5.3 Animation Language Design

The animation language is designed by describing the object classes comprising the animation. In each case, a description of the object class is given and some indication about the operations it understands. No attempt is made at this point to tell how the object performs. This decision will be delayed until the latest possible time in the design. There are two high level abstractions present in the system:

Scriptscontain a complete description of an animation sequence and can be considered the input of the animation process. A script has a file name and collections of objects called scenes, clocks and cues. Scripts are stored in files and can be created using a text editor or by the Interactive Script Generator. Operations that can be performed on these objects include: parsing and execution.

Stripsare the output of the system. These objects consist of ordered collections of frames. They can be created, edited and exposed.

The abstraction process continues with a description of the objects comprising the next level of abstraction. These are objects which are used to build the high level objects described above.

Scenesare collections of animate and inanimate objects and a description of their surrounding environment. Scenes have names, status information, actors, cameras, lights, props and backdrops. They can be created, edited, previewed and rendered.

Clocksare the time keepers of the objects. They have a resolution and speed. They can be created, modified, and turned on or off. Clocks send tick messages to each scene and animate participant in a scene. Other objects can have their own clocks to time their own activities.

Cuescontain temporal information to control the presence and behavior of a scene's participants. They have time intervals over which they receive messages. They can be created, edited and previewed.

Frames are ordered collections of raster images. A frame corresponds to one physical frame in the final animation sequence. Multiple raster images are often required to build one frame. The order reflects the sequence in which the raster images should be displayed in a frame buffer. Frames also have status information and an identifier. They can be created, edited and previewed.

Notice that as the abstraction process progresses, the objects are broken down into more manageable pieces. Also several common operations surface such as creation, editing and previewing which will add a uniformity to the final design. When a new operations is proposed on a given class of object, it is appropriate to ask whether that operation can be applied to other classes. This provides a completeness to the class operations.

For this paper, the abstractions are carried one step further.

Actorsare the animate objects of the scene. They are described by geometric or procedural models. They can also have analysis results associated with them. They can be rigid bodies, non-rigid bodies or procedural bodies. Their behavior can be described as a function of time. Their timing is controlled via clocks. Actors can be created, moved, rotated, turned on and off or told to follow the results of some analysis. Actors can send messages to other objects in a scene.

Cameras are the objects through which the animation is viewed. These objects contain location, direction and focal length. They can be moved, rotated and turned on and off. Cameras have no geometric representation so that if one is in the field of view of another, it is not seen in the animation. Although multiple cameras can be present in a scene, only one camera can be on at one time. When a particular camera is activated, it sends a message to all the other cameras so that they can deactivate themselves.

Lightsare objects that illuminate the scene. They have position, color, type (flood, point) and direction. Lights can be moved, rotated and turned on or off. Multiple lights can be present and active at one time.

Propsare inanimate objects in the scene. Their position is specified once and remains the same for the entire scene. They can be moved by other actors though. The point is, they cannot change on their own.

Backdropsare raster images with no graphics structure. They may be backgrounds painted into a frame buffer with a paint program or images scanned into a file with a video digitizer. Backdropscan be created and stored.

Raster imagesare two dimensional arrays which have no structure, just pixel values, associated lookup tables and optional z buffers to maintain depth information. They can be stored, displayed and interpolated.

The abstraction process continues in this manner until no more new objects are required.

6. A Sample Animation.

This is a description of a scene showing the results of a robot simulation system. The syntax of the animation language is still not finalized but the script does give the overall intent of the system.

6.1 The Analysis System

The analysis package is called GRASP, the General Robot Arm Simulation program. GRASP is capable of predicting articulated robot motion given starting and ending positions of the robot hand. The user of this system interactively prescribes key positions of the robot hand and using kinematic techniques, GRASP produces a graphic display of the motion of the robot. The program also outputs into a file the transformations for each joint and member as it progresses through the simulation. The geometry of the robot required by the analysis system is very simple. Prisms are used to model members and cylinders are used to model joints. However, for display purposes, more realistic representations are used. Here, the engineer modelled the robot using GEOMOD, a commercial modelling package from Structural Dynamics Research Corporation, SDRC. The transformations produced by the simulation are applied to the vertices of the polygons produced by GEOMOD. More sophisticated representations such as super quadrics could be used. The geometric complexity of the robot is independent of the the analysis.

6.2 The Story.

The scene starts outside a work cell. The work cell is an enclosed room with one door. The door is closed. As the door opens, the camera enters the work cell, panning around the room. The work cell contains:

i. A robot.

ii. A table containing work pieces.

iii. A drill press which is used to machine the pieces.

iv. A conveyor belt used to transfer the pieces to the next work cell.

After the room is panned with the camera, the robot begins to move according to the results of the simulation. It picks up pieces from the table and moves them to the drill press. After the piece is processed on the drill press, the robot picks up the piece and places it on the conveyor belt. The process is repeated using various camera angles.

6.3 The Script.

Objects are instanced from a class by the 'is a' operation. Each object contains a series of methods which consist of a method name followed by a colon. Subsequent statements are messages to be sent to other objects or the object itself. To make the language less cumbersome, some non-object oriented syntax is used. For example, the statement

resolution =24

really translates into the following:

send self 'resolution 24.

