Intelligent Camera Control in a Virtual Environment

Steven M. Drucker David Zeltzer
MIT Media Lab MIT RLE

Cambridge, MA. 02139, USA
smd@media.mit.edu
dz@irts.mit.edu

Abstract

This paper describes a framework for exploring intelligent camera controls in a 3D virtual environment. It presents a methodology for designing the underlying camera controls based on an analysis of what tasks are to be required in a specific environment. Once an underlying camera framework is built, a variety of interfaces can be connected to the framework. A virtual museum is used as a prototypical virtual environment for this work. This paper identifies some of the tasks that need to be performed in a virtual museum; presents a paradigm for encapsulating those tasks into camera modules; and describes in detail the underlying mechanisms that make up the camera module for navigating through the environment.

Keywords: Virtual Environments, Camera Control, Path Planning, Task Level Interfaces.

1. Introduction

Current interest in so-called *immersive* interfaces and large-scale virtual worlds serves to highlight the difficulties of orientation and navigation in synthetic environments, including abstract "data spaces" and "hypermedia" as well as more familiar modeled exterior and interior spaces. As Ware and Osborne point out, this is equivalent to manipulating a viewpoint — a synthetic camera — in and through the environment (1), and a number of recent articles have discussed the camera control problem in detail (1, 2, 3, 4).

Nearly all of this work, however, has focused on techniques for directly manipulating the camera. In our view, this is the source of much of the difficulty. Direct control of the six degrees of freedom (DOFs) of the camera (or more, if *field of view* is included) is often problematic and forces the human VE participant to attend to the interface and its "control knobs" in addition to — or instead of — the goals and constraints of the task at hand. If the intention of the human VE participant is, e.g., to observe some *object X*, then allowing him or her to simply tell the system, "Show me *object X*" is a more direct and productive interface. This is an instance of *task level* interaction. In earlier work we characterized the levels of abstraction at which one can interact with virtual objects and processes, and we described the varying "access panels" one obtains (5). Here we will describe a system for specifying behaviors for virtual cameras in terms of task level goals and constraints. As in our earlier work on camera control (6, 7), we make task level control available *as well as* enabling various direct manipulation metaphors.

We share the view of many in the user interface community that one of the first steps in interface design should be a task analysis of the application (e.g., (8, 9)). While this may be a difficult exercise in and of itself, it allows us to identify with reasonable confidence the objects and operations we should provide at the interface, and to specify the necessary software abstractions. While it is impossible to completely describe human behavior at the visual interface for all applications, our analysis, and suggests that the generic visual operations we need to support involve:

- orientation — i.e., visual comparison of egocentric and exocentric coordinate frames;
- navigation from point to point;
- exploration of unknown areas; and
- presentation to external observers.

Here we will describe one of the applications we have chosen in which to implement these ideas, one which we feel is a visually rich domain — that of an art museum. The museum contains both two- and three-dimensional objects spatially arranged in many different rooms. We chose the museum application because it is a kind of spatial information space within which we can formulate a task level description fairly easily. Based on the chosen task domain, we interviewed several

This work was supported in part by ARPA/Rome Laboratories, NHK (Japan Broadcasting Co.), the Office of Naval Research, and equipment gifts from Apple Computer, Hewlett-Packard, and Silicon Graphics.

architects, museum designers, and interactive exhibit designers to find out for what basic tasks they might want assistance. This formed the basis for the task analysis that underlies the framework for the virtual museum system.

In the next section, we discuss related work. In Sections 3 and 4 we describe the basic design of the virtual museum, including the *camera modules* we use to encapsulate various behaviors for the synthetic camera. Finally, in Section 5, we present the path-planning algorithms we have developed which enable an intelligent camera to actively assist and guide the human participant as he or she explores the virtual museum.

2. Related Work

Our current research draws mainly from two areas: computer graphic work on camera specification in 3D environments, and robotics research on pathplanning.

Miller et al (10) created a virtual museum as a multimedia database. They used video segments to create the feeling of choosing a path through a museum and selecting objects within the museum. Since video clips were used, only preexisting paths through the museum could be selected, although the use of video did permit real-time display of a high quality virtual environment on general-purpose hardware. Our emphasis, however, is on performing multiple tasks in a synthetic, three-dimensional environment.

Ware and Osborne (1) described several different *metaphors* for exploring 3D environments including "scene in hand," "eyeball in hand," and "flying vehicle control" metaphors. All of these use a 6 DOF input device to control the camera position in the virtual environment. They discovered that flying vehicle control was more useful when dealing with enclosed spaces, and the "scene in hand" metaphor was useful in looking at a single object. Any of these metaphors can be easily implemented in our system.

