Beyond Viewpoint:
X3D Camera Nodes for Digital Cinematography

Jeffrey D. Weekley
MOVES Institute
Naval Postgraduate School
Monterey, CA 93943
jdweekley@nps.edu

Don Brutzman, Ph.D.
MOVES Institute
Naval Postgraduate School
Monterey, CA 93943
brutzman@nps.edu

ABSTRACT
This paper describes four candidate X3D extension nodes: Camera, Shot, Movement and OfflineRendering. An X3D lexicon for camera movements is defined so that individuals directing virtual cameras in X3D can directly apply terms that film directors and cinematographers understand. This approach greatly simplifies the technical tasks involved in creating precise camera animations and setting up still images for digital photography. Further, candidate methods are examined for implementing Depth of Field for focus control. Moving beyond the typical clumsiness of Viewpoint control can enable authors to create compelling still and moving images from X3D scenes.

Categories and Subject Descriptors
D.3.3 [**Computer Graphics**]: Three-Dimensional Graphics and Realism – Camera viewpoint, rendering, depth-of-field.

General Terms
I.3.7 [Three-Dimensional Graphics and Realism]: Virtual reality—Color, shading, shadowing, and texture I.3.6 [Methodology and Techniques]: Standards—Languages Design, Human Factors, Standardization.

Keywords
X3D graphics, camera movement, viewpoint, depth of field, field of view, digital photography, offline rendering, machinima.

1. INTRODUCTION
The ability to move the camera separates cinematography from still photography. The ability to have both the subject and the camera move independently distinguishes it even further. This both complicates and enables creativity for visual storytelling. The same relationships can be expressed in 3D graphics, which is further unconstrained because so many different aspects of camera, scene entities and behaviors can be animated simultaneously. Non-real-time 3D graphics have long leveraged the power of traditional cinematography plus this unconstrained freedom to great effect.

Currently, such complexity and license is only possible in X3D with user navigation via NavigationInfo or overly complicated Viewpoint animation. A lack of a nuanced camera, stilted or arbitrary camera movements, and a lack of camera-specific vocabulary generally hinder X3D as a source for high quality rendered video. First-class camera capabilities are needed in X3D. We propose a Camera node to include: camera movement, movement sequencing, field of view (FOV), f-stop (and thus aperture) control, focal length, focus distance, and camera aim. Such capabilities can enable authors to duplicate, create, demonstrate and record sophisticated camera work in an X3D scene.

Web3D 2009 Symposium, 16-17 June 2009, Darmstadt Germany.

2. OVERVIEW
Many of the elements required for a Camera node already exist in other nodes within X3D. What is missing is a vocabulary to describe basic camera movements. These are expressed generally in Figure 1 and extended to X3D in Figure 2. Many of these camera movements can be authored already in X3D, but doing so is complicated and their expression in X3D doesn't correspond to concepts in the cinematic domain. While it is not our intention to teach cinematography, we do wish to provide an equal footing for X3D as a medium for high-quality camera work as applied to virtual still photography, real-time directed long-form content, and rendered-to-video 3D graphics.

