The 12th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST
(2011)
M. Dellepiane, F. Niccolucci, S. Pena Serna, H. Rushmeier, and L. Van Gool (Editors)

VSim: Real-time Visualization of 3D Digital Humanities Content for Education and Collaboration

Eduardo Poyart[1], Lisa Snyder[1], Scott Friedman[1], Petros Faloutsos[1,2]

[1] University of California, Los Angeles
[2] York University

Abstract

This paper presents VSim, a framework for the visualization of 3D architectural and archeological models. VSim's design focuses on educational use and scholarly collaboration, an approach that is not commonly found in existing commercial software. Two different camera control modes address a variety of scenarios, and a novel smoothing method allows fluid camera movement. VSim includes the ability to create and display narratives within the virtual environment and to add spatially localized multimedia resources. A new way to associate these resources with points and orientations in space is also introduced.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Application packages—Software support

1. Introduction

The study of built structures is fundamental for humanistic inquiry. Architecture and urban design reveal the aspirations and priorities of cultures across the ages, and allow us to understand these cultures. If one wants to study the ancient Egyptians, for example, it is important to examine the pyramids and religious complexes of the Nile river valley.

Architects and archeologists have used 3D computer modeling to visualize un-built structures and reconstruct cities of the past. But these efforts have yet to move from the research lab to mainstream scholarship and collaboration across disciplines. A secondary problem is that the computer models, by themselves, are essentially raw data, lacking contextual material, subject expert commentary, and textual analysis that could make them engaging and effective tools for teaching and learning [SF10].

This paper describes VSim, a collaborative framework and real-time visualization software that addresses these questions. We address the difficulty in sharing digital architectural content by providing a user interface that is intuitive and effective both for spatial navigation in the 3D world, and for the user's interaction with the software. To address advantages and disadvantages inherent to different camera control methods, a switchable camera control scheme was used, allowing the user to select between two modes. A novel cameral control method used in one of these modes ensures smooth camera motion.

We also propose a system for the end-user to create narratives which provide predefined navigation paths in the 3D environment, and to add contextual material consisting of text, images, videos, sounds and web links. This system was designed to minimize the entry barrier for new users, while still being powerful and flexible. We also provide the possibility of adding spatially-localized resources, which do not follow a narrative but are presented to the user at the appropriate geographic locations. Finally, we discuss the important issue of enforcing copyrights and branding for the content used in VSim.

VSim is based on a non-trivial integration of known techniques that have been adapted to our problem domain of 3D Humanities Content Visualization. Some of these adaptations are novel and a contribution in their own right. Our contributions can be summarized as follows:

1. We present VSim, a collaborative framework for real-time visualization of 3D digital humanities content.
2. We propose a novel camera motion control approach that ensures smooth camera motion during the navigation of a 3D scene.
3. We propose a narrative creation system for end users.
4. We present an approach to embedding spatially-localized resources into the 3D environment.

The remainder of this document is organized as follows. Section 2 reviews related work. Section 3 presents our two approaches for camera control and navigation in the 3D environment. Section 4 describes the system for creating narratives, and Section 5 discusses the embedded resources. In Section 7 we describe our graphical user interface, its model of smooth motion, and how it can be adapted to touch-screen interfaces. Section 8 concludes the paper.

2. Related work

Existing run-time software interfaces for real-time models are inadequate, either because they lack the functionality necessary for educational use or they are obsolete.

Two pieces of real-time software for exploring three-dimensional models have previously been developed at UCLA, but both have reached the end of their usefulness. Active development stopped over five years ago on the original proprietary software developed for the Urban Simulation Team (*uSim*) and it is only available for Linux. Similarly, active development was stopped on the freely available software developed by UCLA's Academic Technology Services (*vrNav*) because of the complexities of improving its interface, adding functionality, and maintaining the requisite software dependencies. Many aspects that influenced our research were studied during development of uSim [JLF96, JLF95, LFJ95, Fri94].