Mackinlay et al (4) describe techniques for scaling camera motion when moving through virtual spaces, so that, for example, users can always maintain precise control of the camera when approaching objects of interest. Again, it is possible to implement these techniques using our *camera modules.*

Brooks (11, 12) discusses several methods for using instrumented mechanical devices such as stationary bicycles and treadmills to enable human VE participants to move through virtual worlds using natural body motions and gestures. Work at Chapel Hill, has, of course, focused for some time on the architectural "walkthrough," and one can argue that such direct manipulation devices make good sense for this application. While the same may be said for the virtual museum, it is easy to think of circumstances — such as reviewing a list of paintings — in which it is not appropriate to require the human participant to physically walk or ride a bicycle. That is, at times, one may wish to interact with topological or temporal abstractions, rather than the spatial. Nevertheless, our camera modules will accept data from arbitrary input devices as appropriate.

Blinn (13) suggested several modes of camera specification based on a description of what should be placed in the frame than than just describing where the camera should be and where it should be looking.

Philips et al suggest some methods for automatic viewing control (3). They primarily use the "camera in hand" metaphor for viewing human figures in the Jack™ system, and provide automatic features for maintaining smooth visual transitions and avoiding viewing obstructions. They do not deal with the problems of navigation, exploration or presentation.

Karp and Feiner describe a system for generating automatic presentations, but they do not consider interactive control of the camera (14).

Gleicher and Witkin (26) describe a system for controlling the movement of a camera based on the screen-space projection of an object, but their system works primarily for manipulation tasks.

Our own prior work attempted to establish a procedural framework for controlling cameras (6). Problems in constructing generalizable procedures led to the current, constraint-based framework described here. Although this paper does not concentrate on methods for satisfying multiple constraints on the camera position, this is an important part of the overall camera framework we outline here. For a more complete reference, see (15).

The problem of finding a collision free path through a complicated environment has been examined a great deal in the context of robot motion planning. There have been several attempts at incorporating pathplanning systems into a graphical environment, or using the special capabilities of graphics hardware to assist in path planning (16). In general, the problem can be thought of as either vector-based approaches or bitmap approaches, somewhat akin to hidden surface removal in computer graphics. The vector based approaches involve decomposing the problem into an equivalent graph searching problem by

constructing visibility graphs or connected regions (17, 18). The bitmap approaches involve discretizing space into regular occupied or free cells and traveling through bitmap from one adjacent cell to the next (16, 19, 20).

Schröder and Zeltzer implemented an algorithm introduced by Lozano-Perez (17) in a virtual environment system called BOLIO (21, 22). A visibility graph was constructed based on the projection of all the objects in the environment onto the floor plane. The actor's position and the destination position were then connected to the visibility graph and the A* algorithm was run to search the visibility graph. The entire algorithm needed to be rerun whenever an object was moved. The visibility graph method tends to produce paths that graze objects as closely as possible as well as paths with straight lines connected to other straight lines which may be an unnatural way to move a camera. The algorithm also does not take advantage of the graphics hardware capabilities present in a graphics workstation. The path planning scheme described in this paper uses a room-to-room visibility graph to first determine the overall path for the camera, but then uses other more appropriate techniques for negotiating each room.

The local path planning technique used here is somewhat based on (20)in which the scene is rendered from an overhead view and a bitmap is constructed with each pixel representing either free space or an obstacle. This bitmap was then turned into a configuration space by growing each obstacle region by the size of the moving object for several different orientations. In our system, we assume a spherical size for the camera so the configuration space does not need to be concern with rotation. A numerical function was propagated through the bitmap from a destination location to the current location and an appropriate path through the configuration space was found by following the downward gradient of the numerical function. Their goal was not for camera motion planning or even for graphics animations, but to use a graphic workstation to assist in robot motion planning. Their paths were not necessarily suitable for camera movements due to the distance metric which they chose to propagate through the configurations space. They propagated the numerical function only until it reached a given starting point. In contrast, our algorithm is designed to be used in a building environment, precomputing as much as possible, and is specifically tailored to controlling a camera in a natural fashion.

3. System Design

The overall structure of the Virtual Museum system is based on a framework for specifying and controlling the placement and movement of virtual cameras. This framework is proposed as a formal specification for many different types of camera control (15). The central notion of this framework is that camera placement and movement is usually done for particular reasons, and that those reasons can be expressed formally as a number of constraints on the camera parameters. We identity these constraints based on analysis of the tasks required in the museum. The entire framework involves a network of camera modules which encapsulate user control, constraints, and branching conditions between modules. The work presented here does not cover the entire framework, but concentrates on the components of individual camera modules, some of the types of constraints for the camera, and different interfaces that can be built to the system. A more complete description of the entire framework is available in (15).

