EFFICIENT UTILIZATION OF GRAPHICS TECHNOLOGY FOR SPACE ANIMATION

Gregory Peter Panos
Lead Engineer, REGIS Lab.
Rockwell International Corporation
Space Transportation Systems Division
Simulation and Systems Test Department
REGIS Computer Animation Laboratory
12214 Lakewood Boulevard MC DA-46
Downey, California 90241
(213) 922-0200

ABSTRACT

Efficient utilization of computer graphics technology has become a major investment in the work of aerospace engineers and mission designers. These new tools are having a significant impact in the development and analysis of complex tasks and procedures which must be prepared prior to actual space flight.

Design and implementation of useful methods in applying these tools has evolved into a complex interaction of hardware, software, network, video and various user interfaces. Because few people can understand every aspect of this broad mix of technology, many specialists are required to build, train, maintain and adapt these tools to changing user needs.

We have set out to create system where an engineering designer can easily work to achieve their goals with a minimum of technological distraction. We have accomplished this with high-performance flight simulation visual systems and supercomputer computational horsepower. Sophisticated but simple to use geometry generation, translation and modification systems input designer concepts to a motion design systems after which our visualization and scene rendering tools are invoked. Control throughout the creative process is judiciously applied while maintaining generality and ease of use to accomodate a wide variety of engineering needs.

INTRODUCTION

Planning of space missions has historically been a slow and tedious process. Drawings and exact measurements were drafted on paper for the various sequences that had to occur throughout a mission scenario. Launch preparation, mission operations, return to earth and post-flight ops are analyzed for time-line schedule conflicts, potential problems, and for detailed failure avoidance.

Within the last decade, CAD systems have become the predominant tools to assist with mission data design, operational determinations and detailed analysis. 3-D CAD systems have proved extremely valuable in the areas of 3D design data storage, distribution, retrieval and modification. Although most mission information can be processed by traditional CAD systems, there are major gaps in their ability to rapidly work out "what-if" changes and to easily create video based presentation materials.

Recent Developments

High-performance simulation graphics systems have ushered in a new age of productivity with tools that allow orders of magnitude increase in performance. These systems allow an analyst to rapidly prototype changes and evaluate operational procedures in real-time, while providing videotape recordings of their results.

The variety of new systems, software tools and fully interconnected networks, allow complex planning and analysis scenarios to be automated.They also tend to be virtually self-documenting. Distribution of video-taped presentations to high level decision makers has rapidly become standard operating procedure for critical projects that require rapid turn-around.

In our quest to satisfy advanced visualization needs for hardware design and operational simulation, we have identified several key areas of concern.

3D Object Data Configuration

Data Compatibility Interfaces

Multi-System Communication

Production System Integration

Production Compatibility

3D OBJECT DATA CONFIGURATION

When working with 3D geometric representations of objects whether they be parts of a large spacecraft or minute gears and wires, it is very important to know their position, scale, and orientation. A designer must define an object hierarchy relative to a global zero point and specify a rotational point for each moving component, otherwise, no useful motion can be performed. Part colors and shading type, levels of detail, transparency and texture choices must also be carried along with object data geometry and topology.

We have constructed a variety of tools which allow a designer to read, status, break up, combine, delete, add, modify, reorient or otherwise change object geometry and / or attributes with a series of simple commands.

For example: p2p -ro 90 -45 30 -su 100 -tr 1000 10 -50 < part > newpart

This "p2p" filter will scale file "part" by 100, rotate X by 90, Y by -45 and Z by 30 degrees, will translate the object to 1000, 10, -50 and create a new file "newpart". These tools allow a designer to avoid manually searching and editing 3D object data files to make changes. Many of these tools can be easily adapted to menu based windowing environments.

We have found that the more robust a 3D object database structure is, the easier it is for designers to define important object attributes early on. This reduces the need to manually add information later. Changes can be made easily, quickly and effectively when a complete database structure already exists.

DATA COMPATIBILITY INTERFACES

Once a 3D object database incorporates all pertinent information, that data must be made compatible with differing 3D graphics systems and software. Many of these systems require their own special format for object data and some reqire more than one file of information to process an image for a single object. Compatibility can become time consuming when migrating useful information from one system to another. To deal with this problem efficiently, a series of tools have evolved. They are:

Filters : Strip out, add or process numerical information or object attributes from one input object to another. Filters usually work on data used by one type of system or software.

Translators : Reformat object data, often in a major way, for use in different computer display system hardware and rendering software packages which require very specific input formats.