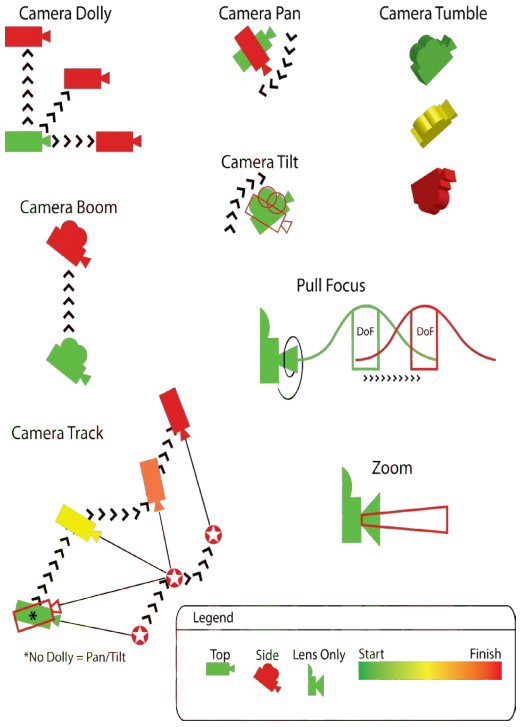

Figure 1. Basic Camera Movements

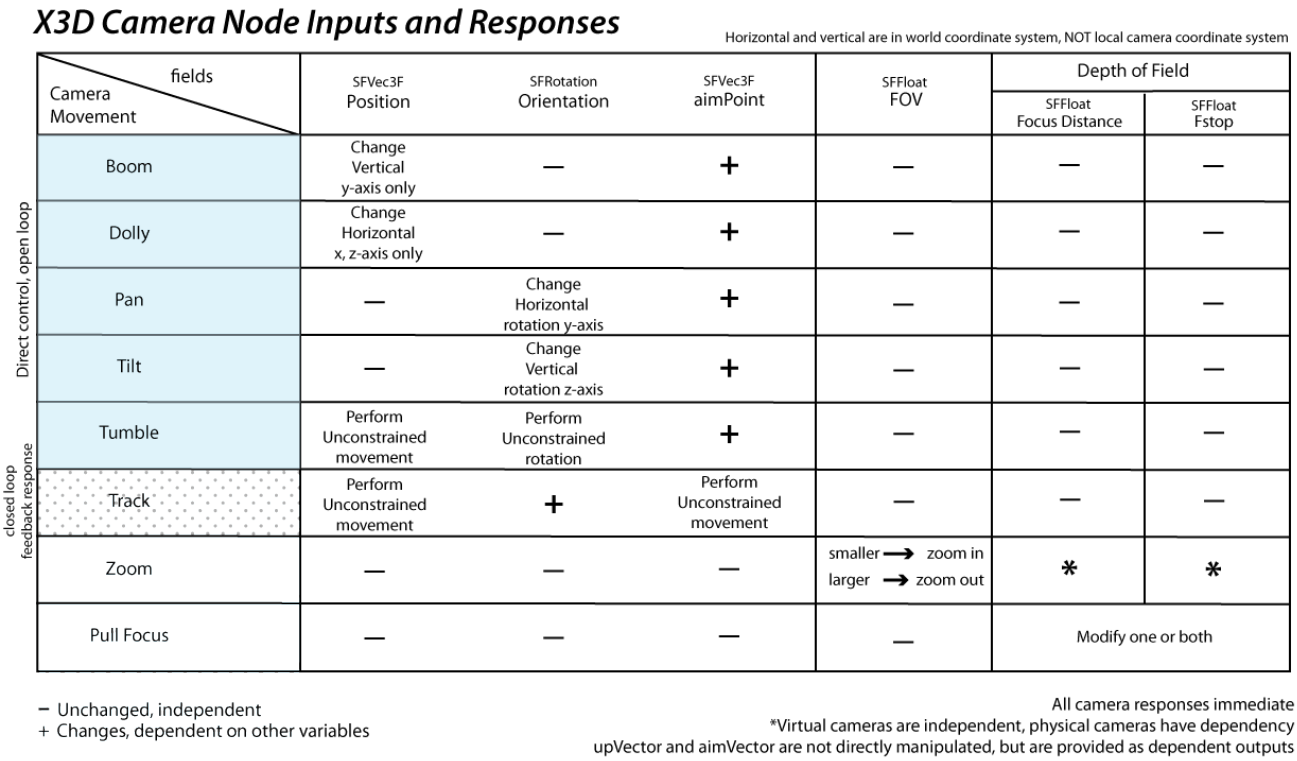

Figure 2. X3D Camera Node Dependencies Corresponding to Basic Camera Movements

2.1 Design Goals

Since X3D is extensible, the overarching design goal is to provide the author with simple Prototypes that express basic camera movements common in cinematography. Such prototypes can be repeatedly used without difficulty, and eventually considered as native X3D nodes.

The physical properties of a camera and lens are often dictated by its geometry, i.e. aspect ratio, focal length, size, weight, f-stop, et cetera. Similarly the physics of optics and the laws of motion have solidly constrained the tradition of visual storytelling: practical cameras and lenses can only move according to physical constraints. This is not to suggest that they are not interesting or innovative. On the contrary, one hundred years of exploration and practical application of cinematography ought to be respected and emulated. It might seem that these physical constraints are no longer relevant in virtual cameras. Even so, virtual cameras still need to behave in ways that we understand.

Constraining and defining virtual cameras to emulate practical cameras is therefore another design goal. We wish to provide common vocabulary and understanding across these domains. Therefore a design goal for new camera nodes is to make X3D camera animation more repeatable and predictable enough to be used as a source for virtual imagery all defined in terms that filmmakers and photographers grasp.

2.2 Use Cases

We identify six basic use cases: invoke a single camera movement; invoke a series of camera movements, both simultaneous and sequential; the ability to match a single practical (real) camera movement for a single shot (e.g. camera instance); the ability to match a series of practical camera movements for a single photograph, single shot, or series of shots; and the ability to loop a series of camera movements.

We also establish two views of time: a Continuous Time approach that corresponds to direct animation of the camera i.e. Interpolator/Chaser/Damper stream of events arriving via ROUTEs; or Duration Time approach that corresponds to the execution of one or more camera movement behaviors, each for a discrete time period in seconds. The Continuous Time approach can be practically beneficial for real-time X3D, while the Duration Time approach lends itself to creating shots for non-linear video editing of a timeline sequence as in film.