Our concept of a camera module is similar to the concept of a *shot* in cinematography. A shot represents the portion of time between the starting and stopping of filming a particular scene. Therefore a shot represents continuity of all the camera parameters over that period of time. The unit of a single camera module requires an additional level of continuity, that of continuity of *control* of the camera. This requirement is added because of the ability in computer graphics to identically match the camera parameters on either side of a cut, blurring the distinction of what makes up two separate shots. Imagine that the camera is initially pointing at character A and following him as he moves around the environment. The camera then pans to character B and follows her for a period of time. Finally the camera pans back to character A. In cinematic terms, this would be a single shot since there was continuity in the camera parameters over the entire period. In our terms, this would be broken down into four separate modules. The first module's task is to follow character A. The second module's task would be to pan from A to B in a specified amount of time. The third module's task would be to follow B. And finally the last modules task would be to pan back from B to A. The notion of breaking this cinematic shot into 4 modules does not specify implementation, but rather a formal description of the *goals* or *constraints* on the camera for each period of time.

Most of the modules that are present in the virtual museum are fairly straightforward and could be implemented in many different fashions. It is only the most complicated modules, – e.g. those that handle

moving along a computer generated constructed path – that show the utility of the framework since they combine complex movements with multiple other constraints.

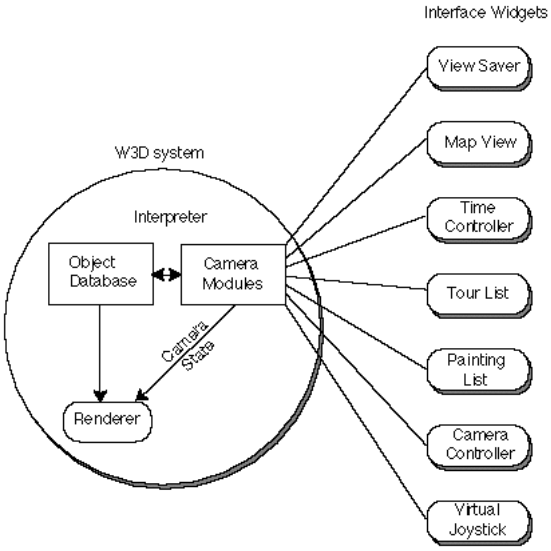

Figure 1: Overall Virtual Museum System

The overall system for the Virtual Museum is shown in figure 1. The *W3D* system is an extension to the *3D* virtual environment software testbed developed at MIT (23). The Virtual Museum system is structured this way to emphasize the division between the virtual environment database, the camera framework, and the interface that provides access to both. The system contains the following elements.

• **A general interpreter that can run prespecified scripts or manage user input**. The interpreter is an important part in developing the entire runtime system. Currently the interpreter used is TCL with the interface widgets created with TK (24). Many commands have been embedded in the system including the ability to do dynamic simulation, visibility calculations, finite element simulation, matrix computations, and various database inquiries. By using an embedded interpreter we can do rapid prototyping of a virtual environment without sacrificing too much performance since a great deal of the system can still be written in a low level language like C. The addition of TK provides convenient creation of interface widgets and interprocess communication. This is especially important because some processes might need to perform computation intensive parts of the algorithms; they can be offloaded onto separate machines.

• **A built in renderer.** This subsystem can use either the hardware of a graphics workstation (currently SGIs and HPs are supported), or software to create a high quality antialiased image.

• **An object database for a particular environment**. In this case, the database is the virtual museum which has precalculated colors based on radiosity computations which the W3D system supports. The database also contains information about the placement and descriptions of all artwork within the museum.

• **Camera modules**. These will be described in detail in the following section. Essentially, they encapsulate the behavior of the camera for different styles of interaction. They are prespecified by the user and associated with various interface widgets. Several widgets can be connected to several camera modules. The currently active camera module handles all user inputs and attempts to satisfy all the constraints contained within the module, in order to compute camera parameters which will be passed to the renderer when creating the final image. Currently, only one camera module is active at any one time, though if there were multiple viewports, each of them could be assigned a unique camera.

The diagram also shows that there are 7 different types of interface widgets that can be used to control the camera within the museum. These different widgets illustrate different styles of interaction based on the task level goals of the user.

