

Maureen and Mike
MANSFIELD LIBRARY

The University of **MONTANA**

Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

*** Please check "Yes" or "No" and provide signature ***

Yes, I grant permission *X*
No, I do not grant permission

Author's Signature *Ann S. Hale*

Date *5/30/96*

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.

Data Visualization Tools for the Production of Data Visual-Animations

by

Saxon Lorien Holbrook

B.A. The University of Montana, 1992

presented in partial fulfillment of the requirements

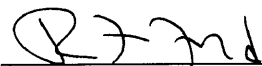
for the degree of

Master of Science

The University of Montana

May 1996

Approved by:



Chairperson



Dean, Graduate School

5-31-96

Date

UMI Number: EP41121

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.

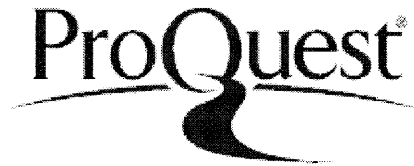


UMI EP41121

Published by ProQuest LLC (2014). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



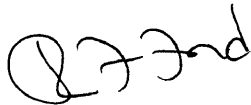
ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Holbrook, Saxon Lorien, M.S., May 1996

Computer Science

Data Visualization Tools for the Production of Data Visual-Animations (56 pp.)

Director: Ray Ford

A handwritten signature in black ink, appearing to read "Ray Ford". The signature is written in a cursive style with a circular initial.

This project describes a pipeline of data visualization tools, processing components and methods for the creation of *data visual-animations*. It identifies the roles of the visualization specialist and the application specialist, both in using the tools and providing the feedback necessary for creating effective data visual-animations. It also discusses the design of a key pipeline component; the smooth motion of the camera through a virtual landscape.

Contents

1 From Data Visualization to <i>Data Visual-Animations</i>	1
1.1 Introduction	1
1.2 Visualization Presentation vs. Research.	5
1.2.1 Scientific Visualizations.	6
1.2.2 Presentation Visualizations.	7
2 Original Pass (paradigm shift)	8
2.1 Background and Motivation.	8
2.2 Visualization Pipeline Components	8
2.2.1 Base Spatial Data Collection	8
2.2.2 Camera Paths	13
2.2.2.1 Single Segment	13
2.2.2.2 Multi-segment Flight Path.	14
2.2.2.3 FlyingCamera Macro	16
2.2.3 The First Movies	17
2.2.3.1 DX Image Caching.	17
2.2.3.2 MPEG Movies	18
2.2.3.3 IBM UMS Video Card.	19
3 Flight	21
3.1 The best way between two points isn't always a straight line.	21

3.2 Visualization Pipeline Components.	23
3.2.1 General Path Interpolation - “Smoothing the Savage Beast”.	23
3.2.1.1 Lagrange	24
3.2.1.2 Cubic Spline.	25
3.2.2 The PathInterpolation Module	26
3.2.3 Defining The Path	28
3.2.4 Multiple Paths / Network	31
3.2.5 Video	32
4 Looking at the World through AutoColored Glasses	34
4.1 More Realistic Landscapes	34
4.2 Visualization Pipeline.	35
4.2.1 Laying an Image over a Landscape.	35
4.2.2 Creating Input Layers.	36
4.3 Starting with Reality	40
4.3.1 “Transmorgification”.	40
5 Conclusions	42
5.1 Conclusions.	42
5.2 What does the Future Hold	44
Bibliography	46

List of Figures

1.1 Spiral Model.	2
2.1 Base Input Data Importing and Verification.	9
2.2 Quadrant Converter Equations	10
2.3 Range-Township Example.	11
2.4 Single and Multi-Segment Camera Paths	13
2.5 Sequencer and Surface.	14
2.6 Multi-Segment Determination Equations	15
2.7 FlyingCamera Macro in Visualization Program.	16
2.8 The First Movies	17
3.1 Segmented Linear Paths and their Slopes.	22
3.2 A Curve Matching Control Points	23
3.3 General Path Interpolation Overview	23
3.4 Lagrange Curve Interpolation Equations.	24
3.5 Lagrange Curve	24
3.6 Special Lagrange Curve	25
3.7 Special Cubic Spline System of Equations	25
3.8 Special Cubic Spline Min/Max Reduction	26
3.9 PathInterpolation Module	27
3.10 Path Designation Overview.	28
3.11 Path Programming.	29

3.12 Path Manipulation	30
3.13 Path View 1.	31
3.14 Path View 2.	31
3.15 Video Revolution Overview.	32
4.1 Landscape Layer Creation	34
4.2 Range of DEM Image Layer	36
4.3 Hydrology Image Layer	37
4.4 DEM Histogram Image Layer	38
4.5 Final Cover Layer	39
4.6 Lake MacDonald Image	40
5.1 The Final Pipeline	43

Chapter 1

From Visualizations to *Data Visual-Animations*

1.1 Introduction

The primary purpose of Data Visualization is the presentation of data in an illuminating form for a target audience. The target audience can range from application specialists such as scientific modelers, to people who can be expected to have at least *a priori* familiarity with the data, to “the public” at large. Traditionally, the target audience has been the modeler, so that the target end products of data visualizations have been individual images or simple time sequences presenting information in a manner assumed to be familiar to an application expert. Effectively presenting information to a less knowledgeable target audience requires a paradigm shift, oriented toward the production of data displays that "move" the target audience from a base level of knowledge to a

significantly enhanced understanding of the phenomena being modeled. The result, referred to as *data visual-animation*, typically takes the form of digital movies distributed on CDs or via Web sites, playable on PC's and workstations, or distributed in comparable non-digital form (e.g., VHS movies for playback via standard VCR).

The focus of this project is the design, identification and implementation of a set of tools and methods necessary to produce data visual-animations. A pipeline of processing components is described, and it is illustrated how they can be used in various natural resource analysis applications. The roles of the visualization specialist and the scientist/application specialist are identified, both in using the tools and providing feedback on the results produced with the tools.

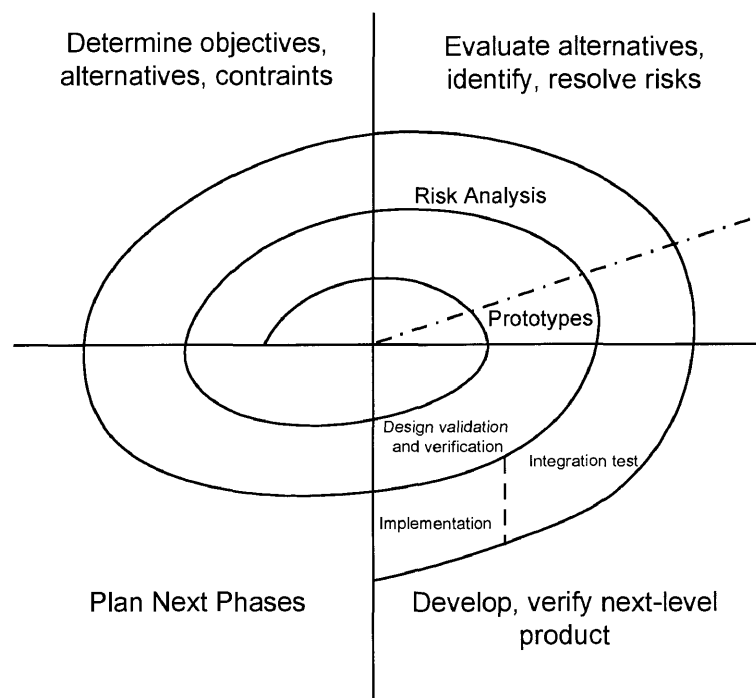


Figure 1.1 Spiral Approach

Boehm's spiral approach to software development [Boehm88] provides a good

model for describing this project. As illustrated in Figure 1.1, this model is itself characterized by a visual analogy: a spiral through design space with quadrants representing activities. The activities associated with the first quadrant and the starting position of the spiral are the determination of the project objectives, alternatives and constraints. In the next quadrant, alternatives are evaluated and risks are identified and resolved through prototyping. In the third quadrant, the next level of the product is developed and verified according to the previous quadrant's risk assessment and resolutions. Planning the next phase of the spiral process happens in the fourth quadrant. Each spiral pass carries the project through determination of the objectives, evaluation of alternatives through prototyping, developing the next-product and planning the next phase.

In applying a spiral model to the development of the data visual-animation pipeline, the initial iteration identifies one possible solution to the problem, in terms of a set of key required components that establish a primitive pipeline from input data to end product. The evaluation following Spiral One reveals which components are "standard" in a typical visualization environment, and which need to be developed specifically to support visual-animations. Each subsequent iteration improves on one or more aspects of the initial pipeline by identifying, specifying and implementing tools that smooth the transition from input to target outputs. As with any design, it is also possible that complete sections of the initial pipeline will be rerouted in order to adapt to the dynamic environment of changing application goals and evolving hardware and software capabilities to achieve the desired goal of a convincing and informative visual-animation. Thus, our description of each spiral pass focuses on the nature of the application environment and constraints, the identification and development of missing components, their combination with existing components to produce prototype visual-animations, and continuing refinement.

The spiral method is a convenient way to describe a natural discovery method, in which there are new aspects of a complex system uncovered gradually as the system is more completely understood. In fact, some of the information and the problems exist the entire time, but are only fully understood when the development has progressed significantly toward the target. For example, we may be unaware that a “false coloring” scheme used in a particular image doesn’t work until we have a smooth camera path and a coherent collection of images collected from that path. At that point, when we move from one image to another we notice that the colors appear incongruent because they obscure the desired effect. To put it another way, we discover the importance of having a better coloring scheme, and we need to find such a scheme. This is indicative of the spiral model’s description of multiple passes through the same design space: we realize at spiral iteration N that we need to solve the coloring problem, in the context of our greatly improved understanding of the role color plays in the application at this point in development.

The project focuses on an application framework involving the particular visualization problems related to depicting large datasets whose values are tied to large spatial regions which are referred to as *landscapes*. The landscapes are complex enough that a single viewpoint is not enough for complete understanding; they must be probed and explored. For this set of applications I first identify the required visualization components, then classify them into those tools, operations, and qualities which are unique to this type of application and those which are common to a wider class of visual-animations. For example, an important requirement is a “programmable moving camera,” i.e. an image capturing device which can be programmed by the visualization designer to move smoothly along a path through, by, or above the data, capturing imagery at

a desired resolution. This type of tool is an example of a facility useful in both landscape-based visualizations and a wide range of other applications.

This project relies on both existing visualization software and extra tools. The project uses IBM's **Data Explorer (DX)** [IBM95] as its basic visualization software running on an IBM RS/6000 workstation. **DX** is widely used in research, instruction, and commercial environments, and is available on a wide variety of computing platforms. **DX** also has an application programming interface and extensive programming libraries which allow an application expert to develop customized visualization components as modules which can be easily added to the general visualization environment. In addition to the tools provided in and added to the **DX** environment, a number of other stand-alone tools have been used and developed to assist in the creation, collection, compression, conversion, and/or maintenance of large sequences of individual images representing "frames" in a data visual-animation.

1.2 Visualizations Presentation vs. Research

The earliest forms of visualizations were graphs, charts and plots. They evolved into many forms with different purposes which can be classified into two groups; visualizations for the scientific modeler evolved to enhance their work and productivity through real time data depiction and a set of interactive tools; and visualizations for use in presentations evolved in order to allow the communication of complex results to a larger, less specialized group of people.