None of the desktop navigational applications available for purchase or free download have been built specifically for educational use of the virtual environments, nor do they take advantage of new advances in graphics hardware and software. Google Earth [Gooa] allows the visualization of environments that can be built by the user with SketchUp [Goob], but it lacks many of the aspects that we needed, among them easy creation and distribution of narratives with seamless on-screen content, smooth camera navigation adequate for a classroom setting and built-in ability to create and export movie files.

The Center for Advanced Spatial Technologies of the University of Arkansas (`http://researchfrontiers.uark.edu/15683.php`) is developing a virtual environment of the city of Pompeii. The software uses the commercial 3D game engine Unity, developed by Unity Technologies (`http://unity3d.com`). This is in line with our philosophy of taking advantage of people's familiarity with video games as virtual reality environments. In our case, however, the need to not be tied to a closed-source product and to be able to load models from several different file formats led us to the choice of OpenSceneGraph [Osf] as the underlying library for efficient model rendering.

Camera control has been studied with various approaches, including: a declarative control language in which camera actions are textually described in a non-interactive way [CAH*96], an automated camera planner making real-time decisions based on predefined high-level information provided textually by the user [BL97], and camera control by hierarchical finite state machines, which are a combination of low-level "camera modules", controlling geometric placement of the camera, and high-level "idioms", selecting camera modules and timing between shots [HCS96].

In [TBGT91], a dynamic, rather than kinematic, camera control system is introduced. We found that, for our purposes, such controller would always have to be over-damped, so as to avoid oscillations around the desired position. Our first-person camera smoothing system, described in Section 3.2, can be considered similar to an over-damped dynamic system, but implemented in a simple and efficient way.

In [ZF99], the camera is controlled using the mouse and a single button. This is achieved through mouse gestures that perform different control functions. Depending on the way the user initiates the gesture (e.g. vertically vs. horizontally), a different type of control is engaged (e.g. zoom vs. translation on film plane).

3. Navigation in the 3D environment

From a computer graphics point of view, the problem of navigating in a 3D environment is essentially one of controlling camera position and orientation. These camera parameters have to be updated at every frame, while taking into account user input (keyboard, mouse, joystick and touch-screen devices), and software-driven events (e.g. collision with geometry, or the playback of a narrative).

Video games are a major portion of the entertainment industry, and considering that many video games are essentially 3D virtual environments, it is natural to look at them for inspiration. Furthermore, users already familiar with video-game-style controls, when faced with the transition to a new software, will feel natural if the control mechanics are similar. However, simply reproducing video-game-style controls doesn't solve all of the problems identified for our real-time visualization software. In particular, while video-game controls are suitable to fast-action aiming at targets, they are not suitable for recording movies with smooth motion, or for pedagogical applications and large-audience situations.

User studies with university faculty made before the development of VSim have covered ease of navigation and use in a classroom setting. Faculty interviewed was, in general, hesitant to try to navigate the 3D models themselves during classes while they talked about them at the same time. The navigation system present here, including the camera controls and the narrative system, was designed to give instructors a non-threatening way to engage with the models as instructional technology.

We have decided to provide the user with two camera control options, called the *uSim* mode and the *first person* mode.

These modes are described below. Both of them follow the "flying vehicle control" metaphor described in [WO90].

3.1. uSim mode

This mode received this name because it is similar to the control mode used in *uSim*, a previous visualization software developed by UCLA [JLF95]. The main goal of this mode is to provide movement that is as smooth as possible. A secondary goal is to allow the user to control both camera orientation and velocity with a single hand on the mouse, an important feature for teachers and lecturers that were interested in using the software as a tool during a classroom presentation.