4. Camera Modules

As shown in figure 2, the generic module will contain the following components:

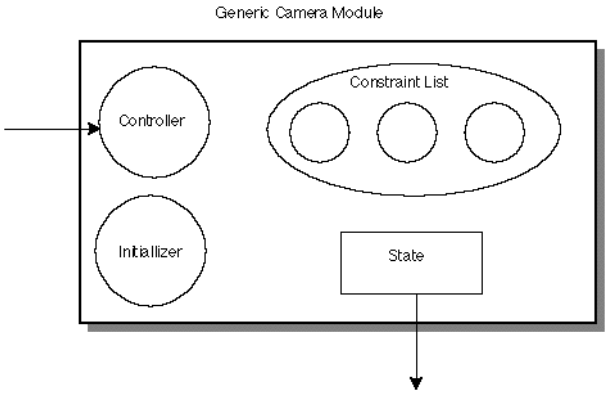

Figure 2: Generic camera module containing a controller, an initializer, a constraint list, and local state

• **the local state vector**. This must always contain the camera position, camera view normal, camera "up" vector, and field of view. State

can also contain values for the camera parameter derivatives, a value for time, or other local information specific to the operation of that module. While the module is active, the state's camera parameters are output to the renderer.

• **initializer**. This is a routine that is run upon activation of a module. Typical initial conditions are to set up the camera state based on a previous module's state.

• **controller**. This component translates user inputs either directly into the camera state or into constraints, there can be at most one controller per module.

• **constraints to be satisfied during the time period that the module is active**. Some examples of constraints are as follows:

  • maintain the camera's *up* vector to align with world *up*.

  • maintain height relative to the floor

  • maintain the camera's gaze (i.e. view normal) toward a specified object

  • maintain the camera's position on a collision-free path through world.

In this system, a constraint can be viewed simply as a black box that produces values for some DOFs of the camera. The constraint solver combines these constraints to come up with the final camera parameters for a particular module. Some constraints are desired values for a degree of freedom, for example, specifying the *up* vector for the camera or the height of the camera. Some involve calculations that might produce multiple DOFs, such as adjusting the view normal of the camera to look at a particular object. Some, like the path planning constraint, are quite complicated, and construct a path through the environment based on an initial and final position. This allows the user to see objects within the museum based on some spatial context or sequence. At any one time step, the path planning constraint still produces only 2 DOFs for the camera: the x & y position in world space.

In the virtual museum system, modules are activated by selecting the corresponding interface widget. The selected widget also passes information to the controller of the module.

Here is a description of what occurs when the user clicks on the *map view* widget. First, the corresponding map view module is activated, which means that this module's state will be used during rendering. The initializer for this module retrieves the camera state from the previous module. This allows the user to control the camera using a single set of controls, while making it possible to further adjust the position further the map view. This module's controller maps the (x,y) position of the user's mouse click directly into the current state of the camera.

The *view saver* module either retrieves the current view and saves it as part of the local state, or retrieves a saved view from the local state information and sets the camera parameters based on that.

The *joystick* module's controller maps the x & y location of the viewer's mouse click into motion through the environment. There is one constraint in the joystick module that prevents movement through a wall. Essentially, the constraint draws a line between the old position of the camera and the new position of the camera. If this line does not intersect with any walls in the environment, then that position is placed into the camera's state. If, however, the line does intersect with a wall, then the old state is kept without any changes.

The *painting list* is connected to several different modules. One module's controller (the paint module) responds to a double click of the middle mouse button and sets its state based on the painting's position in the environment. In addition, a constraint is added to the DOF controller module that constrains the camera to point toward the specified painting. When double clicking with the left button, not only is this constraint added to the DOF controller module, but source information and destination information is added to the local state of the path module. The source information is based on the current state of the camera, the destination is based on the position of the painting in the environment. The path module contains a constraint which uses this local information to construct a collision free path through the environment from the initial state to the specified final position.