1.2.1 Scientific Visualizations

Visualizations were originally developed to enhance the productivity of the modeler, through real time, visual depiction of complex relationships in large datasets. The evolution of the computer made it very easy for scientists to create far more data than they could possibly decipher. The modeler had no choice but to search through reams of output or attempt to mentally “visualize” complex systems, relationships and results. When computer graphics display capabilities started to become readily available, simple graphs, histograms and data plots allowed for the results to be summarized in a much easier to use form. As computers became more powerful, models became more and more complex, and the data produced became more and more complex as well. Fortunately, the hardware evolution also extended to dramatically improved graphics and display capabilities, which enabled the new paradigm of pseudo-animations of 3-D spaces, allowing complex datasets to be displayed in context with the underlying physical landscapes.

Most commonly, the goal in modern scientific visualizations is to show the modeler stages of their model while it executes, i.e. by depicting the data as it is produced by the model. This sort of real-time feed back helps the modeler interact with the model like never before. The effective use of this technology is the focus of the project described here.

1.2.2 Presentation Visualizations

Visualizations created with a broader audience in mind than just modelers, i.e. for *presentation*, were inspired by early efforts in scientific visualization combined with various hardware and software advances. Presentation visualizations are designed to communicate both specific ideas or results, and to set the specific context in which the ideas and results should be interpreted. There is a distinction between these goals and simple “modeler support.” Visualizations created by and for a modeler might display only a few critical attributes in a way to help a specialist in that field understand what is happening. In contrast, a presentation visualization must provide a much “broader” display, to acquaint a less-expert viewer with the phenomena under study.

For example, if a specialist is creating a presentation visualization for model results from a certain water basin, the ability to give some context for the model by showing images of the region in question or flying them over the region can help in the general understand for the audience. If the audience can be assumed to be familiar with the region of interest, the complete context does not need to be set and the “background” information can be minimized. However, for a more general audience, more extensive context setting and “background” information is required, and it’s not very practical to take everyone up in a plane and fly them over the region of interest. A presentation style visualization can bring the region of interest to the audience quickly and effectively. This technique has evolved to become an inherent part of data presentation and dissemination in practically all applications.

Chapter 2

Original Pass (paradigm shift)

2.1 Background and Motivation

The origin of this project is the need to support data analysis, modeling, presentation, and results dissemination in a petroleum reservoir characterization project. Project scientists include field geologists with “field observed” data, geophysicists with 3-D seismic datasets, and petroleum engineers with various forms of simulation modeling datasets. Initial goals include both modeling support and presentation/dissemination of data from all parts of the project.

Initial work on Spiral One focuses on providing petroleum engineer Dr. Tarek Ahmed, of Montana Tech of the University of Montana, and geologist Dr. Karen Porter, of the Montana Bureau of Mines and Geology, tools that combined their model results to allow further study and presentation. The region selected for their characterization project was the Rabbit Hills oil field near the town of Chinook in north-central Montana.

2.2 Visualization Pipeline Components

2.2.1 Base Spatial Data Collection

The first goal is to create a “virtual landscape” based on actual data. As illustrated in the details in Figure 2.1, creating the original visualization involves gathering the appropriate data, figuring out how to import, color, render and

display it. The DEM (Digital Elevation Model) of the Rabbit Hills area is used as a base input layer to **DX**. The wells are positioned on the landscape from their rectangular survey system coordinates. The well positions must be translated into the same scale and coordinate system of the DEM with the use of a very specialized converter. Note that the “landscape” at this point transformed from a surface depiction (often called “2.5-D”) to a full 3-D model in which well depth is an important attribute.

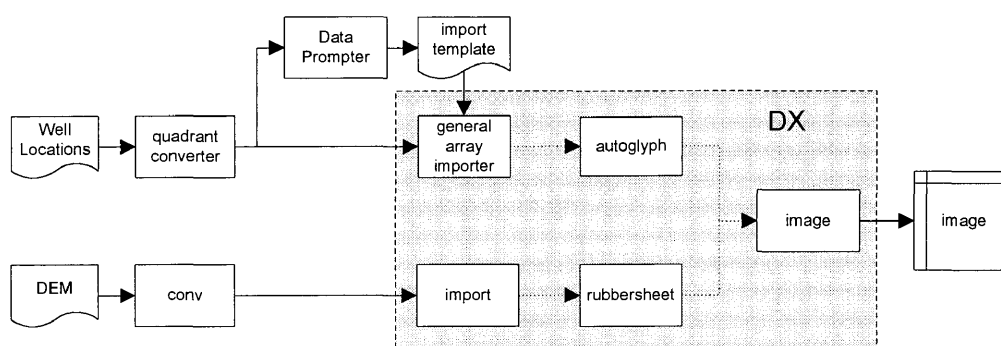


Figure 2.1 Base Input Data Importing and Verification

In combining the base DEM dataset with another dataset, such as well locations, the importance of coordinate systems and transformations, i.e. projections, cannot be understated. The visualization expert and scientist must specify the coordinate system, scale, and resolution of elements very accurately in order to standardize the spatial correspondence (cartographic fidelity) of the resulting visualization. However, because of the different standards used by geographers, surveys, and other data contributors, datasets generally need to be converted and coregistered to a single standard. Once the two base datasets are combined, placing wells accurately on the surface, initial images can be rendered for inspection by Dr. Porter. When she is satisfied that the well locations have been accurately translated the process continues.

The **quadrant converter** [Holbrook93] is a facility required in this application to take the location of the Range-Township corner location in the same reference system as the DEM layer, the table of well positions in rectangular survey coordinates, then combine these inputs and produce the list of well positions in the same reference system as the DEM. Generally, the user must first convert the position of the lower left corner of the Range-Township quad, the width and the height into the same coordinate system as the DEM. The goal for the **quadrant converter** is to input a series of well positions in the rectangular coordinate system and produce the positions of the wells in the same coordinate system as the DEM, and in a form which can easily be imported into the **DX** environment by the **general array importer**. The nature of the rectangular coordinate systems leaves the location defined within the extent of the smallest region. The equation in Figure 2.2 shows how the wells were placed at the center of the final rectangular region with the final term for both x and y.

$$\begin{aligned}
 & \text{Range - Township Location } (x_0, y_0), \quad \text{Width } (w_x, w_y), \\
 & \text{Rectangular Surveys } S[k \text{ elements}] = \{YX, \dots, YX\}, \quad Y \in \{N, S\}, \quad X \in \{W, E\} \\
 & \text{and } S_x = \{X, \dots, X\}, S_y = \{Y, \dots, Y\} \\
 \\
 & x = x_0 + w_x \left(\sum_{i=1,k} \frac{1}{2^i} \cdot (S_x[i] = E) + \frac{1}{2^{k+1}} \right) \\
 & y = y_0 + w_y \left(\sum_{i=1,k} \frac{1}{2^i} \cdot (S_y[i] = N) + \frac{1}{2^{k+1}} \right)
 \end{aligned}$$

Figure 2.2 Quadrant Converter Equations

In the example illustrated in Figure 2.3, the well's position in rectangular survey coordinates is [NW NE SE NW SE].

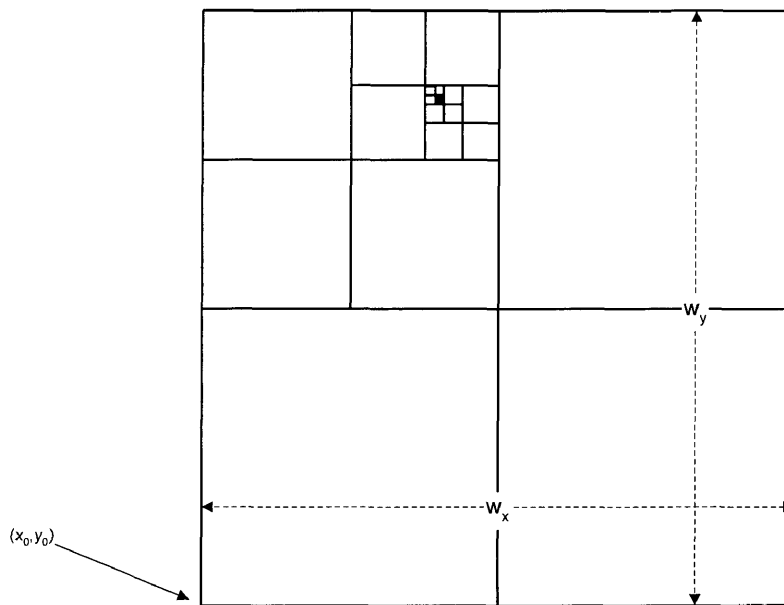


Figure 2.3 Range-Township Example

If the location of the corner of the Range-Township is (1500, 2000) and the width and height of the region is 2000, the location is:

$$\begin{aligned}
 x &= 1500 + 2000*(0.0+.25+.125+0.0+.03125+.015625) = \\
 &1500 + 2000*(.421875) = 2343.635 \\
 y &= 2000 + 2000*(.5+.25+0.0+.0625+0.0+.015625) = \\
 &2000 + 2000*(.828125) = 3656.25
 \end{aligned}$$

The **quadrant converter's** limited scope makes it essential for this part of the project, but once the locations are figured out, it isn't used again until the location of a well is modified following a field verification. Specialized parsers and converters like the quadrant converter are needed regularly to import data from many sources. Model data is generally formatted for human consumption which almost guarantees that the computer will have difficulty dealing with it.

Once the well positions are ready for import, the next step is to convert the DEM to a DX data format. For this, the **conv** [Thompson95] utility created by Dick Thompson and David Thompson at the University of Montana is used. This spatial dataset utility allows for the conversion of many formats, including, DEM, DLG, and various GIS related formats.

The next step is to import base data into the visualization environment, **DX**. The **DX** programming environment is a visual-programming environment where the user has an application canvas and a tools palette. Operations are represented by rectangles with input and output tabs, referred to as modules. The data flow between modules is represented by line connections between operations.

The **Data Prompter** [IBM95] is a facility which is included in the standard distribution of **DX** to allow a user to define import templates for regularly formatted data files. These templates are used by the **general array importer** to import the data into the visualization programs. As illustrated in Figure 2.1, the specific goal is to import the base data, transform it appropriately, and render it to present an appropriate image.

Once a general array importer description is built, the converted DEM data is acquired via the **Import** module. The landscape surface is created from a **RubberSheet** of the DEM data colored in a false color brown representing the grasslands. The **RubberSheet** module creates a surface over a data landscape with elevation controlled by the data value at each point. The position of the wells is initially shown with the **AutoGlyph** module which creates a sphere for a simple scalar data value in a 3-D landscape. For added realism, glyphs are created to resemble oil derricks. They are placed on top of the landscape at the position of the wells.

Once the region of interest is created and the glyphs are placed on the surface, still images are created for inspection. In addition, the interactive view control mode of **DX** is used to investigate the virtual landscape. Through Dr. Porter's input, it became immediately evident that moving over and around the surface would be a great benefit to reveal the location of all the wells and the other characteristics of the landscape.