In this mode, we define a point $\mathbf{c}$ in two-dimensional space, residing at the center of the screen. Let $\mathbf{m}$ be the mouse position in the same coordinate system. As the user moves the mouse away from $\mathbf{c}$, the vector $\mathbf{v}_{usim} = \mathbf{m} - \mathbf{c}$ defines a direction and magnitude. The camera movement at each time interval dt (e.g. at each frame) is defined in the following way: the horizontal component of $\mathbf{v}_{usim}$ defines a rotation around the vertical (y) axis, and its vertical component defines a rotation around the horizontal (x) axis. The y axis is defined in a world coordinate system, always pointing up. The x axis is defined in a local, camera-centric coordinate system.

With this system, the mouse position on the screen defines the angular velocity of the camera. An interesting consequence is that even if the user does not provide very smooth movement with his/her hand on the mouse (which is in fact hard to do), smoothing is a result of the fact that his/her hand movements are translated to a first-order derivative of the camera orientation. Any jerkiness is translated to jerkiness in velocity, not in position. Integrating in time the velocity to compute the position has a natural smoothing effect. In both this mode and the first-person mode described below, the spacebar key toggles between the camera-control mode and the mouse-release mode, the latter meaning that the mouse is no longer used to control the camera, and can be moved around to click at user-interface elements like menus and icons.

Camera linear velocity is controlled by the mouse buttons: the left button accelerates and the right button decelerates. In both modes, the user can choose between being allowed to fly, in which case the camera motion vector always coincides with the eye forward vector, or being constrained to ground level, in which case the camera motion vector is a normalized horizontal projection of the eye forward vector. A collision detection system ensures that the camera is always at a constant distance from the ground. If the user attempts to climb over a step that is small enough, the camera is allowed to climb, updating its height for the next section of terrain.

3.2. First-person mode

In this mode, mouse movements are directly translated to camera movements. If the mouse moves from position $\mathbf{m}_0$ to position $\mathbf{m}_1$, the vector $\mathbf{v}_{fp} = \mathbf{m}_1 - \mathbf{m}_0$ defines the camera rotation if applied in a similar way as $\mathbf{v}_{usim}$, described above. If the user stops moving the mouse, $\mathbf{v}_{fp} = 0$ and the camera stops moving.

This is the control method used in most first-person computer video games. Notice that in this mode, mouse movements are directly translated to the camera orientation, rather than to its first-order derivative. Any jerkiness in mouse movement is translated directly to jerkiness in camera rotation. On the other hand, an advantage of this mode is that the user can quickly point the camera to any direction, as if he/she is turning his/her head. It is arguably more intuitive to think that one's hand is directly controlling the look-at vector, rather than to think that it is controlling the camera's angular velocity.

In order to improve the smoothness of the camera's motion in this mode, without compromising the advantage mentioned above, we proceeded as follows. Let p_0 and w_0 be, respectively, the pitch and yaw components of the camera's orientation at time t_0, as defined with respect to an arbitrary global orientation system. The movement vector $\mathbf{v}_{fp}$ is then translated into "desired" values, $p_{desired}$ and $w_{desired}$. The current p_0 and w_0 values are interpolated towards $p_{desired}$ and $w_{desired}$ following an inverse exponential law:

$$p(t) = p_0 e^{-jr} + p_{desired}(1 - e^{-jr}), \quad r = t - t_0 \tag{1}$$

where t is the current time and j is a scaling constant. An analogous formula is used for the yaw w. An efficient stepwise integration method to approximate this behavior can be implemented as follows. At each frame (a time interval Δt), p is moved towards $p_{desired}$ a distance Δp proportional to the magnitude of the distance between them, i.e.:

$$\Delta p = k \cdot \Delta t \cdot |p_{desired} - p| \tag{2}$$

We had best results with $k = 8.0$. This value can be intuitively understood the following way: if the speed were to be defined on the first frame and kept constant, p would take 1/8th of a second to reach $p_{desired}$. The actual time is larger, since the frame rate is usually higher than 8 fps. Even though mathematically this would result in motion that gets slower and slower but never stops, in practice the limits of floating-point accuracy are reached quickly, which makes the motion stop. A threshold point could have been introduced, but, in our tests, we didn't have any noticeable residual motion when the user stopped moving the mouse, so the threshold test was not necessary.