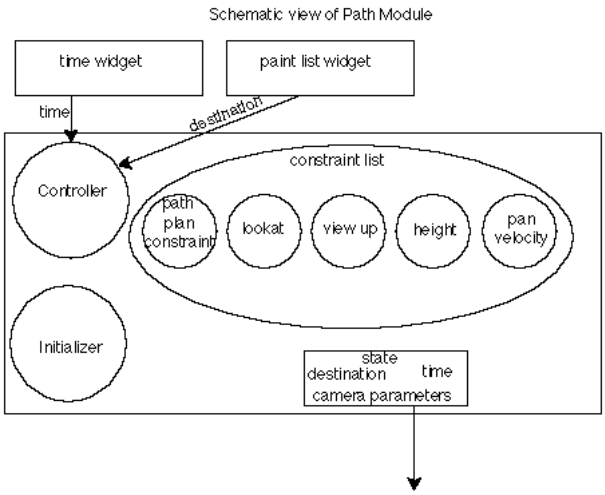

Figure 4: Schematic view of the "path" camera module

The *path* module is the most complicated module (see figure 4). It contains a constraint that calculates a path from a start position to a destination position (set when clicking on the painting list). It uses the local state's time (modified by the time widget) to indicate the current position of the camera in the (x,y) plane. The time widget can either set a particular time, or continuously update time in order to move forwards or backwards along the path. The path camera module uses several additional constraints in calculating the final camera parameters. The height is constrained to be a desired height off the ground (which is adjustable through the height constraint). The camera's *up* vector is constrained to always point in the same direction as the world *up* vector. The gaze direction is adjusted to look at the destination object when it is visible, or to look straight ahead (based on the current derivatives of the camera parameters) when the destination object is not visible. Furthermore, the gaze direction is constrained to change at a rate no greater than a specified maximum pan velocity.

If instead of a single painting, multiple paintings have been selected, the path module's path planning constraint generates a guided tour through all the selected paintings. The immediate destination is kept track of by the controller and placed in the local state's destination slot. All the other constraints act in the same manner as before.

5. Pathplanning

The most complicated constraint in the current framework is used to achieve automatic navigation through the environment. The following section describes this process in detail.

The pathplanning process is decomposed into several subalgorithms, many of which can be precomputed in order to speed calculation as much as possible. First, a general description of the overall process is given, then more detailed descriptions of each subalgorithm follow.

The problem of traveling from one point in the museum to another point is first decomposed into finding which doors to travel through. A node to node connectivity graph is pre-computed based on the accessibility between adjacent rooms in the environment. Accessibility can either be indicated by hand, or by an automatic process which uses a rendered image of the building floor, clipped at door level, and a simple visibility test between points on either side of a door. This visibility graph can be updated based on special accessibility requirements (such as handicapped access between rooms).

Traversing the graph is done by a well known graph searching technique called A* (25). The A* process, described in section 5.1, produces a list of "straight-line" node-node paths. Paths then need to be computed between each node to avoid obstacles within each room.

The process of finding the path from a doorway to any point within a room, or finding the path from any point in the room to the doorway is discussed in section 5.2. This algorithm is optimized for finding paths that originate or terminate at a doorway, so another algorithm must be used to navigate from one point to another point within a room. This second algorithm, described in section 5.3, can also deal with a partially dynamic environment as opposed to the strictly static environment discussed in the first algorithm. Finally, a method for generating a guided tour through the environment is discussed in the last part of section 5.4.

5.1 A*

The A* algorithm is based on (25). It is guaranteed to return a path of minimum cost whenever that path exists and to indicate failure when that path does not exist.

As discussed in Robot Motion Planning (20), the A* algorithm iteratively explores the node-graph by following paths that originate at a particular node. At the beginning of each iteration, there are some nodes that have already been visited, and some that are as yet unvisited. For each node that has already been visited, only the path with minimum cost to that node is memorized. Eventually the destination node is reached (if that node is reachable), and the minimum cost path can be reconstructed. Since the A* is such a common algorithm, readers are referred either to Hart et al or LaTombe for a description of the implementation.

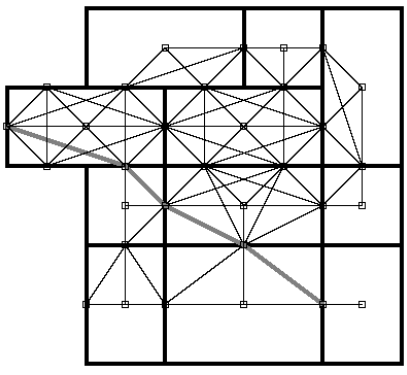

Figure 5: Room connectivity graph for the museum with a path found by A*

5.2 Room to Room Planning

Once the A* algorithm has produced a list of node to node connectivities, each of the rooms must be negotiated from one doorway to the next in turn. The method we use is somewhat similar to that of (19) except for the following: we use a different distant metric more suited to natural motion, we pre-compute and store a navigation function from each door for a room, and we fit the results using a spline for a final representation of the path.

To avoid obstacles within the room, we plan a path based on a two dimensional projection of the obstacles in the room onto the plane of the floor. Much of the following work can be done in a preprocessing stage so that the actual computation of a path through a room is extremely rapid. The 2D projection of the room can be rendered using the hardware rendering of a graphic workstation at whatever resolution is appropriate for the path planning. Subsequently a *global numerical navigation function* (19) is calculated in a wavefront expansion from each of the doorways. A separate navigation map is stored for each doorway into a room. To demonstrate, a Manhattan (L1) distance metric is used for the following example, the manhattan distance metric implies that only the 4 cells surrounding a cell to the N, S, E and W are considered neighbors.