2.2.2 Camera Paths

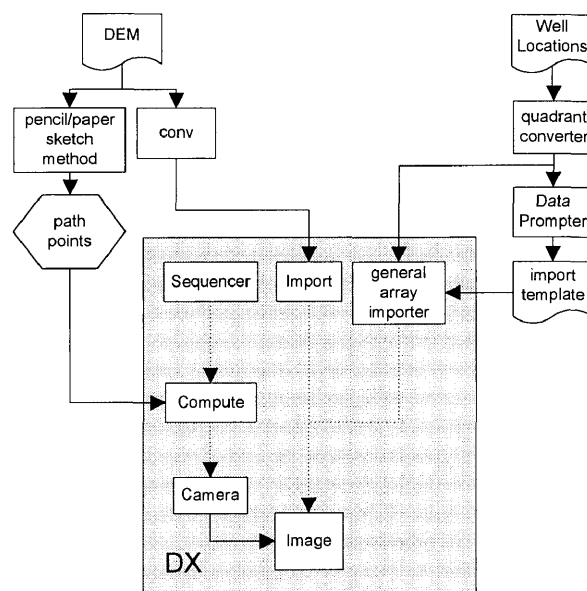


Figure 2.4 Single and Multi-Segment Camera Paths

2.2.2.1 Single Segment

The first approach to depicting motion through a landscape was a very primitive flight path. The **Camera** module operates given a set of values including; camera position ("from"), a position to look towards ("to"), viewing angle, resolution and aspect. The position of the camera can be changed in a series of steps along a linear path that passes parallel to one of the sides of the Rabbit

Hills Quadrant. In all cases, the position of the “to” point is set to the center of the region. The **DX** environment provides a set of tools called *Interactors* for controlling variables. Interactors allow the user to interact with the visualization applications through dialog boxes and a special device called a **Sequencer**. The **Sequencer** increments a value and has controls like a VCR or CD-Player. (Figure 2.5)

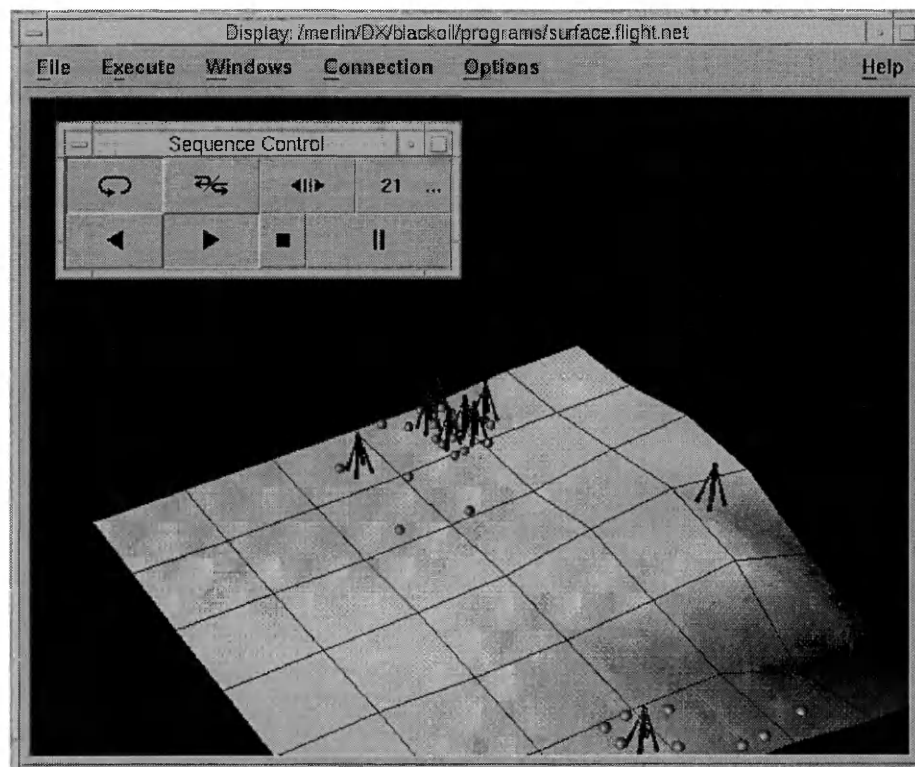


Figure 2.5 Sequencer and Surface

2.2.2.2 Multi-segment Flight Path

To create a more ambitious path, a series of line segments can be linked together to create a flight path around the entire region of interest. To construct such a path, the region of interest is sketched on paper and the path drawn in relation to it. The values for the positions of the control points are estimated

using the position and size of the Range-Township. **Compute** modules can be used to determine which of the segments the camera is currently traversing according to the equations of Figure 2.6. The **Compute** module applies a user specified mathematical expression to it's input and passes the returned value as output. The flight path spirals around and into the region of interest and “flies” down near the surface between a few of the oil derrick glyphs. The flight comes to an end near the middle of the region as if the plane has landed. The apparent motion is smooth while the camera travels along each linear segment, but the images experience a large change when crossing from one segment to another. If the line segments are almost parallel, the change at the control point is less dramatic.

Inputs: list $\{P_0, P_1, P_2, \dots, P_n\}$ [n+1 elements], ω [steps/segment], ρ [current step]

Output: current Position P_{step}

Let $\delta = \text{INT}(\rho / \omega)$ {Current segment RANGE = 0...n-1}

Let $\kappa = \rho - \delta * \omega$ {Current position on segment RANGE = 0... ω -1}

$$X_p = (X_{\delta+1} - X_\delta) / \omega * \kappa + X_\delta$$

$$Y_p = (Y_{\delta+1} - Y_\delta) / \omega * \kappa + Y_\delta$$

$$Z_p = (Z_{\delta+1} - Z_\delta) / \omega * \kappa + Z_\delta$$

Figure 2.6 Multi-Segment Determination Equations

2.2.2.3 FlyingCamera Macro

The size of network which makes the camera move along the multiple line segments makes it impractical to use in other visualization programs. DX's macro facility is the ideal tool to package the flying camera into a more universally useful object. The **FlyingCamera** [Holbrook94] macro encapsulates the large network and lets the user select a list of control points and the number of points the camera should visit between control points. The macro is used inside a visual network in Figure 2.7.

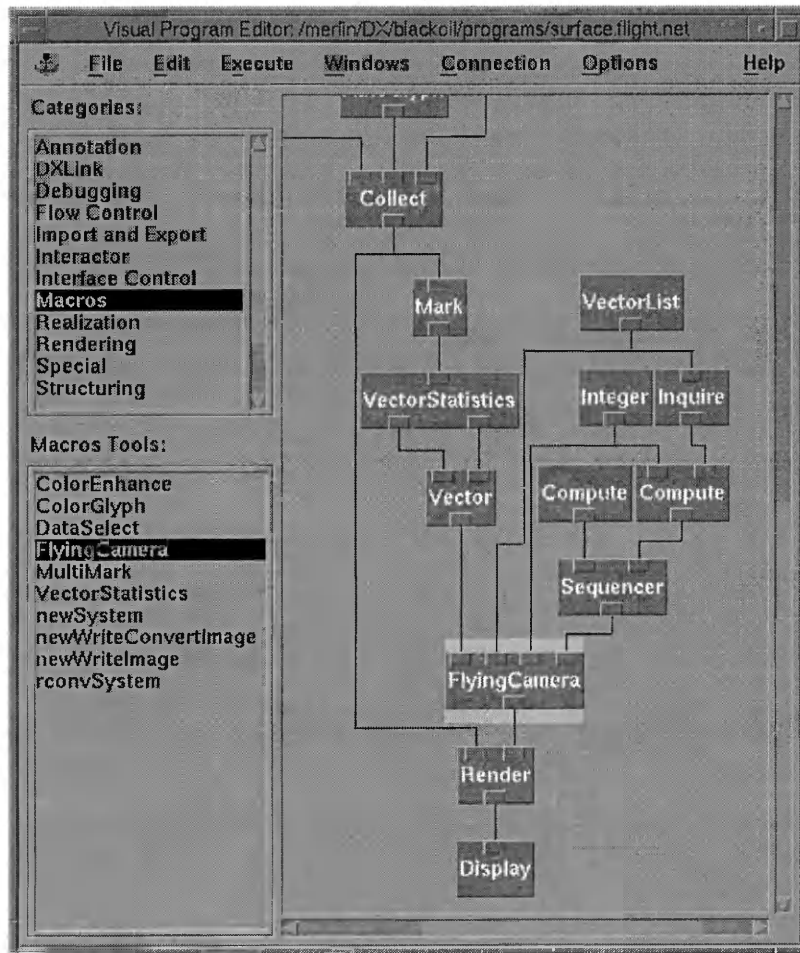


Figure 2.7 FlyingCamera Macro in Visualization Program

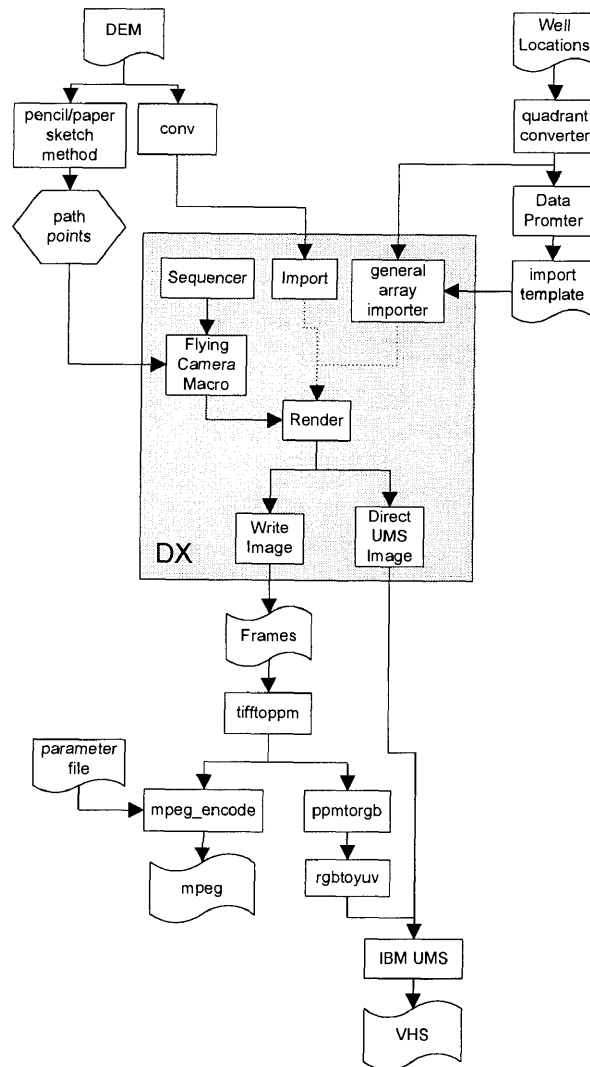


Figure 2.8 The First Movies

2.2.3 The First Movies

2.2.3.1 DX Image Caching

When the **Sequencer** is used with an **Image** or **Display** module, a series of images can be rendered, cached and replayed without re-rendering them. The

available memory and CPU have a large effect on the performance of the original rendering and subsequent caching. If the number of frames in the series is small enough to completely load into memory, the image caching feature is effective, but the initial rendering pass must be done at least once in any case.

The limit on the number of frames and the speed of the image refresh are the next things needing improvement. The dataset is already small and the region of interest cannot be reduced to help speed the image refresh rate. The size of the image window cannot be reduced without losing the amount of information and viewing area brought to the audience. This evaluation suggests the need to find some way to create a self-contained “movie” of the frames.

2.2.3.2 MPEG Movies

The MPEG is a standard movie format which can be viewed on UNIX workstation's, Windows PC's and MAC's. The basic scheme of MPEG movie compression is to predict motion from frame to frame in the temporal direction. In other words, the compression scheme looks for a close match to a block of pixels in the current frame with that of a future frame, and moves it towards the position in the future frame. Each of the input frames becomes a control frame. A user-controlled quantity of interpolated frames are generated between the control frames to move the similar blocks to their next position. The MPEG compression scheme is a lossy process where the integrity of the interpolated frames is not guaranteed.