This method essentially absorbed a large amount of high-frequency, unintended mouse jerkiness and provided a smoother movement while keeping high interactivity.

Linear velocity of the camera is controlled by the keys **w** (forward), **a** (left), **s** (back) and **d** (right). This control scheme is used in numerous first-person computer games. However, in our case, the velocity doesn't change immediately, but linearly increases and decreases over a short period of time, limited by a maximum velocity. We used 0.66s for this time value, and our conclusion is that it should be kept small. We essentially follow the same philosophy as for camera rotations: there should be smoothness in motion, but at the same time the user should feel crisp and responsive camera control.

4. Narratives

Another feature of our framework is the ability to create *narratives*: a mechanism for both the content contributor and the end-user to create arguments, tours and lesson plans, augmented with on-screen multimedia content. The user adds *narrative nodes* (keyframes) defining the camera position and orientation, and organizes these nodes in a timeline. Figure 1 shows the narrative editor UI and its constitutive elements. Figure 2 is a detailed view of part of the narrative editor bar.

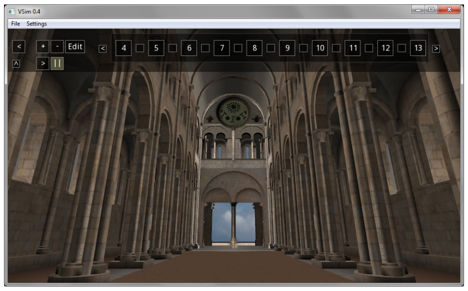

Figure 1: *The narrative editor. The central area of the top bar contains narrative nodes and transitions. The control areal on the left contains buttons for creating and removing nodes. The Edit button launches the overlay editor for the selected node. Model of the Romanesque Cathedral of Santiago de Compostela, courtesy of John Dagenais, UCLA.*

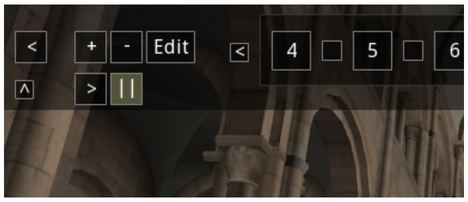

Figure 2: *A detailed view of the narrative editor bar.*

Nodes in a narrative are created by simply positioning the camera and clicking on the "+" button. With this action, a narrative node is added to the timeline. Data associated with nodes include: camera position and orientation, time to remain in the node and node overlay content (described in Section 4.2). The user can edit the timeline by directly manipulating and reorganizing the nodes.

When playing the narrative, the camera position is interpolated between nodes in the manner described below.

4.1. Camera movement

In the current version of the software, the camera follows straight lines between nodes. We have plans to add second-order continuity at nodes, but this was not necessary in the first version, since users were mostly interested in stopping at nodes and showing content there.

Within the straight line followed by the camera, its position is interpolated in an ease-in/ease-out fashion using a cubic function as described below. We start with the following function:

$$y = x - x^3 \tag{3}$$

The x axis is scaled so that the inflection points $x = \pm 1/\sqrt{3}$ fall at 0 and 1, and the y axis is scaled so that the minimum and maximum in that range also fall at 0 and 1. We input a linear interpolant into this function as x, with values ranging from 0 to 1. The output y is a cubic interpolant that is used both for camera position and rotation (as a quaternion *slerp* interpolant). By using this method for ease-in/ease-out instead of a quarter-sine, we avoid the expense of a sine computation.

4.2. Overlays

Each keyframe, or node, can be enriched with textual information, images, videos and sounds – called *overlays*. The user can freely lay out these elements on the screen in 2D space while in a keyframe. When a narrative is playing and a keyframe is reached, its overlay fades in and is displayed for an user-determined time, or until the user presses a key. This function essentially allows the content contributor and end users to augment the virtual world with multi-media content. Figure 3 shows the overlay editor with a text element added to the scene.