7	6	7	8	9	10	11
6	5	6	7	8	9	10
5	4	5	6		10	11
4	3	4	5		11	12
3	2	3	4		12	13
2	1	2	3		13	14
1	0	1	2		14	15

Figure 6: Navigation function calculated with a manhattan metric (L1) starting from the 0. One path planned from the upper right by following the gradient downwards.

Once the navigation function is computed for a room, it is possible to travel to anywhere within the room by simply following the gradient of the distance function from the goal point back to the origin of the navigation function. There are a few problems that exist in using the algorithm as is:

• the paths generated by the algorithm tend to graze objects as closely as possible which is unacceptable for camera motion. To fix this, the objects are increased in size by a fixed amount in all directions. This prevents the navigation function from passing too close to any object in the environment.

• the paths generated from using a Manhattan metric produce paths that are unacceptably aligned to the major axes. By using a different distance metric, we were able to generate paths that made movement along more axes possible. See figure 7.

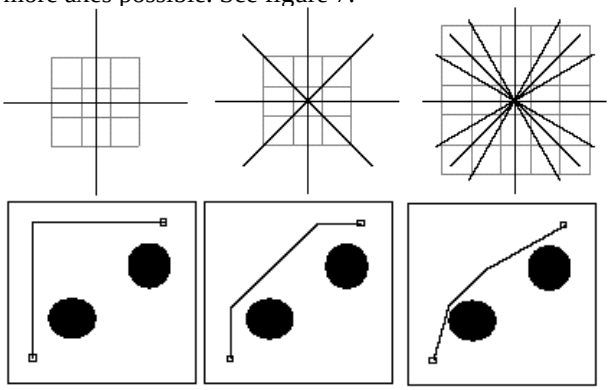

Figure 7: 3 distance metrics and resulting paths. L1 allows 4 distinct directions of movement. L2 allows 8, L3 allows 16. We used L4 which allows 32 directions.

• the paths produced are discretized into the space of the rendered image and must be converted into continuous paths in the space of the room. We can transform the points into the proper space using an affine transformation, and then fit a spline curve to the points using a least squares curve fitting method to find the best path. The control points are chosen to be as evenly spaced as possible while minimizing the difference between the spline and the sample points. We can also apply additional tangency constraints at the starting and ending points to make sure that the path goes through doorways in a perpendicular fashion.

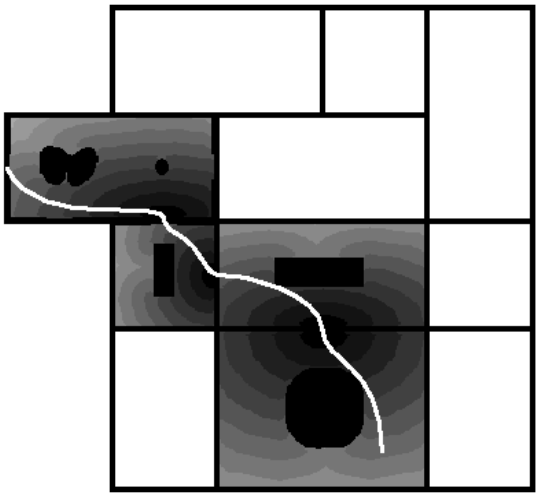

Figure 8: An eventual path from left middle, to middle bottom through 5 rooms of the museum. The navigation function that was used for each room is pictured. Usually the navigation is chosen for a room is the one generated that leads to the exit door of that room. In the final room, the algorithm is run backwards and calculates a path from the destination location to the entry room. The resultant path is reversed and combined with the other paths.

5.3 Travel within a Room

Path planning using the global navigation function as described in section 5.2 is extremely convenient because all the computation intensive work can be performed in a preprocessing stage (all the renderings of the scene, and the propagation of the distance function along the image, for each doorway). The final path planning process is extremely rapid on a typical workstation (less than .5 secs on an R3000 SGI). There are however 2 drawbacks to this algorithm:

• it does not deal with planning a path from one point within a room to another point within the room. Because the navigation function is calculated from the doorways of a room, we can conveniently find a path from any point in the room to a doorway, or from a doorway to any point in the room. But we can not easily find a path between two points in a room except via a doorway.

• it does not deal with dynamically changing environments. The entire, computation intensive parts of the algorithm must be rerun whenever an obstacle in the environment moves.