The **mpeg_encode** [Gong94] utility transforms a series of images into an mpeg movie. The images rendered in **DX** and saved using the **ImageWrite** module are created in TIFF image format. The **mpeg_encode** utility requires a parameter file identifying the input frames and compression method, and the

ppm (portable pixmap) images as input. The **PBMPLUS** (Extended Portable Bitmaps Plus) **Toolkit** [Consortium91] was discovered through Internet at the same location as **mpeg_encode**. The **PBMPLUS Toolkit** contains an extensive set of image converters for most image formats. **tifto ppm**, one of the tools in the set, is used to convert the **DX** created TIFF's to ppm images.

The resulting movie is a great improvement, in the sense of smoother motion and increased limit on the number of frames allowed. However, the image refresh rate is still affected by image size and machine performance. The amount of loss between control frames can be dramatic depending on the image content, e.g. the grid lines surrounding the Rabbit Hills landscape are not interpolated very well.

2.2.3.3 IBM UMS card

A "beta" (pre-commercial) version of the UMS video card for the IBM RS6000 provides the opportunity to experiment with direct output to a TV or VCR. The UMS card comes with a primitive library set of functions for passing images in memory through the card to an attached output display or capture device.

In the first method attempted, the series of images was converted from tiff to ppm to rgb to composit yuv frames for the UMS card using utilities from the **PBMPLUS Toolkit**. The entire sequence was loaded into memory and then passed frame by frame to the UMS card. The movie results were fast, but limited in length by the amount of memory available. There were also serious scanline problems and flashes at the start and stop.

In the second and final method with the UMS video card, a module was created which did the image translation and shipped the images directly to the UMS

video card from within **DX**. Once again we were faced with the same problem that inspired us to go to MPEG movies. The entire sequence needed to be rendered and cached before the animation could happen.

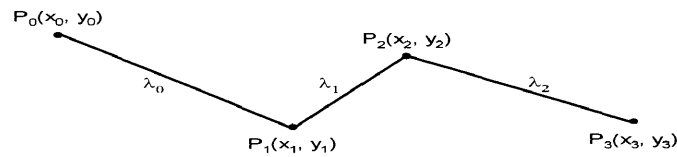
Chapter 3

Flight

3.1 The best way between two points isn't always a straight line

Creating smooth motion in an animation depends highly on creating smooth motion for the "camera" in the visual-animation space. The **Camera** module works in conjunction with a **Render** module to create an image with a specific orientation, perspective, viewing angle, resolution, aspect etc. The orientation of the camera is specified by the "to", "from" and "up" parameters. The "to" and "from" parameters are the locations to look towards and from respectively. The "up" parameter is the vector direction normal specifying the planar orientation of the image. Assigning a series of values to any of the camera parameters results in an image change, e.g. the "from" parameter changing on a path results in apparent motion.

The camera motion during the first spiral pass was controlled by a collection of linear segments with connecting control points. The resulting motion is realistic in only the case where there is only one linear segment.



$$\lambda_0 = \frac{y_1 - y_0}{x_1 - x_0} \quad \lambda_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

Figure 3.1 Segmented Linear Paths and their Slopes

The realism is lost as soon as there are two non-linear line segments because the first derivatives (slopes) of those segments are not the same. The viewpoint of the camera changes dramatically when leaving the end of one segment and beginning another. If one considers the two dimensional example shown in Figure 3.1 the slope is λ_0 while traversing from P_0 to P_1 and is λ_1 from P_1 to P_2 . The slope transition between the segments is $\Delta\lambda = |\lambda_1 - \lambda_0|$. $\Delta\lambda$ is a finite and results in a perceivable change.

The goal of this spiral pass is to remove the perceivable changes between frames. If the slopes match at the control points, the camera view will not change when crossing control points. In mathematical terms, the first derivatives need to be matched to achieve continuity along the path resulting in a smooth "flight". That is, a curve needs to be fit to the control points. With a curve, the slope changes constantly with a very small $\Delta\lambda$ as illustrated in Figure 3.2.

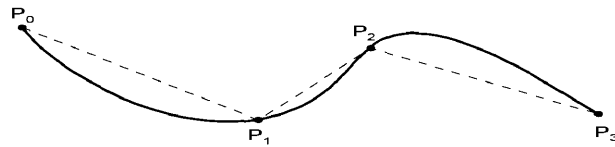


Figure 3.2 A Curve Matching Control Points

Controlling the path of the camera with the mathematical equation like that of a circle provides a smooth path. The data visual-animations are seamless and smooth but constrained to the path along the circle, eclipse or regularly defined curve. If we need to move off the circle, we need a more general solution.

3.2 Visualization Pipeline Components

3.2.1 General Path Interpolation - “Smoothing the Savage Beast”

We examine two methods for matching a curve to an ordered set of points. The set of points can be matched with a single curve encompassing the entire ordered set, or the derivatives can be matched at each control point. Each method results in a mathematical solution to the curve fitting problem.

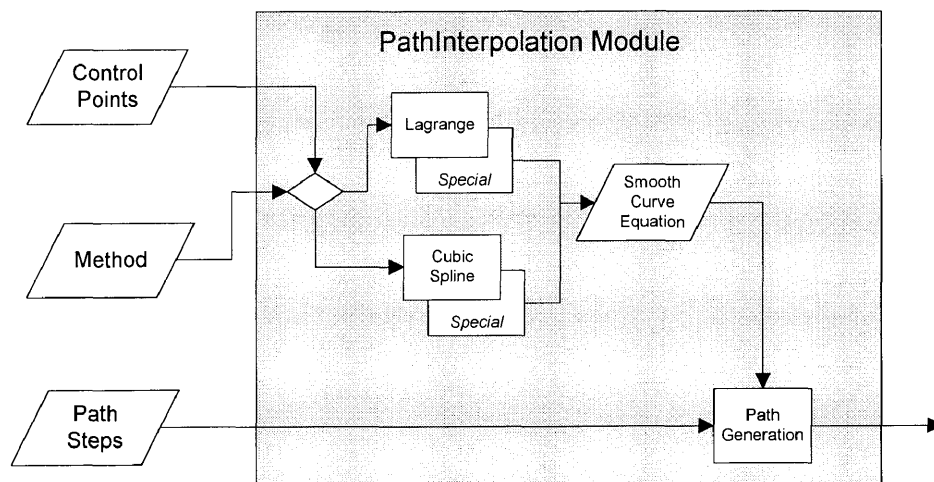


Figure 3.3 General Path Interpolation Overview

3.2.1.1 Lagrange

The Lagrange method fits a single order $k+1$ curve to any set of k control points.

$$\text{Control Points} = \{P_0(x_0, f_0), P_1(x_1, f_1), P_2(x_2, f_2), \dots, P_k(x_k, f_k)\}$$

$$L_i(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_k)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{k-1})(x_i - x_{k+1}) \cdots (x_i - x_k)} = \prod_{\substack{j=0 \\ j \neq i}}^k \frac{(x - x_j)}{(x_i - x_j)}$$

$$p_{\text{Lagrange}}(x) = f_0 L_0(x) + f_1 L_1(x) + f_2 L_2(x) + \cdots + f_{k-1} L_{k-1}(x) + f_k L_k(x)$$

Figure 3.4 Lagrange Curve Interpolation Equations

Although the resulting curve passes through each control point correctly, it has extremely high minimums and maximums, as illustrated in Figure 3.5.

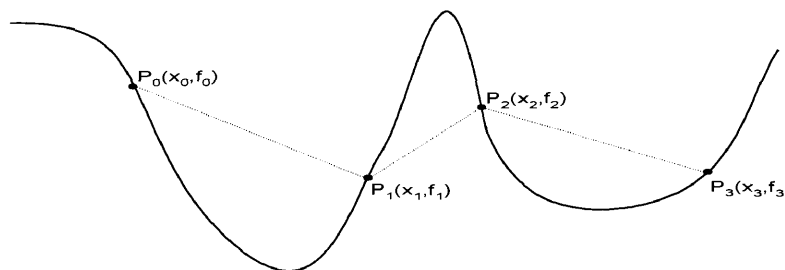


Figure 3.5 Lagrange Curve

Since the camera is going to move along the curves in a finite number of steps, the amplitude peaks and troughs along the path should be minimized. To do this, extra control points $P_0(x_0, f_0)$ and $P_3(x_3, f_3)$ are added beyond both ends of the path in order to “pull” the curve tight as in Figure 3.6. This “Special Lagrange” curve is well behaved compared to that of the standard Lagrange interpolated curve.

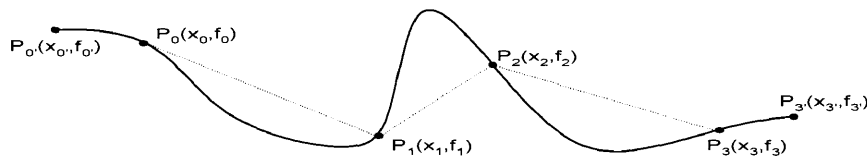


Figure 3.6 Special Lagrange Curve

3.2.1.2 Cubic Spline

The spline family of curves produces a piece-wise solution to a set of control points. Smooth transitions across the piece-wise curves are achieved by matching the slopes at each control point. The cubic spline is a 3rd order piece-wise solution to this problem.

These curves match two control points and the end derivatives. In addition, the derivatives at the control points are matched such that they minimize the magnitudes of the min's and max's. Solving the cubic spline involves solving a system of four equations with four independent variables for each piece-wise curve segment.

$$\begin{aligned}
 at_i^3 + bt_i^2 + ct_i + d &= x_i \\
 at_{i+1}^3 + bt_{i+1}^2 + ct_{i+1} + d &= x_{i+1} \\
 3at_i^2 + 2bt_i + c &= \frac{x_{i+1} - x_{i-1}}{t_{i+1} - t_{i-1}} \\
 3at_{i+1}^2 + 2bt_{i+1} + c &= \frac{x_{i+2} - x_i}{t_{i+2} - t_i}
 \end{aligned}$$

Figure 3.7 Special Cubic Spline System of Equations

The first two equations of Figure 3.7 match the third order solution at the control points. The second two equations match the derivatives at those points.

The slope of the curve at every control point is set to a value equal to that of the line segment connecting the points on either side of the control point in question. In Figure 3.8, the slope of the curve at P_1 is set to λ_1 , the slope of the line connecting points P_0 and P_2 . The slope of the curve at P_2 is set to λ_2 , the slope of the line connecting points P_1 and P_3 . The special cases at the beginning and end of the curve are solved by setting the slope to a value half way between the next two points in the case of the beginning and the previous two points in the case of the end of the curve. Setting the slope in this way minimizes the min's and max's of the curve connecting the points by forcing the curve to remain within the bounds of the control points.

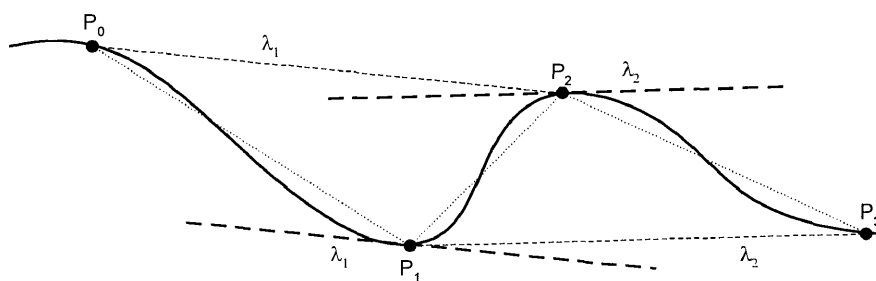


Figure 3.8 Special Cubic Spline Min/Max Reduction

3.2.2 The PathInterpolation Module

The Special Lagrange and Special Spline solutions can be extended to 3-dimensions through parameterization with time as the independent variable. This 3-D extended version has been integrated into the Data Explorer environment in the form of the new **PathInterpolation** [Holbrook94] module as shown in Figure 3.9. The user provides inputs that select which method to use, the list of control points, and the list of values that represent the number of steps between control points. In addition, the user can also specify a value which

controls the distance between the end control points and the additional points used to “stretch” the curve tight.