Compressors : Strip out unused information, truncate long numbers, optimize, encode and combine data in databases to to avoid redundancy. Archive utilities are specialized compressors. They are useful in reducing data storage requirements and in minimizing data transfer time.

Switchers : Substitute one section of data for another, often geomety, when a boundary condition is reached or some external flag has been set. These are useful for performing dynamic "Level Of Detail" changes on display systems. Limited throughput often degrades performance degrades as level of detail increases. As an object approaches the viewer, higher fidelity versions are switched into display system memory

Pixel Encoders : Allow data-rich pixel based images to be reduced into smaller more compressed files for better storage size.

Color Compressors : Allow images with many colors to be reduced and averaged down for systems with less capability to process and display that data. Also reduces file size.

Compositors : Allow a neutral user designated color to act as a window for another image to show through. A composite image file of both images can be saved independently.

Mappers : Allow a designer to define sections of an image to be used as surface texture maps for wrapping around or pasting bit-map images onto objects. Often images are scanned in with a video device to create the images used in texture mapping applications. This technique can be useful, giving the viewer the impression greater apparent detail exists on an object's surface then is contained in its geometry description alone.

Flexible data compatibility tools allow designers to free themselves of the limitations imposed by different display hardware and / or rendering software. Virtually any type data can be used anywhere it is needed (with a little help).

MULTI-SYSTEM COMMUNICATION

Data transfer and remote system control

Production environments are often complex and interconnected. This imposes many constraints upon a designer. Passing data between unlike systems can occupy valuable creative time. Human interaction in performing file transfers and conversions is unneccesary and inefficient. These tasks should be handled in an automated fashion. To this end, the following tools have been developed:

Transfer Utilities : Small, command driven tools which allow a designer to transfer single or multiple files from one place to another place with a minimum of headaches. These tools can be highly intelligent. They might check specific system directories to determine what files are there and which of them are current so as to retieve or send them.

For example: Getnet Regis DUA0 users.panos DAT 5 /usr/greg/data

This command will Get all version 5 .DAT files from the directory users.panos on disk DUA0 on the Regis system over the network link and it will place them in the directory /usr/greg/data on the system where the command was evoked.

Control Utilities : Allow a designer to send a series of control commands from a system port to an external peripheral device (such as a Videotape Recorder) to do something useful.

For example: Vtr -m 1 -l 6 -b 2000 -r 10 -S -f Regis::Renderer

This command will send a command out to a pre-designated port on the system where the command was executed. The port is wired to a Videotape Controller and the command is asking it to select VTR machine number 1 (-m 1) and to connect the incoming video line number 6 to the machine's input (-l 6). The -b 2000 option places the Vtr edit in-point to frame 2000 and allows a 10 frame edit (-r 10) at that point. The -S option asks the Vtr to go to "Standby" mode after it is done and the -f option will send a "done flag" to the "Renderer" program which is running back on the "Regis" system so it may begin another task.

Very often these utilities are highly system and software specific. Many of these tools contain security passwords, system identification numbers, codes, data-word sizes, and directory destinations and will allow privledged access that should be carefully protected.

Production environment developers must determine the safest and most efficient scheme for inter and intra-system communication and control. Frequent changes to information embedded within these tools should be avoided. Insured reliability of use for all users and the programs that serve them should be a top priority.

PRODUCTION SYSTEM INTEGRATION

Production system designers often overlook analog video signal routing problems. Digital computer display system details and their networks are often closely studied, while requirements for video signal distribution, propagation and interfacing are left to last and regarded as the least important aspects of the system.

Video signal needs must be attended to as a primary area of concern for the production system designer. It is a common fallacy to believe that many desired effects can be achieved digitally with rendering tricks and additional computer based techniques. Although this may be true for those who posess intimate knowledge of these tools, it is much simpler to create a desired effect with the use of video signal mixing and compositing with multi-track recording, time-base correction, and encoder function controls. This is the cross-over point where designers who have concentrated on becoming very proficient programmers become lost. Video engineers, with their knowledge of RGB, sync, key-channel, matting and analog calibration often take over at this point. Here are a few examples of video signal processing options:

Encoding : All Frame-buffer and display systems require signal conversion of their RGB output into a composite video form, usually NTSC. Encoders perform this conversion and are able to fine tune and juxtapose certain components of the output video. For special-effects, calibration, and interfacing to video switchers, color-keyers and VCRs, encoders are needed.

Keying : Enables one encoded video signal to be superimposed over a background video signal from a different source. A neutral color acts as a window on the foreground video signal. This is a very important feature for any advanced production system.

Synching : All input and output video signal sources should be locked to a synchronization clock signal to allow glitch-free effects. Source switching, dissolving, fading, keying, etc. all work much better when all systems are "genlocked" to house sync.