5. Embedded Resources

Users may wish to explore the world through free navigation, i.e. not following a narrative. It is natural to think that these users should also be presented with contextual multimedia elements (*embedded resources*) associated with geographical locations in the environment. As an example, when

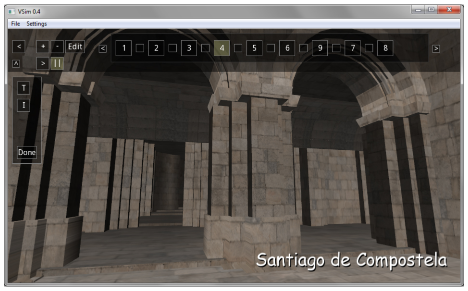

Figure 3: *The overlay editor.*

the user is viewing a reconstruction of a historical building, an embedded resource could show documents related to the original construction of this building: blueprints, photographs, letters, videos of the actual site and so on. Due to the fact that the user should not be interrupted in his/her continuous navigation, embedded resources don't pop up on screen, but rather, icons appear on the *embedded resource bar* at the bottom of the screen. The user can choose to view them or not.

An important question is how to define the area in which an embedded resource icon should appear. A naive approach would be to associate the resource with a point in space, and make it appear whenever the camera is inside a sphere of a certain radius, centered at this point. However, this would show resources that are behind the user. Another approach is to combine the sphere with a specific camera angle, which doesn't entirely solve the problem, as the user could be, for example, at the side or back of a building, and the content creator only wants to associate a resource with the front side. Figure 4 illustrates this problem.

The approach that we took is described as follows. The content creator defines a resource's location in the environment through a data structure composed of:

1. Target **t** (3D position).
2. Camera position center **c** (3D position).
3. Camera area radius r (scalar value).

Whenever the following conditions are true, the embedded resource becomes available to the user and appears in the embedded resources bar:

1. The scene has the target point **t** in the field of view.
2. The camera is inside a sphere of radius r centered at **c**.

Past approaches essentially had the center **c** and target **t** combined as the same value. By separating them, we allow resources to appear whenever the user is, for example, *in front* of a building and looking *at* the building. Figure 5 shows the top view of a street (parallel lines) and a building (square). Points **c** and **t** are shown, as well as the sphere defined by radius r. If the user, in his/her virtual walk on the street, steps inside the sphere and turns the camera towards

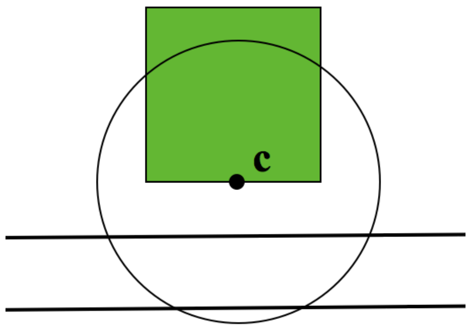

Figure 4: *Embedded resource positioning: less-than-ideal solution. This is a top-down view of a building (square) and a street (parallel lines). A point in space* **c** *and a radius r are used to describe the area of interest. This point is both the point of interest and the center of camera influence. Notice that the user can see resources associated with the front facade (the side closest to the street) when he is at the side of the building or even inside of it, which the author, in general, does not want to happen.*

the building, both conditions are met and the embedded resource appears. This corresponds to the author's intention – he/she can now express the fact that the resource should appear when the user is in front of the building, looking at the front facade.

If desired, a flag can be added to the embedded resource data structure to inhibit the need for a target; this would allow the authoring of resources that "attract attention", independent of the direction the user is looking; however we have concluded that this mode should not be the default.

An inside-sphere test has to be made on every frame and for each resource, and there could be many resources associated with a scene. The use of spheres makes this computation efficient. Other, more complex shapes could be used as area of influence, but spheres are adequate in most cases, and they also simplify a content creator's method of thinking about the problem.