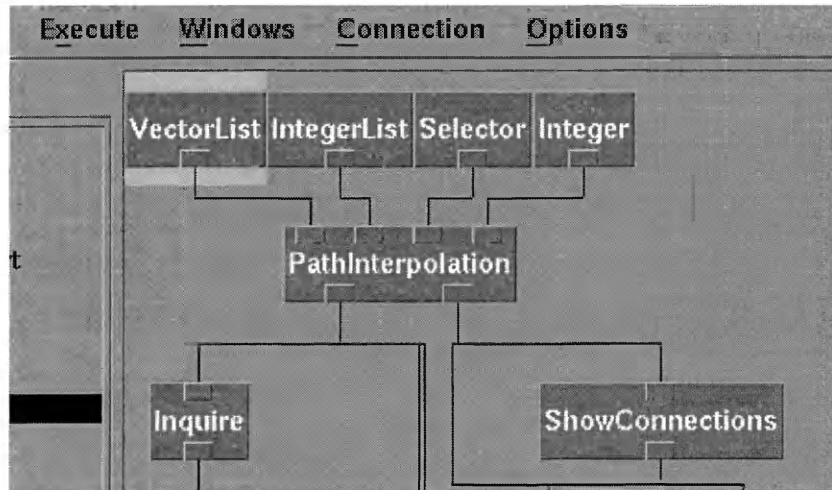


Figure 3.9 **PathInterpolation** Module

Although the calculation of the flight path is a small operation compared to rendering the frames after the path is created, algorithm performance does affect the amount of time it takes to create and fine tune a flight path. Fortunately, the best behaved solution is also the most efficient. In a path with p control points and s steps between control points, the complexity of each of the two computations is:

$$\text{Lagrange} \rightarrow \theta(ps^2 + p^2)$$

$$\text{Spline} \rightarrow \theta(ps)$$

For the smoothest video there are as many frames as possible between control points, so we can assume $s \gg p$.

$$\text{Lagrange} \rightarrow \theta(ps^2)$$

$$\text{Spline} \rightarrow \theta(ps)$$

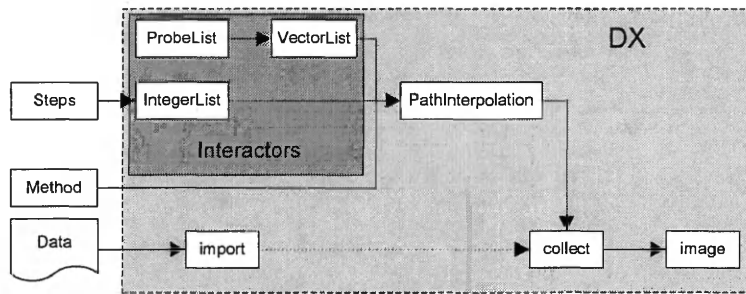


Figure 3.10 Path Designation Overview

3.2.3 Defining The Path

Once the flight paths are created, the data visual-animations produced by **DX** are quite promising. However, creating the paths manually turns out to be a near impossible task. The pencil and paper sketch method for creating the flight path control points is not easily extended into a real 3-D object. For example, the 3-D Rabbit Hills model has above ground relief, subsurface strata, and well drops, making it hard to predict exactly what path(s) are the most interesting. There needs to be an interactive way to create the flight path. Previously, the flight path was sketched on paper, but it is not really tested until the entire series of images is rendered. There is no way of knowing if the flight path is going to “collide” with an object in the landscape without rendering the whole series. For example, the oil towers on the surface can create problems in the Rabbit Hills visual animation when the flight path comes in to meet the land.

Thus, the next required tool is one that supports interactive designation of the flight path. By adding control points through point and click, modifying the flight path with drag and drop actions, and displaying the flight path as a trajectory line in relation to the visualization object, all prior to image sequence rendering, the task of creating flight paths is greatly simplified.

The generality of this problem does not lend itself to the creation of a self contained path-programming macro; it requires a collection of modules and a network framework that can be integrated into an existing spatial visualization. The critical parts of the “path programming” framework include the following: the **PathInterpolation** module; a **Probelist** module for on screen control point creation and modification; a **Vectorlist** interactor panel for display and manipulation of control point location values; the original visual object network or problem bounds; a **Collect** module to collect the flight path with the original object; and at least one **Image** module. An example of the “path programming” minimum network requirements is shown in Figure 3.11.

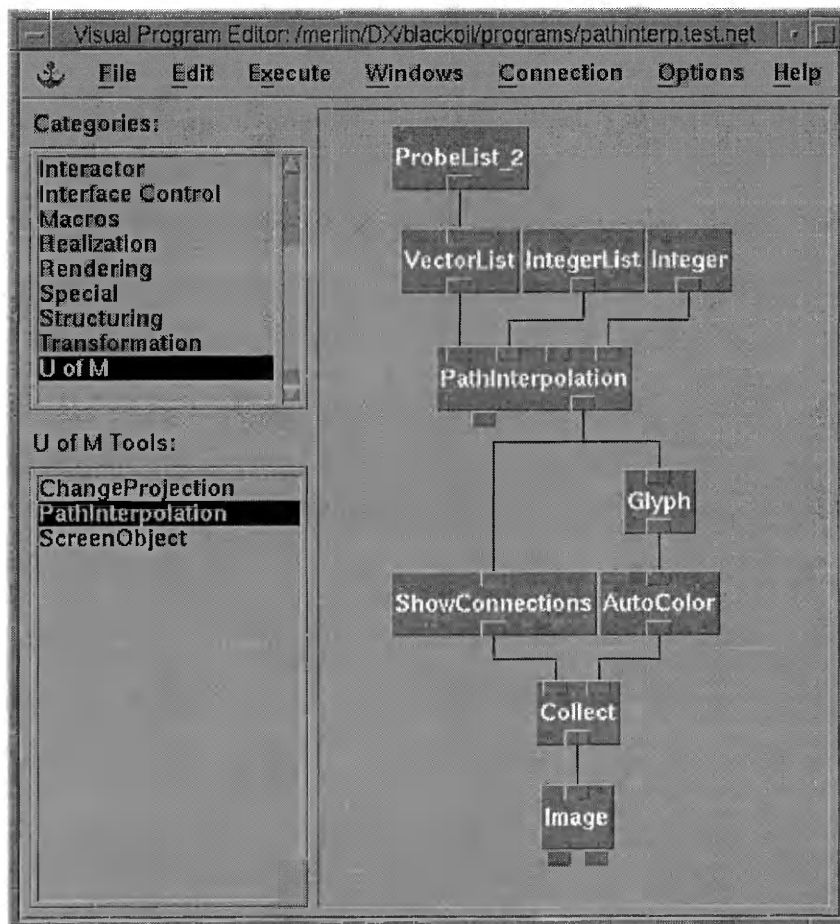
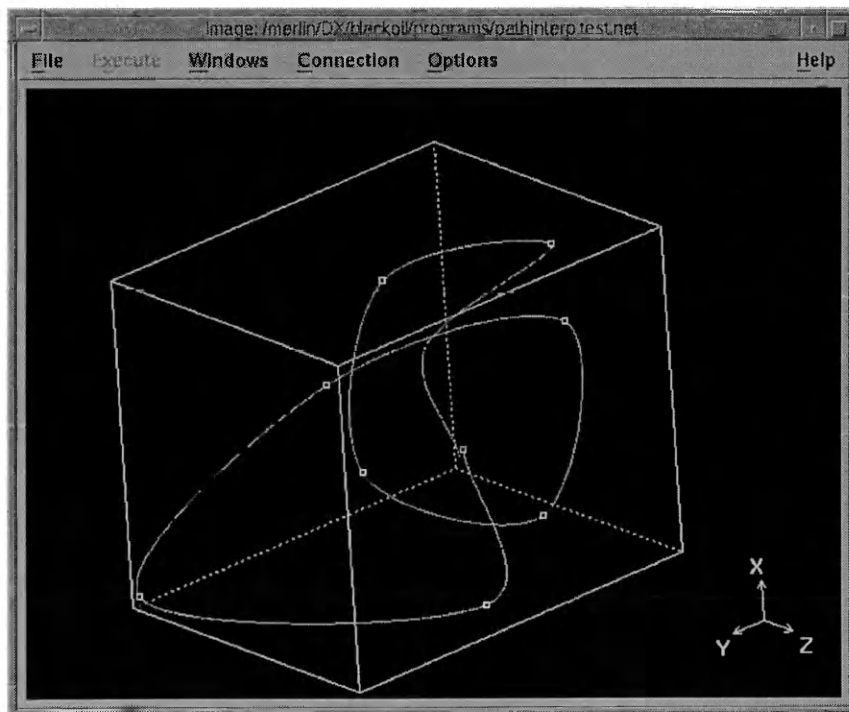


Figure 3.11 Path Programming

The **Probelist** module works in connection with an **Image** display window, allowing the user to specify points in space by selecting a position with the mouse. The **Vectorlist** interactor takes the point list from the **Probelist** module and displays the numerical values of the positions of the points. The list of points and their positions can be altered using the interactor dialog box, enabling the user to edit and insert points which might have been missed. The output of the **Vectorlist** interactor is passed to the **PathInterpolation** module where the actual flight path is created. The secondary output of the **PathInterpolation** module is an object formed by connecting all the path points in order. This path object is then annotated using the **ShowConnections** module and displayed. The **PathInterpolation** module also assigns data values to each point according to whether or not it is an interpolated point or a control point. This allows each point to be displayed as a sphere with **Glyph**, then colored to show its location on the path and in relation to the original data visualization object. In Figure 3.12, the control points are shown in respect to the interpolated path.



3.12 Path Manipulation

Once a position is created, it is moveable with two degrees of freedom relative to the current perspective. For this reason, multiple image windows with different perspectives allow the user to move the control points in all three dimensions, i.e. from above and from the front as illustrated by Figures 3.13-3.14. Moving the control points around in the visualization space in this manner can still be a tedious job, but it is a vast improvement over the previous pencil/paper/sketch method.