Switching : Multiple video sources can be switched electronically by computer or with keyboard based control functions in a good switcher. Different video lines can be re-channeled on the fly as production needs change and advanced effects like bordering, split-screen, quad-screen, title insertion, fade, dissolves, wipes, and highlights can be performed by good video switcher component which has been properly configured.

Recording : VTR Controls that are properly interfaced to the production system can be a major benefit to creative usage of all possible effects. It is a disadvantage for a designer to have to manually set up and button push a VTR's console to get it to perform the required actions. Today, most VTRs have controllers available to enable remote operation. Computers may also be allowed control of a VTR with multi-tasking.

These options enable a designer to create advanced visual material with the appropriate interconnection of video component and display system output. Multiple passes allow for increased scene complexity when using matting and keying techniques that a computer display alone would not be capable of. Simple effects like a wipe or page turn tend to be very time consuming for a digital computer to perform while a video effects switcher will function in real-time to accomplish such an effect.

With proper initial configuration of the above video components, a designer will be able to dynamically set up each scenario by using commands and script based execution sequences run from the host computer where they are already doing most of their work. This approach can remove the need to hand wire a video panel, perform manual VTR edits, or keep a video person around on a full time basis.

PRODUCTION COMPATIBILITY

Naming Conventions

In any production environment, there exist many different formats to contend with. Digital text and binary data files can be extremely varied in their formats. A program which requires a picture file for display on a computer frame-buffer may want it's data in ASCII, human readable text, while a CAD program needs a BINARY compressed geometry file representation as input.
Therefore a good plan for production compatibility becomes essential to assure continuity and efficiency throughout the creative process. Step one is to think out a good labeling scheme for the varity of information one needs to deal with.

Good use of terminology for different information types can be essential to properly understand what takes place in a production script. As a production evolves and is modified, it can be virtually impossible to keep track of files and taped sequences for replacement and editing if sensible names are not used. Disk and tape archiving is greatly simplified with this practice as well. Refer to Appendix A for details regarding suggested format naming conventions.

Commenting

When program tools modify files of data it is important to add comment lines describing what has been changed in the data and when changes were made.
For example, here is a geometry file that was processed through our "p2p" filter to reposition the 3D object data:

```
| Tue Feb 28 11:41:55 1989  polyp2p -ro 90 180 -45 -su 10 -tr 100 200 -300
surface Active FlatShaded   |   attributes Active FlatShaded
v  7   (7 Vertices)         |   p  5        (5 Polygons)
w 256                       |   0 1 2 3     (255 175 0)
100 -1610 980 256           |   4 5 6 0     (255 0 255)
100 -1610 -1580 256         |   0 6 5 1     (255 255 255)
1910 200 -1580 256          |   3 2 5 4     (0 0 255)
1910 200 980 256            |   3 4 6 0     (255 0 0)
100 2010 980 256
100 2010 -1580 256
-1710 200 -1580 256
```

In the above example, the "p2p" filter added the comment line to the top of the file saying what exactly has been done. In this case the file was rotated by X = 90, Y = 180, Z = -45, scaled up by 10, and translated by 100, 200 -300.

If necessary, all the values in the comment line could be used to convert the data back to the original unmodified form by negating the numbers and re-filtering the modified file. This feature can save hours for a designer who has made a mistake on or deleted by accident a critical file.

Listing

Another good practice is for a designer to build complex objects out of as many smaller object component files as possible. It is easy to combine them later with a "concatenate" command or in a script. This allows a great deal of flexibility when making slight changes, color-coding pieces or when showing only what is needed to aviod overloading a display generator or renderer.
Using a "Listing" file is the preferred way of specifying many different parts to be treated as one big part at runtime.

For example, in a Space Shuttle made of 7 major component files, the "Listing" file would be defined like this:

```
# ---------------------------- Space Shuttle components ------------------#
#
# Pathname/file              Description                  # of polygons
# -------------              -----------                  ------------
/shuttle/fuselage.P          # FUSELAGE                   p  170
/shuttle/blkhd.P             # BULKHEAD                   p  60
/shuttle/cnopyfwd.P          # CANOPY                     p  80
/shuttle/bay.P               # BAY                        p  50
/shuttle/vrtail.P            # VERT. TAIL                 p  30
/shuttle/wings.P             # WINGS                      p  80
/shuttle/rms.P               # rms pieces (4 of them)     p  70
```

When a display program is invoked we would specify the "shuttle.l" file as the file to display. This would treat all the pieces as one singular object. They would all move together but each would retain their own attributes such as color, shading type, transparency level, etc.