Other potentially valuable use cases are machinima and previsualization of scenes in virtual sets. Machinima involves the creation of video from 3D games or applications. Previsualization is widely used in effects-driven feature films and feature-length animated films where 3D graphics are used as storyboards, informing planning and production with respect to camera position, blocking, set design and visual effects.

3. CURRENT X3D FUNCTIONALITY

Figure 2 summarizes current X3D node capabilities of interest for viewpoint animation and shows field interdependencies.

It is helpful to remember that X3D scenes are fixed in place. User navigation around these scenes is either driven by user input or driven by placing the viewpoint under a Transform node that is then animated. In many ways Viewpoint nodes are like cameras already, often prepositioned by the scene author in locations and directions of interest. In combination with the

NavigationInfo node, they dictate how users interact with the X3D scene. It remains quite difficult for a user to navigate precisely and consistently enough to generate a 'take' for video suitable for the motivating use cases, however. The Viewpoint and NavigationInfo nodes do not provide enough information to perform off-line rendering of video, or even single-shot rendering of still images for digital photography.

Several well-known problems persist with Viewpoint and NavigationInfo. Each set of nodes can be bound independently, making coordinated control difficult or impossible. Since users can select them from parent or subsidiary Inline scenes, it is not possible for an author to strictly control view-based navigation without turning off all related nodes and disabling navigation entirely. Furthermore, there is no way to interrogate the Scene Access Interface (SAI) to determine which of many candidate nodes are currently bound, to interrogate current viewpoint location and direction, etc.

More than just enabling a 'magic carpet ride' viewpoint, a combined, coherent approach to viewing and navigation is needed in order to meet the needs of many common author intentions and to allow for proper cinematic camera control.

3.1 Traditional X3D View Animation

In order to animate (i.e. reposition and orient) a Viewpoint in X3D, it is necessary to ROUTE values to either or both of the position and orientation fields by using interpolator-ROUTE mechanisms. One must always be careful that these are animated in tandem, since moving the Viewpoint in XYZ space can easily point it astray. This painstaking method works, but it requires attention to detail and a sound conceptual grasp of the 3D motion you intend to portray. There are also multiple linkages that must be maintained throughout such a scene. The same can be said of animation by the Transform node above a Viewpoint node. Practically, this method often results in overly simple camera movements, static camera positions or, even worse, default Viewpoints only. Expert users can often generate compelling Viewpoint movements using free navigation, but these movements are not easily repeated. They rely on expert user skills peculiar to the browser being used, not on author intent. Example X3D implementations that generalize such approaches are the *ViewpointSequencer* and *Animated ViewpointRecorder* prototypes in the Tools/Authoring section of the Savage X3D model archive.

4. CAMERA NODE DESIGN

Most of the functionality needed for an X3D camera node is already present in Viewpoint, NavigationInfo, TimeSensor, and the interpolator nodes (Position, Orientation and Scalar). Camera prototype construction is mostly a matter of correctly connecting them. This technical task is complicated somewhat by the artistic requirements of camera work, since most cinematic camera movements use the patterns of Figure 1.

Viewpoint contains two essential fields: position and orientation. NavigationInfo provides visibilityLimit (farClippingPlane) and the first field from Avatar Size (nearClippingPlane). Still, purposefully positioning and orienting a camera in X3D is hard. We refactor information already present to simplify camera placement so that cinematic effects can be achieved. Figure 2 maps common camera movements to their respective X3D attributes. This correspondence helps to match the lexicons of cinematography and directing camera movements. Given an understanding of this vocabulary, these new camera nodes are designed so that such movements can be accomplished directly.

4.1 Camera Aim

Currently in X3D there is no direct concept of camera aim, whereby one translates a viewpoint and it automatically calculates the orientation required to keep looking at the object of intent. Some browsers include this functionality as a user aid to navigation, but it is not well specified and not universally or consistently implemented. By exposing camera aimPoint, the correct orientation is computed so that the camera continues to point in the direction intended (aimVector). Pan and Tilt without camera aim is also allowed. Pan and Tilt with camera aim is actually a camera Track, even if one or both are moving. Non-tracking and tracking camera movements in sequence are allowed in cinematography, especially in documentary style (though not always in an artistically effective fashion). The aimVector is outputOnly and is provided as a calculated vector so that supplementary animation scripting can avoid quaternion mathematics. It is related to aimPoint, but is not directly manipulated. The X3D Follower component also has appeal for animating aimPoint. This camera-animation design enables authors to set a goal aimPoint in sequential camera Movements. Such an approach is much easier than simultaneously trying to translate a Viewpoint while orienting it in a specific direction.