To address the problems described above, an alternate path planning algorithm loosely based on (19) can be used. More computation needs to be done for this algorithm so it is only used when necessary.

Again, as much preprocessing as possible is performed to make the algorithm as interactive as possible. As before, the static part of a scene is projected onto a 2D plane by graphics hardware. Wavefront expansion is used to propagate a penalty distance function outwards from each object. The wavefront expansion is stopped as soon as the wave meets another wave coming from some other object (or from another place on the same object). We can use a manhattan (L1) distance metric, but we keep track of the origin pixel of each wavefront. For each pixel, we can then calculate the Euclidean distance to the nearest object. This preprocess generates a distance map which can be turned into a repulsive gradient field by using a function like a/d^n where d is the distance, a and n are constants that can be chosen for the task. An absolute cutoff value beyond which repulsion effects are ignored can be used if desired.

At runtime, an attractive potential is created to the goal destination (using a function of the form $c*d^n$ where d is the distance and c and n are chosen for the task) and this is summed with the repulsive potential. Any moving obstacles can also generate a repulsive potential. The sum of the potentials produces the overall gradient.

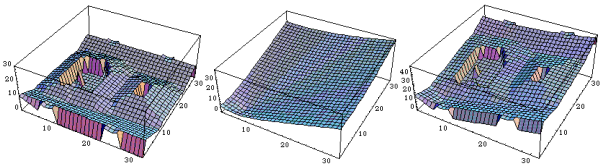

Figure 9: A precalculated repulsive potential is added to a run-time attractive potential to produce the final navigation function on the right. This is used to guide a breadth first search through potential space to find the resultant path.

Simple gradient following tends to get trapped into local minima. Instead of always following the gradient, the gradient information in the discrete grid is used as a breadth first guide in a search of the entire grid space. When the algorithm heads into a local minima, the algorithm essentially backs out of the minima on its way to finding the appropriate path to the goal. Again, the algorithm produces discretized paths which must then be fit by splines to produce a continuous path.

5.4 Tour Planning

Finally, we developed an algorithm to generate the shortest path that will view all the artwork that is specified by the user. In a similar fashion to the point to point navigation, the tour planner divides the problem up in several stages. First, the algorithm locates all the rooms that are slated to be visited. Then, an exhaustive search is made of all the paths that connect each room. This is an exponential time algorithm, but since there is a relatively low branching factor for each room (as well as a fairly small number of rooms), the algorithm is still rapid enough to be used interactively. After the rooms have been ordered, the paintings within each room need to be ordered based on the entry and exit doors (visit the painting first which is closest to the door from which the room is entered, and visit the painting last next to the exit door). At this point we have a list of paintings that will be visited in the order specified. The algorithms discussed in the previous three sections can be used to plan paths between each of the paintings and the results can be combined into the global tour.

6. Summary

We have presented an overall framework for exploring camera controls in a 3D virtual environment. Special constraints based on an analysis of task requirements can be designed and combined with a host of other constraints for camera placement. Interfaces can be connected to the system to explore human factors issues while maintaining a consistent underlying structure. We feel that it is important to separate the

underlying framework which can incorporate task level requirements from the user interface.

Future work can be in several different directions. More efficient path planning algorithms can be substituted into the camera module framework as they are implemented. In particular, algorithms to deal with totally dynamic environments would be useful. One common task in many virtual environments is the presentation of the information to a third party observer. While the path planning constraint goes toward convenient automatic presentation, a number of other considerations must be made, including the difficult problem of editing a single move into several, smaller cuts. We are incorporating a variety of constraints from cinematography into the camera framework and current work is progressing on techniques that combine those constraints in a meaningful fashion.

References

1. Ware, C. and S. Osborn. *Exploration and Virtual Camera Control in Virtual Three Dimensional Environments* in **Proc**. **Proc. 1990 Symposium on Interactive 3D Graphics**, Snowbird, Utah, March 25-28, 1990, pp. 175-184.

2. Chapman, D. and C. Ware. *Manipulating the Future: Predictor Based Feedback for Velocity Control in Virtual Environment Navigation* in **Proc. 1992 Symposium on Interactive 3D Graphics**, Cambridge, MA: ACM Press, March 30-April 1, 1992, pp. 63-66.

3. Phillips, C.B., N.I. Badler, and J. Granieri. *Automatic Viewing Control for 3D Direct Manipulation* in **Proc**. **1992 Symposium on Interactive 3D Graphics**, Cambridge MA: ACM Press, March 29 - April 1, 1992, pp. 71-74.