6. Restrictions and enforcing copyright

We incorporated a lightweight model of restrictions, through which the content creator can have some level of assurance that his/her content will not be easily modified, and that his/her branding elements will not be easily removed.

The content creator can set a series of flags that are associated with his redistributable file (a file that combines the 3D model with meta-information about narratives and embedded resources). These flags enable or disable VSim functionality for, among other things:

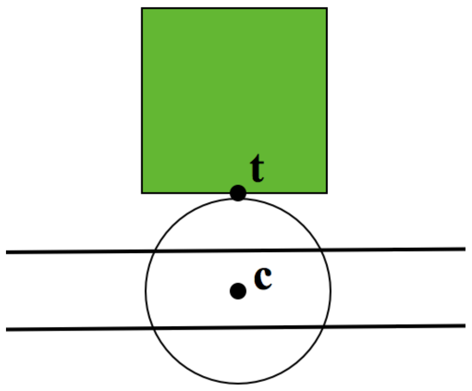

Figure 5: *Embedded resource positioning: our solution separates the center* **c** *from the target point* **t**. *The embedded resource is activated if the camera is inside the sphere and the target* **t** *is in the field of view. The author can express the fact that the resource should appear when the user is in front of the building and looking at the facade.*

1. Adding and removing a branding bar, which can show, for example, the institution logo.
2. Modifying narratives.
3. Modifying embedded resources.
4. Keeping the user restricted to paths defined by narratives, i.e. no free-form navigation.
5. Creating video files.
6. Increasing the screen resolution beyond a certain limit.

The initial implementation is lightweight, and it is a step toward a more secure implementation. We decided not to add encryption and the associated burden of encryption keys. Taking into account the fact that the software is open-source, it is not impossible for someone with the source code in hand to circumvent the restrictions. However, it is not easy for someone without programming expertise to do so, which is adequate for our requirements. Security can be enhanced in future versions by means of encrypted data files, building on top of our current system.

7. Graphical user interface

Two ideas guided the graphical user interface design: it should be intuitive and easy to use, and it should be adaptable to emerging technologies such as tablet devices and touch-screen interfaces.

We envision a typical user of the software to be a lecturer in a discipline in humanities. This user may have little or no experience with other 3D navigation software and with tools to create narratives and movies. We designed a *narrative bar* residing on a *control bar* at the top of the screen. It consists of *nodes* (large rectangles corresponding to keyframes, containing a thumbnail of the scene) and *transitions* (smaller rectangles in between nodes). Accessing a node opens up a window to control node aspects (time to remain in node, fade in/out time, and others). Accessing a transition, similarly, allows the user to control variables such as transition timing. Adding and removing nodes is done through buttons labelled "+" and "-", a concise and efficient representation.

All GUI animations are smooth. For example, the side scrolling of the narrative bar (when there are more nodes than the screen can fit) is visually a rolling movement rather than a jump. This gives the user important visual cues. Similarly, the whole control bar (which is semi-transparent) can be closed to reveal the whole 3D scene, also with a smooth movement. We found that the most pleasant motion was achieved not linearly, but by making the speed follow an inverse exponential law, similarly to the first-person camera (equations 1 and 2).

The UI was designed on a style that is adequate for porting of the software to tablet platforms. All interactive elements (widgets) are large (by being "finger-sized", they can be used with touch-screen input, as opposed to mouse clicks). Rather than separate windows with OS-specific features, the widgets are rendered using OpenGL on top of the 3D environment. A custom widget system was developed for that purpose. By sidestepping this OS dependency, the widget system should work out-of-the-box on touch-screen platforms, and the look-and-feel is preserved. Minor OS-specific elements are still used, e.g. on dialog boxes to open files, which shouldn't present a big portability problem.

8. Conclusion and future work

We have presented VSim, a framework for visualizing 3D architectural and archeological content, enhanced with creation of narratives and integration of embedded resources. VSim is being used at UCLA in an experimental phase for classroom presentations. Initial response of users that have been exposed to the prototype software has been positive.