This causes the instance variable resolution to be changed from it's default value to 24. The other method names are self-explanatory.

```
/*
 * first describe the scenes
 */
  Scene_1 is a scene {
      initial:
         actors = (GP66,drill_press),
         cameras = (camera_1,camera_2),
         lights = (light_left,light_right),
         props = work-table,
                 clock = scene-1-clock,
         cues = (startup,pan,simulate).
      }
  Scene1_clock is a clock {
      initial:
         resolution= 24,
         start=0,   .
         duration=1 minute.
      ]
/*
 * next define the participants
 */
  GP66 is an actor (
      initial:
         modeller='GEOMOD',
         analyst='GRASP',
         renderer='MOVIEBYU',
         type='robot'.
      tick:
         send analyst 'tick.
      }
  Camera_1 is a camera {
      initial:
         lens='wide-angle'.
      ]
  Camera-2 is a camera {
      initial:
         lens='regular-lens'.
      ]
  Light-left is a light {
      initial:
         type='flood',
         color='red'.
      ]
  Light-right is a light (
      initial:
         type= 'point',
         color='white'.
      ]
  Work-table is a prop {
      initial:
         x=-5,y=5,z=0,
         type= 'table'.
      ]
  Drill-press is an actor {
      initial:
         x=5,y=8,z=0,
         type='tool'.
```

```
        }
    Clock–1 is a clock (
        initial:
          resolution = 24,
          time=0.
        }
    Cam1_action is a time–interval{
        initial:
          start=10 seconds,
          duration= 30 seconds,
          increment=180.0/number_tics.
        tick:
          send camera–1 'rotate increment.
        }
    Cam2_action is a time–interval{
        initial:
          duration=15 seconds.
        }
/*
 * define the temporal relationships
 */
    startup is a cue (
        initial:
          time–start=0,
          duration=10 seconds.
        start:
          send camera–1 'on,
          send light–left 'on,
          send light–right 'on,
          send work–table 'on,
          send drill–press 'on,
          send GP66 'on.
        end:
          send pan 'start.
        ]
    pan is a cue {
        initial:
          interval=cam1–action.
        start:
          send camera–1 'on.
        tick:
          send camera–1 'tick.
        end:
                    send simulate 'start.
        ]
    simulate is a cue {
        initial:
          interval=cam2_action.
        start:
          send camera–2 'on.
        tick:
          send camera–2 'tick,
          send GP66 'tick.
        }
/*
 * now run the animation
 */
  send scene–1 'run.
```

Notice that the animation really requires creating instances of objects and specifying values for their instance variables and methods to handle messages. The objects interact with other objects by sending them messages. The whole animation process is started with one message to the scene.

7. Implementation.

The animation system design is nearing completion and thoughts are turning to the implementation. C [12],a portable system/application language will be used. The current strategy calls for the implementation of a flavor-like facility and C is attractive because of its dynamic memory allocation support, data structure capabilities and pointer features. The initial implementation will support a limited number of analysts, modellers and renderers. Additional module support will be added once the system's capabilities have been demonstrated.

8. Summary.

An object oriented approach to the design of a computer graphics animation language has been described. It has been illustrated that the data abstraction process is the critical step of the design process of object oriented systems. Besides creating a useful animation system, this work is meant to provide a testbed for using the object oriented approach to system design and implementation. The lessons learned in going through this rigor will improve our understanding of the object oriented methodology. The same technique will be applied to the design of the Sequence Clerk, Frame Clerk, Liaisons, Editor and Recorder.

9. Acknowledgements.

The Animation Project team at the Rensselaer Polytechnic Institute Center for Interactive Computer Graphics has made many technical contributions to the ideas proposed in this paper. This group, led by Gray Lorig, is engaged in a parallel effort to build a graphics animation system. The CRD project expects to benefit from their expertise in the areas of modelling and sophisticated rendering software. Jon Davis of CRD's Solid Mechanics Branch has provided interfaces to the GRASP robot simulation system. Vince Scavullo of the Information System Operation offered many helpful refinements during the preparation of this manuscript.

0. References.

1. 1. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," PhD Thesis, MIT, 1963.
2. Byte, August 1981.
3. Lisp Machine Manual. Symbolics Inc., 1983.
4. D. Weinreb and D. Moon, "Introduction to Using the Window System," Symbolics Inc., 1983.
5. G. Booch, "Software Engineering with Ada," Benjamin Cummings Publishing, 1983.
6. "Beyond Special Effects," Computer Graphics World, vol. 6, no. 1, January 1983, pp. 63-66.
7. Reynolds, Craig W., "Computer Animation with Scripts and Actors", Computer Graphics (Proc. of SIGGRAPH '82), vol. 16, no. 3, July 1982, pp. 289-296.
8. "Disney Takes the Lead with TRON" , Computer Graphics World, vol. 5, no. 4, April 1982, pp. 41-45.
9. N. Magnenat-Thalmann and D. Thalmann, "The Use of High-Level Graphical Types in the Mira Animation System", IEEE Computer Graphics and Applications, vol. 3, no. 9, December 1983, pp. 9-16.
10. R. Chuang and G. Entis, "3-D Shaded Computer Animation - Step by Step," IEEE Computer Graphics and Applications, vol. 3, no. 9, December 1983, pp. 18-25.
11. H. Hedelman, "A Data Flow Approach to Procedural Modeling," IEEE Computer Graphics and Applications, vol. 4, no. 1, January 1984, pp. 16-26.
12. B. Kernighan and D. Ritchie, "The C Programming Language," Prentice Hall, 1978.