4.2 Camera upVector

In addition to providing camera aim, we include an UpVector that allows the camera to be constrained so that camera movements do not generate new camera rotations that are unnecessary or undesirable. Certain camera movements such as Track, Tilt and Tumble generate UpVector changes that are used to constrain the roll, pitch and yaw of the camera. Generally, it is desirable to keep the Camera upVector in the Cartesian quadrants I and II, even when doing a Tumble, as this keeps the 'world' right-side-up.

4.3 Non-Motion Lens Adjustments

Two important motionless camera movements correspond to Field-of-View (FOV) and Depth-of-Field (DOF) lens adjustments. Both fields can be modified simultaneously with other movements of the camera itself. Moving the focal plane towards the camera is known as "Pulling Focus", while moving it away from the camera is known as "Pushing Focus." Cinematographers often use this device (perhaps subtly) to direct viewer attention to a desired element in the scene.

4.3.1 Field of View (FOV)

FOV corresponds directly with the physical property of a lens focal length. In X3D, a smaller FOV effectively *zooms* the lens; a larger FOV has the practical effect of a wide-angle lens. Moving the viewpoint closer, while increasing the FOV creates a fish-eye distortion. See Figure 3.

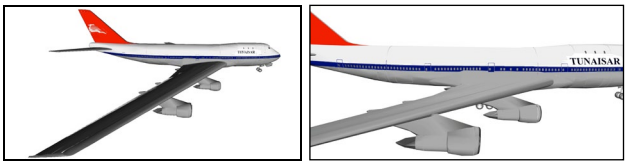

Figure 3. Savage model of 747 aircraft seen from 25M with FOV 1.57 (90°) and then less-distorted FOV 0.7853 (45°)

Careful FOV authoring usually prevents unwanted distortion, but it is conceivable that wide-angle lens distortions, which are commonly used in cinema, might purposefully be authored to convey a sense of fantasy or distorted reality. The classic camera movement of dolly-zoom (dolly the camera away at the same rate the lens is zoomed in) is an example of this distorted reality used for dramatic effect. Emulation of other practical lens distortions remains interesting and appears to be feasible using FOV and DOF animation effects.

4.3.2 Depth of Field (DOF)

Depth-of-Field (DOF) is defined to be the area enveloping the focal place in an optical lens system within which objects retain a definitive focal quality [Scofield 1992]. As in many approaches to 3D graphics, it has no immediate corresponding X3D attribute, since the virtual 3D graphics camera is idealized so that there is only a single path for light to travel from the scene to the viewer [Demers 2004]. The blur associated with a real lens is therefore approximated and called the "Circle of Confusion" [Potmesil 1982]. There are multiple generally recognized methods for this approximation: distributed traced rays across the surface of a (nonpinhole) lens [Cook et al. 1984]; accumulated buffer technique [Haeberli and Akeley 1990]; rendering multiple layers [Scofield 1994]; forward-mapped z-buffer techniques [Potmesil and Chakravarty 1981]; and reverse-mapped z-buffer techniques [Arce and Wloka 2002, Demers 2003]. A new method utilizes graphics card programmability: anisotropic filtering using partial differential equations, which may be particularly attractive because it is optimized for real-time rendering and does not suffer from many of the artifacts of previous methods [Bartalmio et al. 2004].

5. X3D CAMERA NODE SIGNATURES

The Camera node contains Shot and Movement nodes, each carefully combined with camera-specific information required to deliberately render a scene. A camera Movement is atomic and can be used to assemble a Shot of one or more Movements. A Shot has a start and an end, and so multiple Shots can be sequenced to animate the full action in a scene. It logically follows that Shot duration is the sum of the durations of each of its constituent Movements, and similarly a Camera's duration is the sum of the contained Shot durations. The Camera node and its dependent nodes supply all the necessary information and map quite nicely to existing X3D interpolators and associated nodes. It should be noted that the X3D Camera node is used for author-directed rather than user-interactive viewing of the scene, and may also serve as a recording camera for off-line photographic or video rendering of a scene. When the Camera node is bound, all other types of Navigation (EXAMINE, WALK, FLY, etc.) and other Viewpoints are disabled.

5.1 Movement Node

There are two possible approaches to implementing a Shot: each camera Movement gets its own interpolators, TimeSensor clock and ROUTEs; or else all child Movement interpolations are together aggregated into a single Shot animation, constructing a single set of interpolators, Time Sensor and ROUTEs to run them. The second approach is most appealing since it simplifies the computation of values, can be repeated for each Shot, reduces the complexity of animating ROUTEs, and allows for the Duration Time approach.