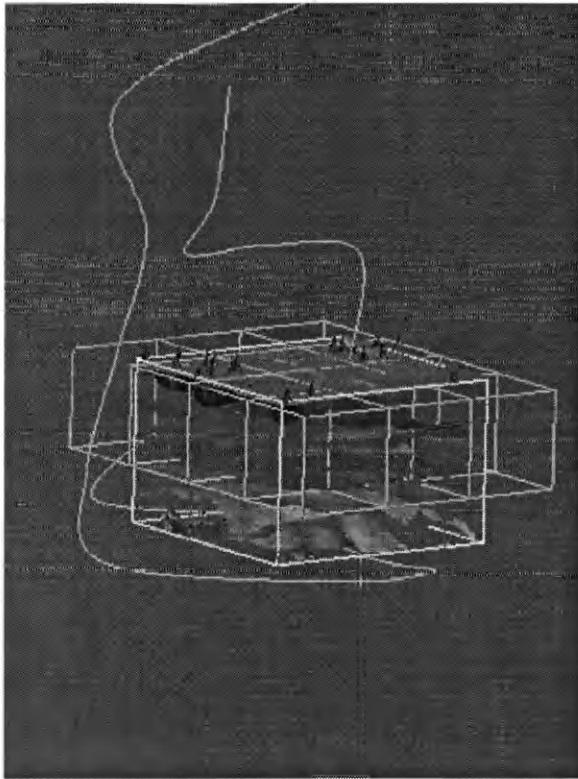


Figure 3.13 Path View 1

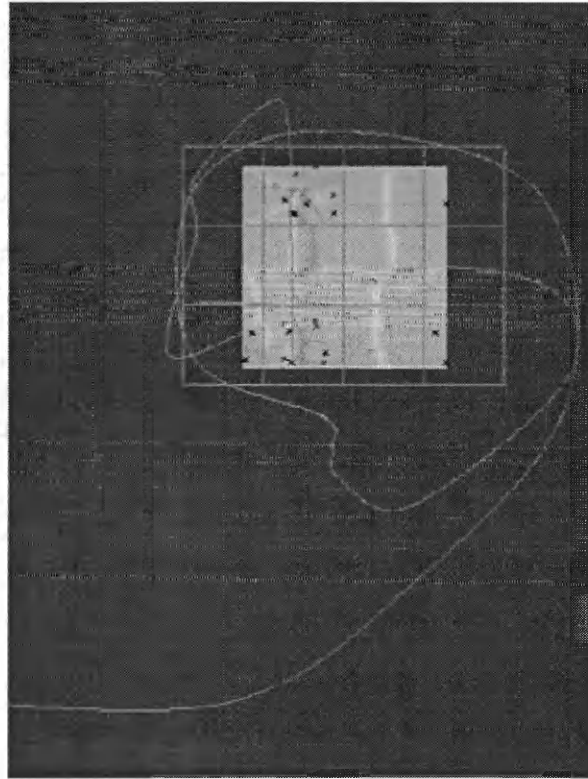


Figure 3.14 Path View 2

3.2.4 Multiple Paths / Network

In this and the remaining example the resulting landscapes are very complex 3-D structures. In such a structure, we find that a flight path with the view looking straight ahead the entire time is often ineffectual at showing all the object's interesting features. The best analogy for the solution is to consider a passenger

who needs to look from side to side to get the best observations while flying through interesting scenery. This type of camera movement can be accomplished by creating another path for the “look to” variable of the camera. This multiple path method can also be extended to control the other variables, e.g., changing camera “width” and perspective for effects like smooth zooming in and out.

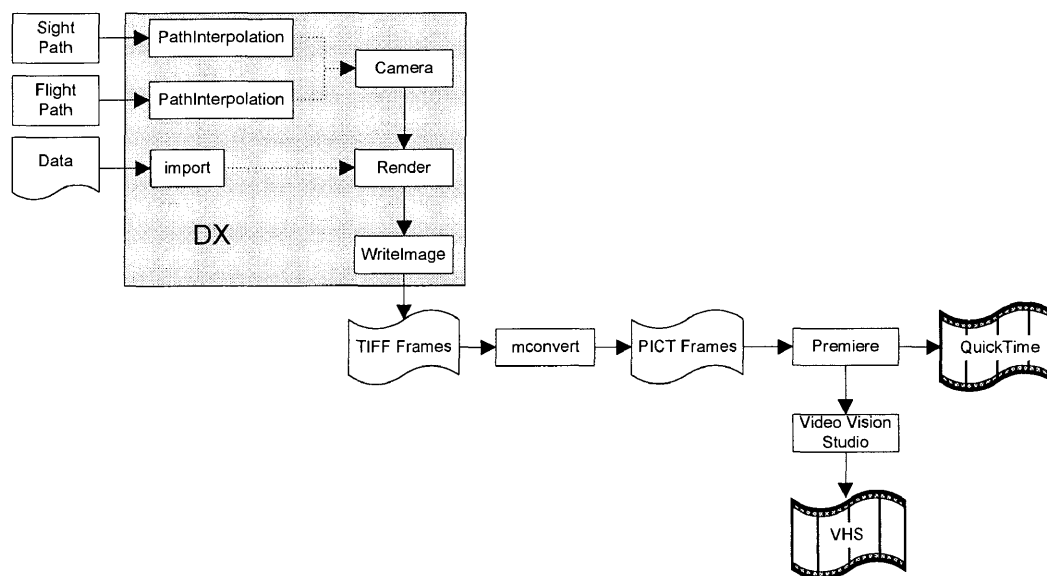


Figure 3.15 Video Revolution Overview

3.2.5 Video

The evolution of the smooth camera path and path programming techniques opens the door for more complex and lengthy animations. The new animation complexity and the lossy characteristics of the MPEG movies created in the previous spiral suggests there is a need to find a better movie solution. **QuickTime** [Apple94] is a standard video movie environment originally developed by Apple for the MAC platform, but also now supported for Window’s PC’s as well. The QuickTime environment supports audio and video with multiple software and hardware based compression schemes. The acquisition of

new hardware in the form of two PowerMac 8100's with the **VideoVision Studio** [Radius95] adapters and a variety of new software provided the ability to create Quicktime movies. The **VideoVision Studio** adapter is a hardware compression based NTSC video input and output device allowing for the production and capture of broadcast quality video.

Premiere [Adobe94] is the movie editing and creation software used to assemble the frames and audio sequences. **Premier** comes standard with a large selection of software compression schemes. The Video Vision Studio hardware compression becomes a option when the adapter is present in the machine.

After all the frames are successfully created on the RS/6000 workstation, they are converted to the PICT image format by the **mconvert** [Thompson95] utility. **mconvert** is designed to take a desired image type and a list of target files as input and convert every image to the desired type using the **convert** utility from **ImageMagick** [Cristy92] set of imaging tools. These PICT images are moved to the PowerMac via the **fetch** [Dartmouth95] ftp utility and loaded into **Premiere**. The user selects the desired compression scheme and lets **Premiere** create the movie.

Chapter 4

Looking at the World through AutoColored Glasses

4.1 More Realistic Landscapes

At this point, the motion of the camera flying around and through objects is very realistic. The camera has the ability to look from side to side or zoom into an area while moving over the landscape. The problem now is finding colors for the landscape which help set the context and do not distract the viewer.

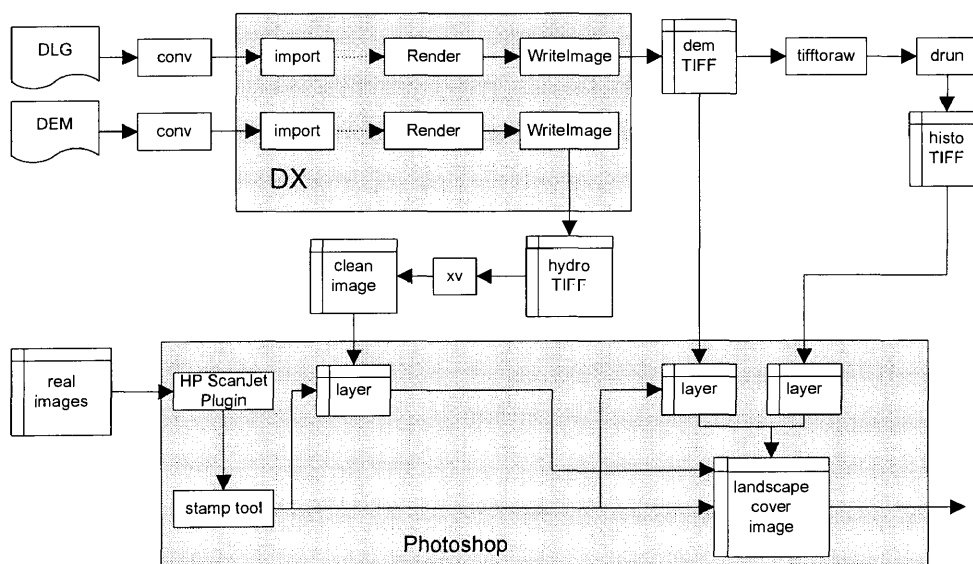


Figure 4.1 Landscape Layer Creation

4.2 Visualization Pipeline

The introduction of the Glacier project uncovered a set of deficiencies that needed to be addressed during the next iteration. The purpose of the Glacier project was to set the context for a watershed study of the Lake MacDonald basin. The DEM was the first data layer acquired and it was imported and turned into a surface with **RubberSheet**. Unlike the Rabbit Hills area, Glacier National Park is one of the more geologically dynamic regions on the planet. Initially, the **ColorMap** module was used to color the landscape according to elevation. The tops of the peaks were colored white to represent snow and greens and browns colored everything below. A flight path through the region was created and frames for the animation were created, but it was obvious the colors weren't quite right. The colors created were not the same as "natural" greens and browns. Also, the data values representing elevation don't necessarily correspond to the location of natural objects like snow fields, rock faces, tree growth and water. In order to produce the desired effect, the landscape needs to be painted with a much more natural color palette.

4.2.1 Laying an Image over a Landscape

After considering a number of approaches to generating a natural color palette, I realized that a much simpler solution would be to "borrow" the coloring from real photographic imagery. The first part of the solution to this problem is to lay such an image over a surface. The **ReadImage** module will currently import rgb, TIFF and gif images, allowing it to import the composite image created to cover the landscape. Once again, it is critical that the dimensions of the image and the landscape are identical. This makes laying the image over the surface possible

with the use of the **Replace** module which replaces a specific data field in one object with that of another.

4.2.2 Creating Input Layers

The ability to lay an image over the top of the surface opened the door for “custom” landscape coloring. The key to creating a realistic landscape is to base the overlay image on as many of the natural features as possible. The Glacier Park overlay was based on the hydrology, elevation and actual pictures of the region. The hydrology was obtained from the USGS DLG (Digital Line Graph) data which includes marshes, swamps, lakes, rivers and glaciers. The **conv** utility was used to import the DLG into the Data Explorer Environment. The DEM in Figure 4.2 provided the elevation data and was used to identify peaks, slopes, benches and saddles. After the layers were verified in **DX**, they were exported as TIFF images.

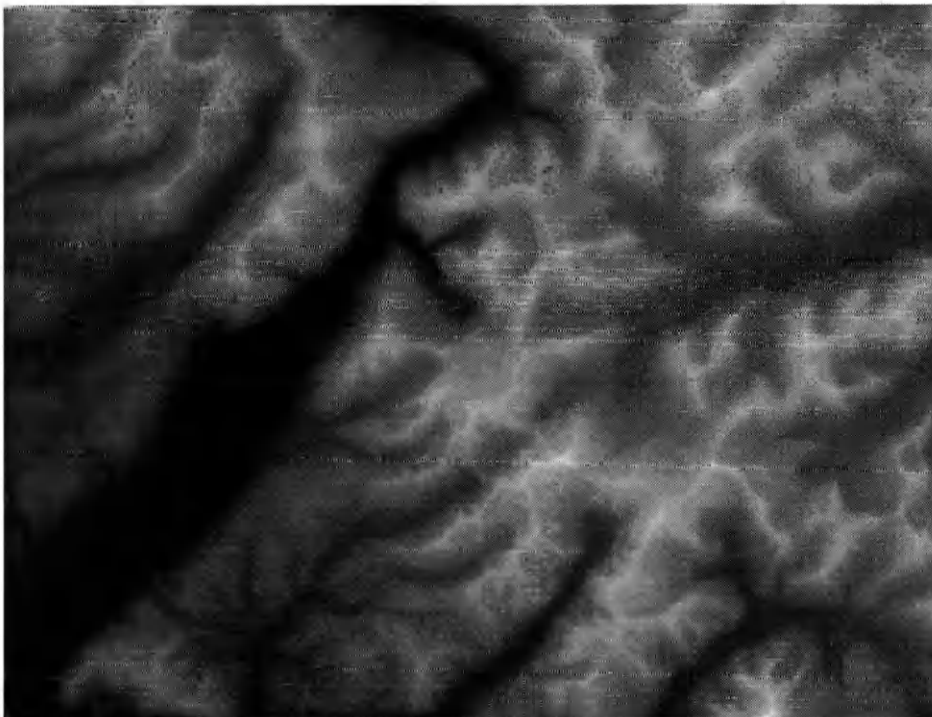


Figure 4.2 Range of DEM values

The hydrology image was edited with the image utility `xv` [Bradly94] to remove the marshes and swamps and clean up the image as illustrated in Figure 4.3.