4 Mackinlay, J.S., S. Card, and G. Robertson, *Rapid Controlled Movement Througha Virtual 3d Workspace*. **Computer Graphics**, **24**(4), pp. 171-176.

5. Zeltzer, D., *Task Level Graphical Simulation: Abstraction, Representation and Control*, in **Making Them Move: Mechanics, Control and Animation of Articulated Figures,** N. Badler, B. Barsky, andD. Zeltzer, eds., 1991, Morgan Kaufmann: San Mateo CA, pp. 3-33.

6. Drucker, S., T. Galyean, and D. Zeltzer. *CINEMA: A System for Procedural Camera Movements* in **Proc**. **1992 Symposium on Interactive 3D Graphics**, Cambridge MA: ACM Press, March 29-April 1, 1992, pp. 67-70.

7. Zeltzer, D. and S. Drucker. *A Virtual Environment System for Mission Planning* in **Proc**. **1992 IMAGE VI Conference**, Phoenix AZ, June 19-23, 1992, pp. 125-134.

8. Brooks, R., *Comparative Task Analysis: An Alternative Direction for Human-Computer Interaction Science*, in **Designing Interaction: Psychology at the Human-Computer Interface,** J.M. Carroll, ed., 1991, Cambridge University Press: Cambridge, England, pp. 50-59.

9. Bass, L. and J. Coutax, **Developing Software for the User Interface**. 1992, Reading MA: Addison-Wesley.

10. Miller, G., *et al., The Virtual Museum: Interactive 3D Navigation of a Multimedia Database*. **The Journal of Visualization and Computer Animation**, **3**(3), pp. 183-197.

11. Brooks, F.P., Jr. *Grasping Reality Through Illusion -- Interactive Graphics Serving Science* in **Proc**. **Proc. CHI '88**, May 15-19, 1988, pp. 1-11.

12. Brooks, F.P., Jr. *Walkthrough -- A Dynamic Graphics System for Simulating Virtual Buildings* in **Proc**. **Proc. 1986 ACM Workshop on Interactive 3D Graphics**, Chapel Hill NC, October 23-24, 1986, pp. 9-21.

13 Blinn, J. Where am I? What am I looking at? **IEEE Computer Graphics and Applications** :76-81; (July 1988).

14. Karp, P. and S. Feiner. *Issues in the Automated Generation of Animated Presentations* in **Proc**. **Graphics Interface '90**, Halifax, Nova Scotia, pp. 39-48.

15 Drucker, S.M. **Intelligent Camera Control in Graphical Environments**, PhD Thesis, 1994, Massachusetts Institute of Technology, Cambridge, MA.

16. Lengyel, J., M. Reichert, B.R. Donald, and D.P. Greenberg. *Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware* in **Proc**. **ACM SIGGRAPH 90**, Dallas TX, August 6-10, 1990, pp. 327-335.

17. Lozano-Perez, T. and M.A. Wesley, *An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles.* **Communications of the ACM**, **22**(10), October 1970, pp. 560-570.

18. Brooks, R.A. *Solving the Find-Path Problem by Good Representation of Free Space* in **IEEE Trans. Systems, Man and Cybernetics**, SMC-13(3), March/April 1983, pp. 190-197.

19. Barraquand, J., B. Langlois, and J.C. Latombe. **Numerical Potential Field Techniques for Robot Path Planning,** STAN-CS-89-1285, 1989, Stanford University.

20. Latombe, J.-C., **Robot Motion Planning**. 1991, Kluwer Academic Publishers.

21. Schröder, P. and D. Zeltzer. *Path Planning in BOLIO* in **Course Notes, Synthetic Actors: The Impact of Artificial Intelligence and Robotics on Animation**, Atlanta GA, August 2, 1988.

22. Zeltzer, D., S. Pieper, and D. Sturman. *An Integrated Graphical Simulation Platform* in **Proc**. **Graphics Interface '89**, London, Ontario, Canada, June 19-23, 1989, pp. 266-274.

23. Chen, D.T. and D. Zeltzer. **The 3d Virtual Environment and Dynamic Simulation System,** Computer Graphics And Animation Group, August 1992, MIT Media Lab, Cambridge MA.

24. Ousterhout, J.K. *Tcl: An Embeddable Command Language* in **Proc**. **Proc. 1990 Winter USENIX Conference**.

25. Hart, P.E., N.J. Nilsson, and B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. **IEEE Transactions on Systems Man and Cybernetics**, **4**(2), 1968, pp. 100-107.

26. Gleicher, M. and A. Witkin *Through-the-Lens Camera Control*..**Computer Graphics 26**(2), 1992, pp. 331-340.