For complex camera animation, multiple camera Shots and Movements can be authored individually, making series of Shots easier to author. Even complex Shots can be split into individual Movements with initialization parameters and goal parameters. Movements can be strung together so the camera follows a path, and shots can be constructed so that they cut from one to another. In video editing, a 'jump cut' is a sequential shot of the same subject where the camera moves only slightly between cuts. The general rule is that the camera move at least 30 degrees between shots. This is easily accomplished with the X3D Camera node by making sure the initialPosition in the subsequent Shot is sufficiently far away from the prior action.

Each node is presented in order from simplest to most complete. The proposed Movement node signature is shown in Figure 4.

Movement : X3DChildNode

description	SFString	inputOutput	Descriptive summary
enabled	SFBool	inputOutput value=true	Whether node is active
duration	SFFloat	inputOutput value=0	Duration in seconds for this move
goalPosition	SFVec3f	inputOutput value=0 0 10	Goal camera position for this move
goalOrientation	SFRotation	inputOutput value=0 0 1 0	Goal camera rotation for this move
goalAimPoint	SFVec3f	inputOutput value=0 0 0	Goal aimPoint for this move ignored if tracking=false
goalFieldOfView	SFFloat	inputOutput value=0.7854	Goal fieldOfView for this move
goalFStop	SFFloat	inputOutput value=5.6	Goal focal length divided by effective aperture diameter, indicating focal plane width
goalFocusDistance	SFFloat	inputOutput value=10	Distance to focal plane of sharpest focus
isActive	SFBool	outputOnly	start/stop yields true/false, useful to trigger external animations

Figure 4. Proposed X3D Movement Node Signature

5.2 Shot Node

The Shot node has two modes: tracking and non-tracking. See Figure 2 for a comparison of the node inputs and responses. When 'tracking' is true, the author can change the camera position and aimPoint, but the camera's orientation is calculated automatically by computing orientation values that align to direction vector between the camera position and aimPoint. The initialOrientation is set, but subsequent camera orientations are calculated to keep the camera pointing at the aimPoint.

In non-tracking mode, the author can either specify goalOrientation (as in a tilt or pan) or else ignore orientation altogether by moving the camera (as in a dolly). In every case, since all the characteristics at each time step are known, the appropriate arrays are built as the keyValue in a traditional interpolator and the key is calculated in proportional time steps over the range (0..1). See Figure 5.

$$\text{key} = \frac{[\ 0,\ \text{duration}[0], \text{duration}[1], \text{etc.}\ ]}{\text{cycle.Interval} = \Sigma\ \text{durations}}$$

Figure 5. Calculation of Key Array in X3D Camera Node

The proposed node signature for Shot is shown in Figure 6.

Shot : X3DChildNode

description	SFString	inputOutput	value=""	Descriptive summary
enabled	SFBool	inputOutput	value=true	Whether node is activated
tracking	SFBool	inputOutput	value=true	Whether or not camera is tracking aimPoint, fixed for the conduct of this shot
moves	MFNode	inputOutput	value=NULL	Set of Movement nodes
initialPosition	SFVec3f	inputOutput	value=0 0 1	Setup to reinitialize camera position for this shot
initialOrientation	SFRotation	inputOutput	value=0 0 1 0	Setup to reinitialize camera orientation for this shot
initialAimPoint	SFVec3f	inputOutput	value=0 0 0	Setup to reinitialize aimpoint (relative location for camera direction) for this shot
initialFieldOfView	SFFloat	inputOutput	value=0.7854	pi/4
initialFStop	SFFloat	inputOutput	value=5.6	Focal length divided by effective aperture diameter indicating width of focal plane
initialFocusDistance	SFFloat	inputOutput	value=10	Distance to focal plane of sharpest focus
shotDuration	SFTime	outputOnly		Subtotal duration of contained Movement move durations
isActive	SFBool	outputOnly		start/stop yields true/false, useful to trigger external animations

Figure 6. Proposed X3D Shot Node Signature

5.3 Camera Node

Since we have taken a hierarchical approach to the Camera node construction, the Camera node itself contains only summary information about camera movement. Exposure of the position and orientation fields permits direct animation of Camera posture by external animation nodes. Binding a Camera node unbinds any bound Viewpoint, OrthoViewpoint or NavigationInfo node.