Display tools

The "examine" program allows a designer to work with 1 to 4 objects, each with up to 16 data files. This enables a designer to easily rotate and translate the view, change the background color and light direction, and to manipulate each object separately. The program also has a helpful arrow which points at the light source and a clock hand which rotates once every second to indicate the display system frame rate for use in overload assessment.

For example: examine -f0 shuttle.l -f1 earth.p -f2 gps1.p -f3 gps2.p

Here we control the shuttle, the earth and two gps satellite models all as separately moving objects. Each object has independently controllable offset, rotation, and translation, and is manipulated through separate data tracks.

"Examine" is one of our major workhorse animation programs that we use in our production environment. It runs on the Poly 2000e computer image generator, however, its functionality is extensible to virtually any real-time display. File flipping of 3D object data, where object data varies its geometry from one frame to the next, is also possible. This feature will allow animated display of incrementally deformed 3D objects and to rapidly flip through them while maintaining real-time control of their position and view orientation. However, one must always watch their polygon count when loading these large amounts of data into a real-time system. Otherwise, performance may degrade and other undesirable display artifacts may appear. Refer to Appendix B for the examine program's options and argument specification.

Rapid Prototype Generation

The last Real-time animation program that I will mention is the "RPG" program or Rapid Prototype Generator. This "CASE-like" tool was originally conceived by many different people at different facilities simultaneously (like all great ideas). RPG is another major tool which helps us to carry the banner for more efficient and creative production.

The concept is to allow a non-programmer (and hopefully good designer) to rapidly define, build and animate complex hierarchies of 3D object components. Without the need for a great deal of complex technical display-system-specific knowledge, a designer can use RPG to do this in record time.

First an inventory is made of the 3D objects that are to be used. Determination of what is going to move relative to what (defining the hierarchy) and finding offset distances and an axis of rotation for each moving part is next.

This is done by making a simple move to the center point (0,0,0) in any simple display program and then reading the numbers off the dial box or screen). A text script is then created by typing in the object hierarchy, offset values, track assignments and display diagnostic features. This step can take from 5 minutes to a half hour depending on how well a plan is mapped out and how fast one can type.

The script is then run through the RPG program "update"on the host system. Essentially a compiler, "update" generates C source code and compiles it into an an executable module. Next, a "builder" program is run on the image generator system where the hierarchy (as described in the RPG script) is assembled and the 3D object modules are loaded into real-time display list memory.

Last, the executable module created by "update" is run on the image generator. The designer can then use standard channel-based animation package features to define, save and preview keyframe files.

A detailed description of the statement format and syntax for RPG scripts is beyond the scope of this paper. We also feel it would be possible to make the implementation we have chosen even simpler to use. A menu driven script builder utility with a full graphical interface has been suggested and will be developed in the near future. Generalized structures in the program will allow differing real-time displays to utilize the same scripts and 3D object data through the use of special device driver modules that can be linked to the RPG program. Refer to Appendix C for an example of a simple RPG program script.

CONCLUSION

Only through years of experience have we been able to best determine what was needed to enable efficient production of computer animation and effects. After many hours of difficult 3D object geometry debugging, we were able to design flexible and easy to use tools to allow us to do in minutes what used to take hours. After many days and weeks of writing custom non-reusable C programs for every animation, we developed "RPG" with which we could produce our working programs in minutes. Once we created our animation package library we could generate and save reusable motion data files that took hours to develop by hand. Motions can be recalled in seconds and used in virtually any animation.

Once things begin to work as efficiently as possible, the greatest burden lies with the designer to dream up, build and produce the visuals that they desire. And yet, after countless creative sessions, more efficient ways to facilitate the creative process always seem to emerge.

ACKNOWLEDGEMENTS

I wish to thank Kenneth M. Stern, my associate in the REGIS Lab. for his fine contributions and corrections to this material. I would also like to thank Ben Thompson, STSD Advanced Engineering for additional comments and inspiration.

APPENDIX A

Description of suggested naming convention for different data file types.