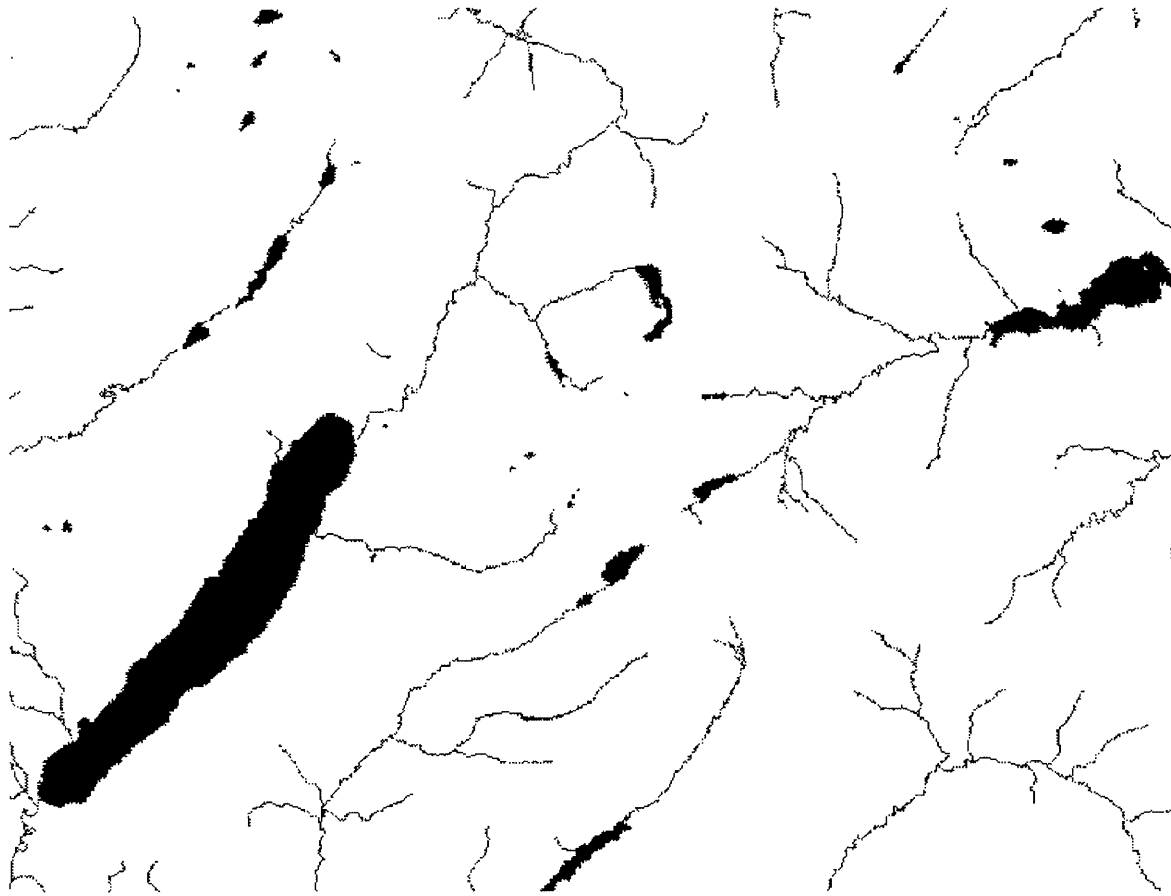


Figure 4.3 Hydrology Image Layer

The DEM image was converted to raw data using the **tiff2raw** converter from the **PBMPLUS Toolkit** then it was converted to a histogram gif image by the **drun** [Lammers92] raw data imaging package. The histogram, illustrated by Figure 4.4, of the DEM provided a guide when it came time to paint the landscape with the real colors of the land.

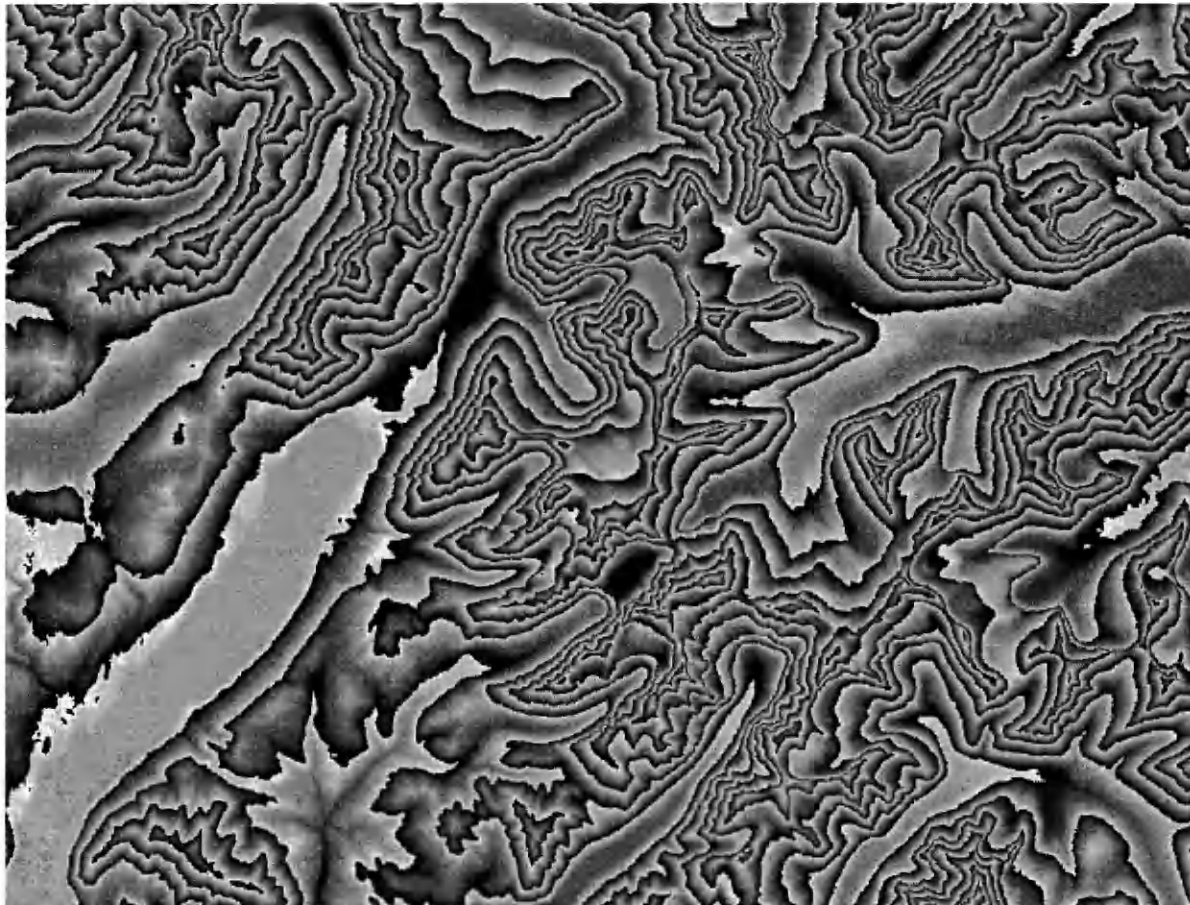


Figure 4.4 DEM Histogram Image

Pictures of Glacier Park were scanned using a **Hewlett Packard ScanJet** [HP94] flatbed scanner and the **Photoshop** [Adobe94] plug-in. The pictures included images from the shores of Lake MacDonald and multiple view points from the *Going to the Sun* road. More images were captured from a video recording of a real plane flight over the watershed using the **VideoVision Studio** hardware and **Premier**. The video included segments of glaciers, trees, rock formations and Lake MacDonald.

All of these image sources were imported as layers into **Photoshop**. The stamp tool creates a copy of a region of an image and lets the users stamp it into another image. The desired result is a single image which represents the natural coverage of the landscape including trees, grass, water, rock and snow. The regions defined by the hydrology layer were colored with samples of the water in the real pictures. The forested hill sides were colored with samples of tree stands and the rock and glacier regions were colored with samples of the rock and snow. The coverage image is illustrated by Figure 4.5.

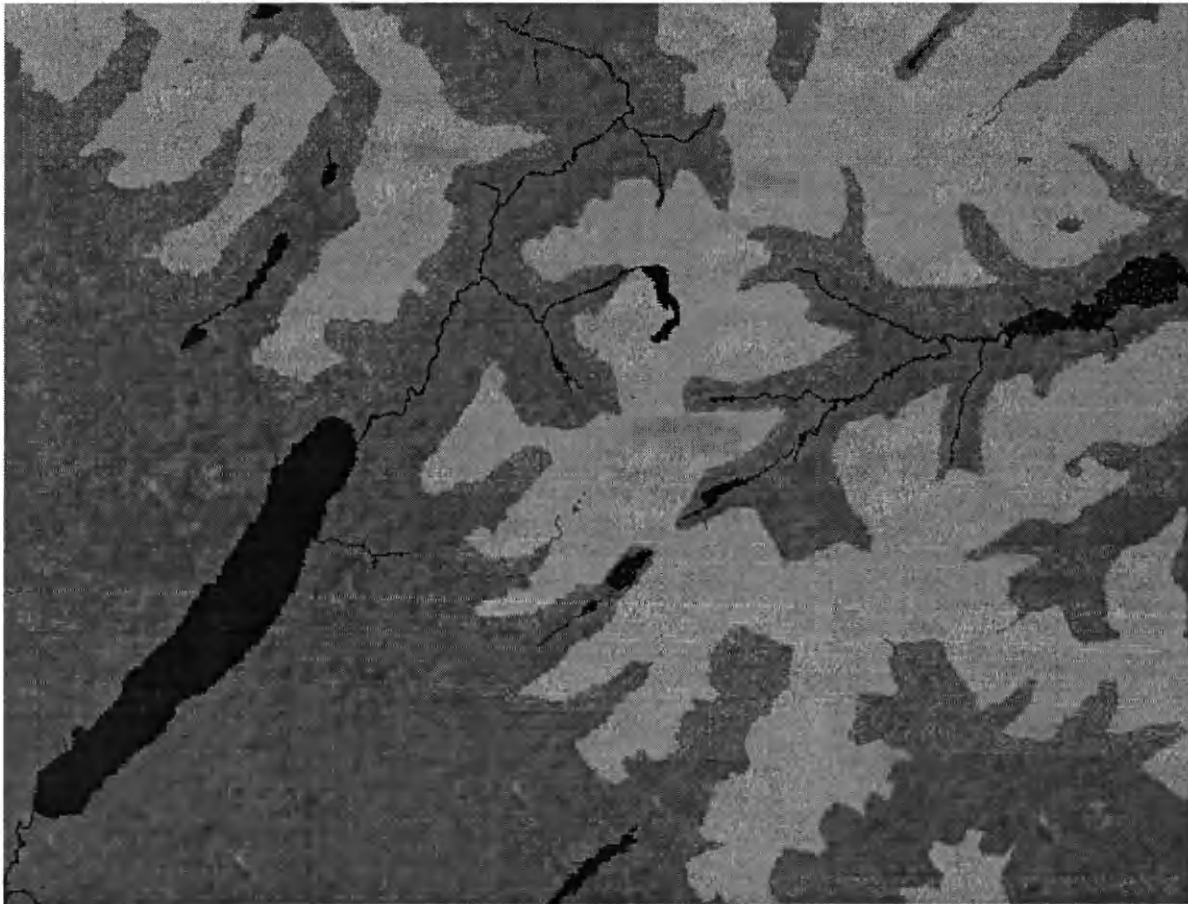


Figure 4.5 Final Cover Image

4.3 Starting with Reality

Although the reality achieved during the previous spiral pass was very effective in setting the context of the Glacier Watershed region, I realized that the experience could be improved through the use of more real images. If the user can start with a real image of Lake MacDonald and then pass into the virtual world of the data visualization we would have a very sound and complete product.