Camera : X3DBindableNode

<!-- Viewpoint-related fields -->				
Description	SFString	inputOutput		Descriptive summary
position	SFVec3f	inputOutput	value= 0 0 10	Camera position
orientation	SFRotation	inputOutput	value= 0 0 1 0	Camera rotation
fieldOfView	SFFloat	inputOutput	value= 0.7854	pi/4
set_bind	SFBool	inputOnly		
bindTime	SFTime	outputOnly		
isBound	SFBool	outputOnly		
<!-- NavigationInfo-related fields -->				
nearClipPlane	SFFloat	inputOutput		Vector distance to near clipping plane
farClipPlane	SFFloat	inputOutput		Vector distance to far clipping plane
headLight	SFBool	inputOutput	value= true	Camera headLight on or off
<!-- Camera-unique fields -->				
shots	MFNode	inputOutput	value=NULL	Array of Shot nodes, which contain Movement nodes
headLightColor	SFColor	inputOutput	value=1 1 1	Camera headLight color
filterColor	SFColor	inputOutput	value=1 1 1	Camera filter color to modify virtual lens capture
aimPoint	SFVec3f	inputOutput	value=0 0 0	Relative location for camera direction
upVector	SFVec3f	inputOutput		Any changes modify camera orientation
fStop	SFFloat	inputOutput	value= 5.6	Focal length divided by effective aperture diameter indicating width of focal plane
focusDistance	SFFloat	inputOutput	value= 10	Distance to focal plane of sharpest focus
isActive	SFBool	outputOnly		start/stop yields true/false, useful to trigger external animations
totalDuration	SFTime	outputOnly		Total duration of contained, enabled Shot durations
<!-- Offline rendering parameters -->				
offlineRender	MFNode	inputOutput	value=NULL	OfflineRender node(s)

Figure 7. Proposed X3D Camera Node Signature

The Camera node describes the initialization state of the camera and provides parameters important to the Shot and Movement constructions. Some of these parameters might change during the Shot or Movements, while others might not. Timing parameter totalDuration is calculated by summing the shotDuration of each contained Shot, which in turn is the sum of duration values for each contained Movement. There may be multiple Camera nodes, each computing independent totalDuration values.

5.4 Scripting and Advanced Techniques

5.4.1 Shaders for DOF Animation

As stated before, Camera animations are constructed from discrete Camera Movements, interpolators and ROUTEs. Internal Camera implementation functionality simply steps through the mechanics of interpolator construction. DOF is more complex and can have many implementations. Nevertheless, since the bindings from X3D to OpenGL Shading Language (GLSL), Microsoft High Level Shading Language (HSLS) and nVidia Cg shading language are well understood and supported by the X3D Specification, application of relevant algorithms appears to be feasible. This paper surveys various techniques, but implementation and evaluation remain as future work.

Animating the fieldOfView appears to be easier, though probably not as dramatic an effect. Since the focal length of the lens is approximated through the fieldOfView parameter and we do not consider the optical properties of the physical lens in its determination, an author might emulate a zoom effect by simply dollying the camera closer to the aimPoint. Zooming by increasing the fieldOfView attribute has the (possibly unintended) consequence of distortion.

5.4.2 X3D Follower Component for Animation

X3D nodes in the Follower component enable authors to dynamically create parameter transitions at run time by receiving a destination value, from which a set of smooth transition values is computed going from the current value towards the newly set destination value. These nodes can be considered direct, smoothed substitutes for interpolator nodes. PositionChaser and PositionDamper correspond to PositionInterpolator. OrientationChaser and OrientationDamper correspond to OrientationInterpolator. ScalarChaser corresponds to ScalarInterpolator for animating DOF and FOV.

Followers might be applied in several possible ways. The Camera point of view might chase an author-defined list of waypoints and orientations, or the camera aimPoint might chase a target as it moves. Thus it appears that the Follower nodes are well suited for camera and subject animation.

6. OfflineRender Node

Historically, X3D authors who want to record a video or take a snapshot must run separate screen-capture software while the user interacts with the X3D scene. This approach requires that the bound Viewpoint node be manipulated by the user or animated by the author.

Since camera shots and movements are well defined, the OfflineRender node is designed to provide the additional information needed to directly render movies or still images with complete precision. Previously, there simply was not enough information in an X3D scene to perform offline rendering satisfactorily. Rendering movement by movement and shot by shot (plus perhaps camera by camera) all corresponds nicely to how practical cameras work and how films are made.

OfflineRender : X3DChildNode

description	SFString	inputOutput		Descriptive summary
movieEnabled	SFBool	inputOutput	value= true	Whether this OfflineRender of a movie can be activated
ImageEnabled	SFBool	inputOutput	value= true	Whether this OfflineRender of an image can be activated
frameRate	SFFloat	inputOutput	value=30	Frames per second recorded for this rendering
frameSize	SFVec2f	inputOutput	value=640 480	Size of frame in number of pixels
pixelAspectRatio	SFFloat	inputOutput	value=1.33	Relative dimensions of pixel
set_startTime	SFTime	inputOnly		Begin render operation
progress	SFFloat	outputOnly		Progress performing render operation (0..1)
renderCompleteTime	SFTime	outputOnly		Render operation complete
movieFormat	SFString	initializeOnly	value= mpeg	Format of rendered output movie (mpeg, mp4, etc.)
ImageFormat	SFString	initializeOnly	value= png	Format of rendered output images
moviePath	MFString	initializeOnly	[url]	where rendered movies are written
ImapgePath	MFString	initializeOnly	[url]	where rendered images are written