Pixel Image data file formats:

```
Image.BIN   : An Image file in computer readable BINARY.
Image.RLE   : An Image file in "Run Length Encoded" BINARY.
Image.ASC   : An Image file in ASCII, human readable TEXT.
Image.HAM   : An Image file in Amiga "IFF HAM mode" BINARY.
Image.IFF   : An Image file in Amiga "IFF" Low, Med or HiRes BINARY.
Image.R8    : An Image file in BINARY RLE for "Cubicomp Picturemaker".
Image.PIX   : An Image file in BINARY RLE for Cubicomp Map Mode.
```

Text data file formats:

```
file.COM  : A Script file of run-time Commands in TEXT.
file.LOG  : An output file of program run results in TEXT.
file.POS  : A "TRACER" hierarchical Scene data / offset file in TEXT.
file.MAT  : A "TRACER" part Material coefficient data file in TEXT.
file.LGT  : A "TRACER" Light source description file in TEXT.
file.ENV  : An "TRACER" Environment data file in TEXT.
file.RPG  : An RPG hierarchy, offset and control data file in TEXT.
file.HDR  : A "Cubicomp Picturemaker" Header file in TEXT.
file.WFT  : A "Cubicomp Picturemaker" Wireframe test filein TEXT.
file.SCN  : A "Cubicomp Picturemaker" Scene test file in TEXT.
file.MMP  : A "Cubicomp Picturemaker" Color Map file in BINARY.
file.ENV  : A "Cubicomp Picturemaker" Environment file in BINARY.
file.CM   : A "Cubicomp Picturemaker" Command Macro file in TEXT.
file.L    : A List of Geometry files with various components in TEXT.
```

3D Object Geometry data file formats:

```
data.GEO  : "Aegis Videoscape-3D" Polygon floating-point TEXT.
data.MOV  : "MOVIE.BYU" Polygon floating-point TEXT.
data.OBJ  : "Symbolics S-GEOMetry" polygon floating-point TEXT.
data.TRI  : "TRACER" Polygon triangle floating-point  TEXT.
data.SPH  : "TRACER" Sphere data in floating-point TEXT.
data.DAT  : "RI-CDAS" quartic data in floating-point BINARY.
data.WS   : A "Cubicomp Picturemaker" WorkSpace file in BINARY.
data.P    : "Poly 2000" Polygon 16 bit integer BINARY.
data.p    : "Poly 2000" Polygon 16 bit integer TEXT.
```

Motion data file formats:

```
data.MOT   : "Aegis Videoscape-3D" Object motion floating-point TEXT.
data.CAM   : "Aegis Videoscape-3D" Camera motion floating-point TEXT.
data.PW   : "Cubicomp Picturemaker" keyframe Position Word TEXT file.
data.K    : "Animation Package" Multi-Channel 16 bit integer BINARY.
data.k    : "Animation Package" Multi-Channel 16 bit integer TEXT.
```

APPENDIX B

Examine program options and arguement specifications.

Flag	Meaning	Default
-a#m <f>	animate object # using	
	mode m	once
	changing every f frames	1
	modes: [a]dd - add files starting at current position	
	[o]nce - 0 up to n.	
	[b]ounce - 0 up to n, down to 0, ...	
	[c]ycle - 0 up to n, 0 up to n, ...	
-b <file>	filename of background objectn	one
-c#	# channels	# present
-d	debug mode	false
-f#	filename(s) of object # follows	
-k	keyframe (.K) filename follows	default.K
-m	light moves with view	light is motionless
-r #	new rotation sensivity follows	32
-s	silent mode (no beeps) when switching files	false
-tn #	new translate W value for object n follows	256
	if n not present, value used for view	
-T	do not clear or write to text screen	write help screen
-v # # #	view offset is # # # (x,y,z)	0 0 0
-w #	new scale W value follows	256

APPENDIX C

Simple RPG text script for an interlocking gear mechanism animation.

```
# -------------- Interlocking Gear RPG Script ----------------------------#

First we Define the Hierarchy

TREE

world :         Chan0
                100 101 102 103 104 105
                XYZ.1
box :           world
                0
                box.1
gear0 :         box
                10 11
                gear0.1
gear1 :         box
                20 21 22
                gear1.1

# Next we assign the Offset and Track assignments

MATRICES

trans(  100, val[0][0], val[0][1], val[0][2] )

rotx(   101, val[1][0]*32 )
roty(   102, val[1][1]*32 )
rotz(   103, val[1][2]*32 )
trans(  104, val[2][0], val[2][1], val[2][2] )

scaleu( 105, val[0][3] + val[1][3] + val[2][3] + 256 )

trans(    0, val[3][0], val[3][1], val[3][2])
trans(   10, 0, 160, -110 )
rotz(    11, val[5][0] *32 )
trans(   20, 0, 40, 0 )
roty(    21, val[5][0] * ( -32) + 4096 )
rotx(    22, 16384 )

# Last we Define our Display Diagnostics

leds(0,"0 K0:%6d K1:%6d K2:%6d K3:%6d",val[0][0],val[0][1],val[0][2],val[0][3])

# ---------------------------------- EOF ----------------------------------
```