Camera control was one of the questions approached during development. There are advantages and disadvantages to both camera control modes that were tested, so we followed the route of implementing both, and allowing easy switch between them. The *first-person* mode is easy to use and provides fast and direct camera control. The *uSim* mode, on the other hand, has smooth motion appropriate for video recording, and one-hand control appropriate for classroom presentations. We believe that a combination of both modes was the ideal solution.

We have also approached GUI aspects, with an aim to keep the software portable and adaptable. By developing a custom widget library in OpenGL, we have seamless integration of UI elements with the 3D environment, and the possibility for future adaptation for touch-screen interfaces.

We found that the benefits of developing such a custom library offset its cost.

Narratives allow end users and content creators to provide argumentation and story-telling within the 3D environment. Embedded resources allow the augmentation of the 3D environment with relevant content for users that choose to perform free navigation. The elegant scheme developed for the activation of embedded resources based on camera position and orientation has met all of our requirements, allowing the content creator to express when and where the resources should appear in a variety of scenarios.

As future work, we have plans for adding pre-processing modules to increase the realism of the scene, which can be used to provide ambient occlusion and other physically-based lighting effects. The system can also be used as a platform for the integration of physical simulation modules, for example to simulate earthquakes and other natural disasters and study their effects, as well as erosion and weathering of materials.

References

[BL97] BARES W. H., LESTER J. C.: Cinematographic user models for automated realtime camera control in dynamic 3d environments. In *Proceedings of the Sixth International Conference on User Modeling* (1997), Springer, pp. 215–226. 2

[CAH*96] CHRISTIANSON D. B., ANDERSON S. E., HE L.-W., SALESIN D. H., WELD D. S., COHEN M. F.: Declarative camera control for automatic cinematography. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1* (1996), AAAI '96, AAAI Press, pp. 148–155. 2

[Fri94] FRIEDMAN S.: Large scale urban visualization. Technical Report, UCLA Department of Architecture, 1994. 2

[Gooa] GOOGLE: Google earth. http://earth.google.com. 2

[Goob] GOOGLE: Google sketchup. http://sketchup.google.com. 2

[HCS96] HE L.-W., COHEN M. F., SALESIN D. H.: The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 217–224. 2

[JLF95] JEPSON W., LIGGETT R., FRIEDMAN S.: An environment for real-time urban simulation. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (New York, NY, USA, 1995), I3D '95, ACM, pp. 165–ff. 2, 3

[JLF96] JEPSON W., LIGGETT R., FRIEDMAN S.: Virtual modeling of urban environments. *Presence: Teleoperators and Virtual Environments* (1996), 72–86. 2

[LFJ95] LIGGETT R., FRIEDMAN S., JEPSON W.: Interactive design/decision making in a virtual urban world: Visual simulation and gis. In *15th Annual ESRI User Conference* (1995), pp. 22–26. 2

[Osf] OSFIELD R.: Open scene graph. http://www.openscenegraph.org. 2

[SF10] SNYDER L., FRIEDMAN S.: Enhancing the humanities through innovation. Grant proposal for the National Endowment for Humanities, UCLA, 2010. 1

[TBGT91] TURNER R., BALAGUER F., GOBBETTI E., THALMANN D.: Physically-based interactive camera motion control using 3d input devices. In *Scientific Visualization of Physical Phenomena (Proceedings of CG International Õ91* (1991), Springer, pp. 135–145. 2

[WO90] WARE C., OSBORNE S.: Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of the 1990 symposium on Interactive 3D graphics* (New York, NY, USA, 1990), I3D '90, ACM, pp. 175–183. 3

[ZF99] ZELEZNIK R., FORSBERG A.: Unicam – 2d gestural camera controls for 3d environments. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (New York, NY, USA, 1999), I3D '99, ACM, pp. 169–173. 2