Figure 8. Proposed X3D OfflineRender Node Signature

This capability satisfies an important use case. It now becomes easily possible to create a finished narrative using just the camera nodes and OfflineRender node, whereupon the X3D scene itself contains the motions needed to define camera work for a complete movie. We further suppose that X3D scenes might be rendered for editing in post-production. At that point, it becomes the video editor's job to put video sequences together to form a narrative. Even for this use case, carefully constructed offline renders of X3D camera animation might greatly simplify the post-processing of video in Non-linear Video Editing systems and even 3D graphics compositing with live action.

7. AN EXAMPLE SCENE

We consider the common need to construct a scene using avatars driven by behavior engines, where the author only has a general idea of where they might be. Figure 9 shows how such a typical scene might be staged and described in X3D using Camera, Shot and Movement nodes.

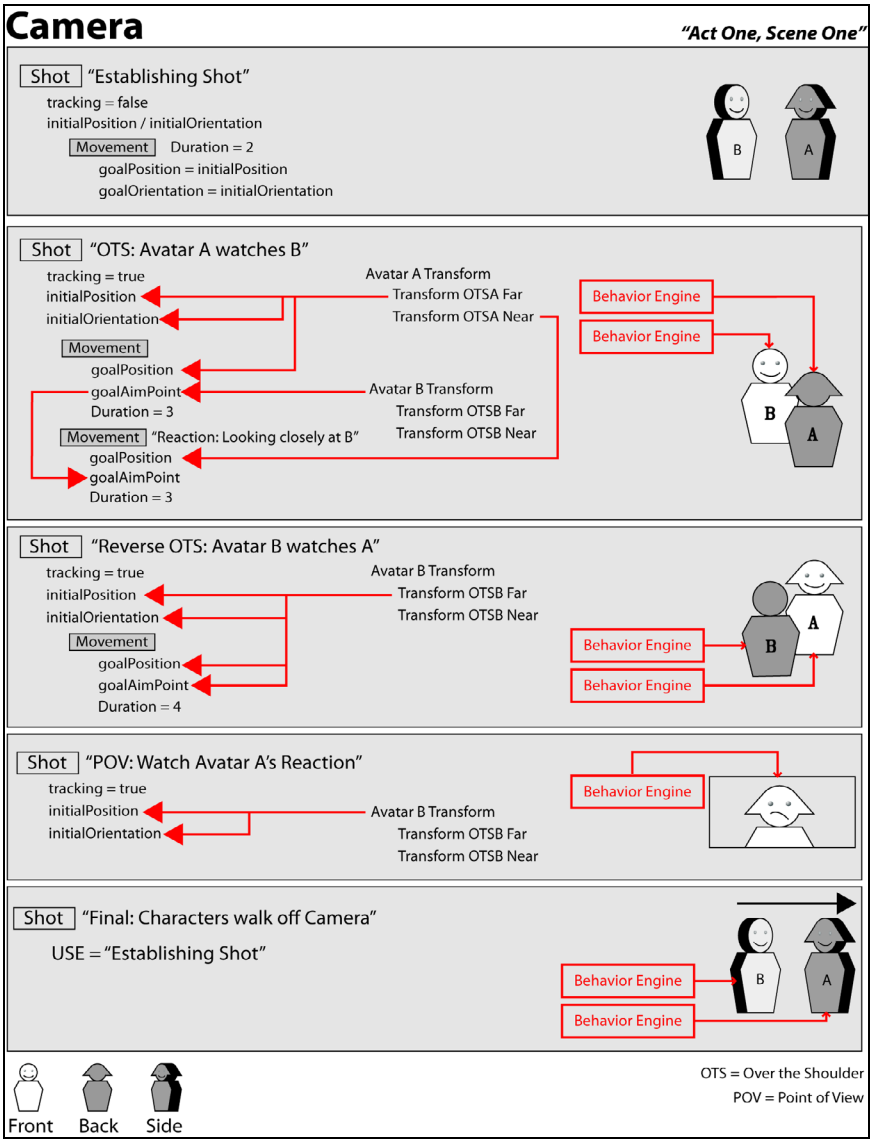

Figure 9. Directing Scene Animations via Camera, Shot and Movement Nodes

8. FUTURE WORK

The proposed Camera, Shot and Movement nodes for X3D simplify camera animation and support sophisticated DOF rendering. They are, easily authored and applied for both real-time and offline rendering. The proposed OfflineRender node provides the information necessary to produce multiple format outputs in an offline mode.