4.3.1 “Transmorgification”

The final phase to completing the visual portion of the Glacier movie required a modification of the flight path from the previous result in order to match its beginning location to that of a real image taken from the shore of Lake MacDonald, and shown in Figure 4.6.



Figure 4.6 Lake MacDonald Image

The initial section of the animation is generated using **Premier's transition** and **morph** tools. These tools create the smooth transition series of frames from the real image to the new first frame of the frames generated by the **DX** fly-through.

Chapter 5

Conclusions

5.1 Conclusions

I have successfully described the pipeline for producing data visual-animations. The tools and methods have been described, identified and presented in the form of action flow charts as illustrated in Figure 5.1. I've shown through the process of natural discovery set in the cast of the spiral model the evolution of a product which fills a previously vacant niche.

The methods and tools described in this paper have been used to create data visualization-animation following products; The Hydrogen Atom: an exploration of the inner workings of the hydrogen atom; The Rabbit Hills Fly-By: an exploration of surface and subsurface features of the Rabbit Hills oil field for the DOE Petroleum Reservoir Characterization Project; The Lake MacDonald Glacier Fly-By: a context setting flying tour of the Lake MacDonald watershed for the Columbia River Basin Project. These products range from MPEG movies to QuickTime movies on CDROM and video. All of these examples can be seen on a demo video [DASL95] and multimedia CDROM [ITRC95].

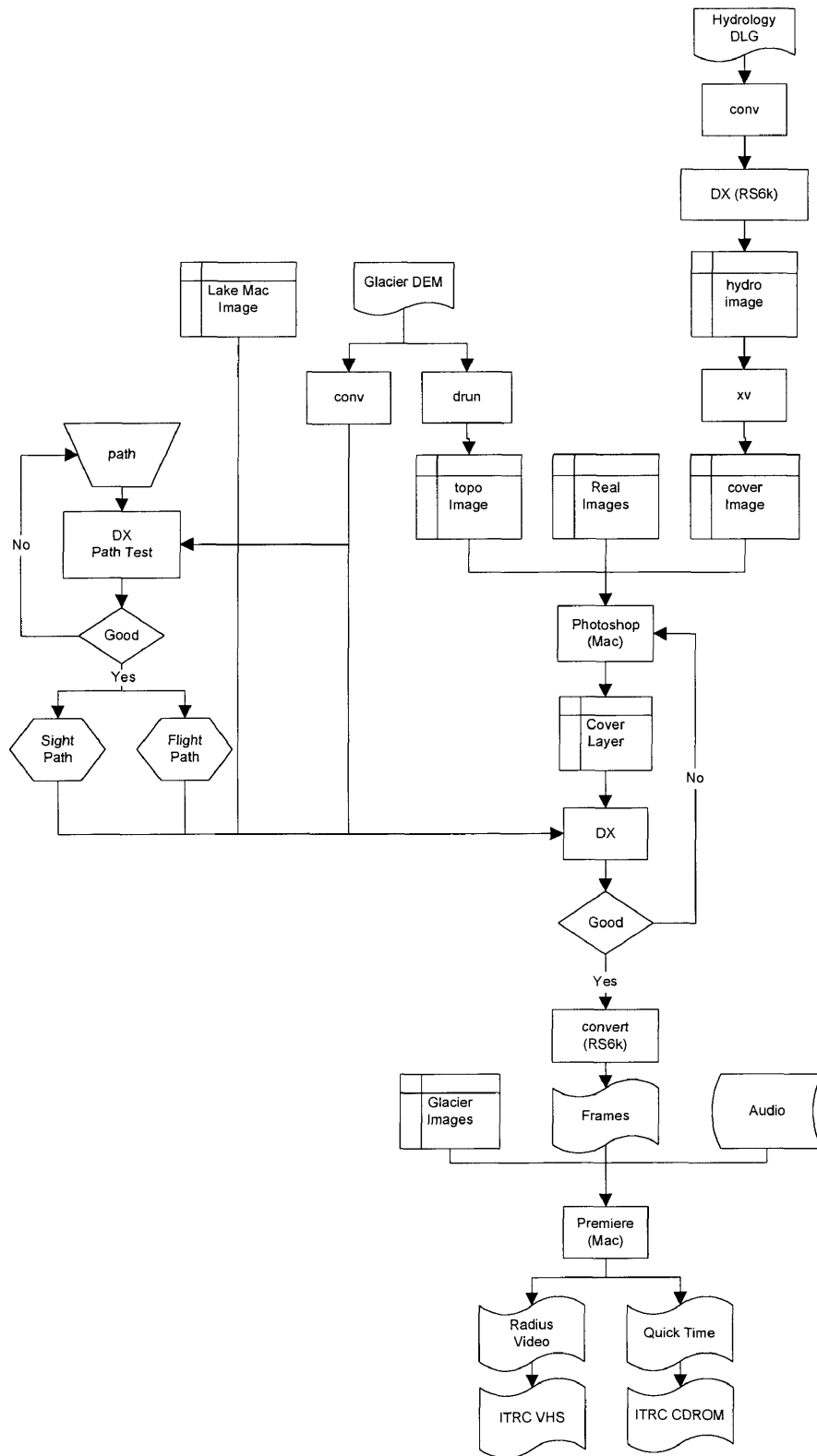


Figure 5.1 The Final Pipeline

5.2 What does the future hold

5.1.1 Real Time Control

Real time control of the flight path through the virtual landscape is a common request. There are some problems associated with extending to real time control.

- Real Time Calculation of Splines

We can certainly calculate splines dynamically given user “influenced” control points. The control point locations could be modified using keyboard controls or a joystick. The spline to fit the control points can be calculated using the current position, current trajectory, and two control points assuming the slope at the first control point is set to be parallel to the line segment connecting the current position and the final control point.

- Real Time Rendering

Rendering images given a dynamic flight path is another story, due to the extreme CPU and memory intensive aspects of rendering. There are a number of limiting factors including CPU speed, size of data set and I/O performance. Though there are special purpose real time rendering computers available (e.g. Silicon Graphic Inc.’s “Reality Engine”), these are analogous in price and use to special purpose supercomputers. For most workstation users, real time rendering is likely to be beyond reach for some time. For example, the rendering of the frames for the Glacier example took 18 hours on an RS/6000 25T workstation with 256MB of memory.

- Pre-Rendering

Even with a reduction in acceptable resolution and the landscape size, we would need a machine with extremely high computing and storage performance. Is it possible that we could pre-render a limited subset of the available data set space?

One possible solution is *QuickTimeVR*. A QuickTimeVR object, once rendered, gives the user the ability to negotiate with two degrees of freedom (rotational and tilt up/down) in a region of interest. In addition, the user can be given the ability to move from one region of interest to another with the “appearance” of choice. Each region of interest is compiled as a single QuickTimeVR object and they are linked together using “hot-spots”. A QuickTimeVR object is created using a series of still images encompassing the full range of motion at the desired object.

Another possibility is a *VRML* (Virtual Reality Modeling Language) object which is created like the QuickTimeVR object, but allows the user to access the object over the World Wide Web with any VRML capable browser, e.g. **Netscape Navigator** [Netscape95] with the **Topper** [Kinetix96] plug-in.

Bibliography

- [Adobe94] Adobe, Photoshop, Version 3.0, 1994. (Apple Macintosh, Windows).
- [Adobe94] Adobe, Premiere, Version 4.0, 1994. (Apple Macintosh, Windows).
- [Apple94] Apple Computer Inc., QuickTime, Version 2.0.3, 1994. (Apple Macintosh, Windows).
- [Boehm88] Barry W. Boehm, & TRW Systems Defense Group, A Spiral Model of Software Development and Enhancement, IEEE Computer, May 1988: 61-71.
- [Bradly94] John Bradly, xv, Version 3.10a, <http://aixpdslib.seas.ucla.edu>, 1994. (UNIX).
- [Consortium91] Consortium, PBMPLUS Toolkit, (tiffppm, tiffrow, etc.), <ftp://export.lcs.mit.edu>, 1991. (UNIX).
- [Cristy82] John Cristy, ImageMagick, (convert, etc.), E.I. du Pont de Nemours & Company, <ftp://ftp.x.org>, 1992. (UNIX).
- [Dartmouth95] Dartmouth, fetch, Version 3.01, <ftp://mirror.apple.com>, 1995. (Apple Macintosh).
- [DASL95] Distributed Application Systems Lab, DASL Demo Video, ford@cs.umt.edu, 1995. (VHS Video).
- [Gong94] Kevin Gong, mpeg_encode, keving@cs.berkeley.edu, 1994. (UNIX).

- [Grossman88] Stanley I. Grossman, & William R. Derrick, Advanced Engineering Mathematics, Harper & Row, Publishers Inc., New York, 1988: 808-809
- [HP95] Hewlett Packard, HP Scanjet 3c, 1994.
- [Holbrook94] Saxon Holbrook, FlyingCamera, <http://www.cs.umt.edu/DX>, 1994. (IBM Data Explorer, UNIX).
- [Holbrook94] Saxon Holbrook, PathInterpolation, <http://www.cs.umt.edu/DX>, 1994. (IBM Data Explorer, UNIX).
- [Holbrook93] Saxon Holbrook, quadrant converter, 1993. (IBM Data Explorer, UNIX).
- [IBM95] International Business Machines Corp. (IBM), Data Explorer (DX) Version 3.1.0, 1995. (UNIX).
- [ITRC95] ITRC, UM ITRC Demo CDROM, church@selway.umt.edu, 1995. (Windows, Macintosh CDROM).
- [Kinetix96] Kinetix, Topper, 1996. (Windows)
- [Lammers93] Richard Lammers, & Larry Band, drun, 1993. (UNIX).
- [Netscape95] Netscape, Netscape Navigator, 1995. (Apple Macintosh, Windows, UNIX).
- [Radius95] Radius, VideoVision Studio, 1995. (Apple Macintosh).
- [Thompson95] Dave Thompson, Dick Thompson, & Saxon Holbrook, mconvert, 1995. (UNIX).
- [Thompson95] Dick Thompson, & Dave Thompson, conv, Version 2, <http://www.cs.umt.edu/>, 1995. (UNIX).