More work to produce numerous examples and establish good practices is needed to ensure that these capabilities are thoroughly designed and fully capable of meeting diverse authoring requirements. We hope that implementation, evaluation and optimization provides a path towards eventual adoption as part of the X3D Specification.

9. REFERENCES

Bertalmio. M., Fort, P. and Sanchez-Crespo, D. 2004. Real-time, Accurate Depth of Field using Anisotropic Diffusion and Programmable Graphics Cards. *In Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'04)*

Brutzman, Don and Daly, Leonard, X3D: Extensible 3D Graphics for Web Authors, Morgan Kaufmann Publishers, 2007. Examples, slidesets and video available via http://www.x3dGraphics.com

Cook, R., Porter, T. and L. Carpenter. 1984. Distributed Ray Tracing. In *Computer Graphics (Proceedings of SIGGRAPH 1984)* vol. 18, ACM, 137-145.

The Cutting Edge: The Magic of Movie Editing. Wendy Apple. Wendy Apple. Kathy Bates, Jody Foster, Quentin Tarrantino, Walter Murch, James Cameron. Feature Documentary. A.C.E, 2004.

Demers, J. 2002. Depth of Field: A Survey of Techniques, GPU Gems, Chapter 23

Demers, J. 2003. Depth of Field in the 'Toys' Demo. From "Ogres and Fairies: Secrets of the NVIDIA Demo Team," presented at GDC 2003.

Haeberli, P., and Akeley, K. 1990. The Accumulation Buffer: Hardware Support for High-Quality Rendering. Computer Graphics 24(4).

Pocock, L., and Rosebush, J. The Computer Animator's Technical Handbook. New York, New York: Morgan Kauffman, 2002.

Pixar's Renderman. Teapot and Box. Rendered in Maya 2008.

Potmesil, M., and Chakravarty, I. 1982. Synthetic Image Generation with a Lens and Aperture Camera Model. ACM Transactions on Graphics, April 1982.

Savage X3D Model Archive https://savage.nps.edu/Savage

Scofield, C. 1994. 2½-D Depth of Field Simulation for Computer Animation. In Graphics Gems III, edited by David Kirk. Morgan Kaufmann.

Web 3D Consortium, Extensible 3D (X3D) Graphics Standard. http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification

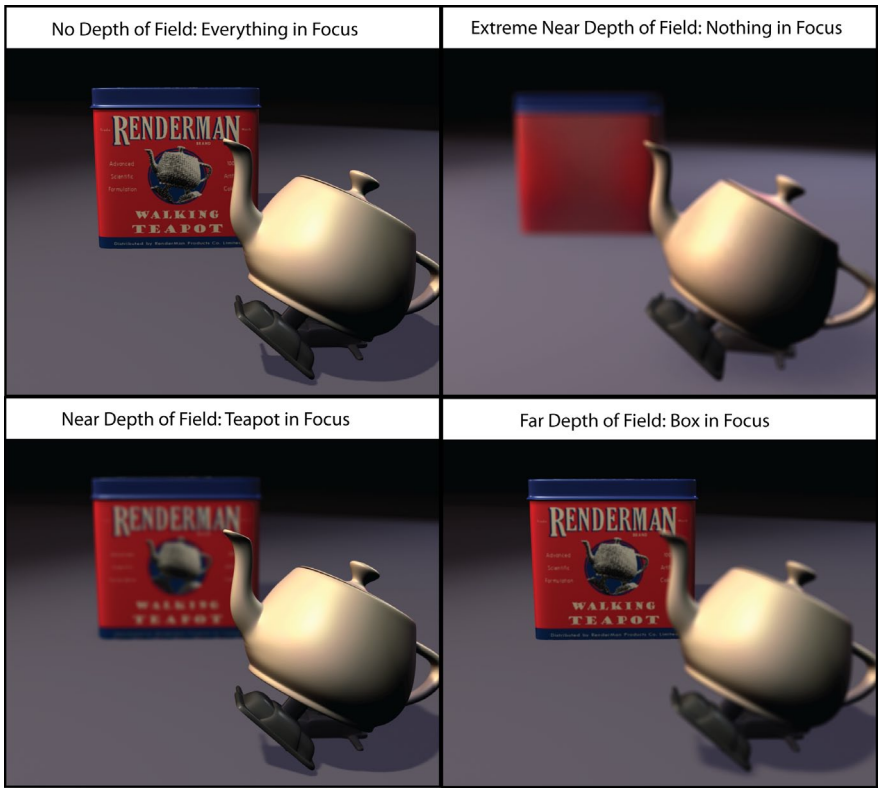

Figure 10. Depth of Field (DOF) Image Examples, Produced using Pixar's Renderman