Multiscale 3D Navigation

James McCrae, Igor Mordatch, Michael Glueck, Azam Khan

Autodesk Research
210 King Street East, Toronto, Ontario, Canada
first.last @autodesk.com

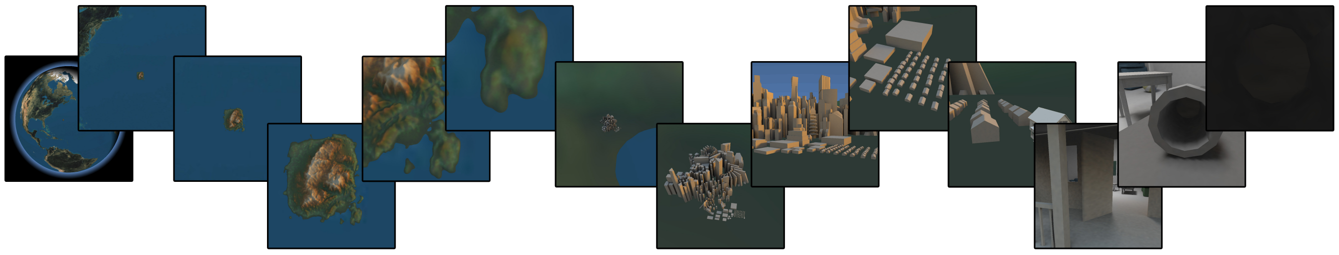


Figure 1: Our scale-sensitive approach allows navigation between scales in 3D scenes. Here, the user navigates from thousands of kilometres above Earth's surface to come to rest inside a jug on a table only centimetres in diameter.

Abstract

We present a comprehensive system for multiscale navigation of 3-dimensional scenes, and demonstrate our approach on multiscale datasets such as the Earth. Our system incorporates a novel image-based environment representation which we refer to as the *cubemap*. Our cubemap allows consistent navigation at various scales, as well as real-time collision detection without pre-computation or prior knowledge of geometric structure. The cubemap is used to improve upon previous work on proximal object inspection (*HoverCam*), and we present an additional interaction technique for navigation which we call *look-and-fly*. We believe that our approach to the navigation of multiscale 3D environments offers greater flexibility and ease of use than mainstream applications such as Google Earth and Microsoft Virtual Earth, and we demonstrate our results with this system.

CR Categories: H.5.2 [User Interfaces]: Graphical User Interfaces (GUI)—3D graphics;

Keywords: 3D navigation, 3D widgets, Desktop 3D environments, virtual camera.

Copyright © 2009 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212) 869-0481 or e-mail permissions@acm.org.
I3D 2009, Boston, Massachusetts, February 27–March 1, 2009.
© 2009 ACM 978-1-60558-429-4/09/0002 $5.00

1 Introduction

As computers continue to grow in processing and storage capacity, users continue to push the limits creating ever larger data sets that include both large structures as well as small details. A particularly compelling large multiscale data set is the Earth itself and the geospatial information of human activity, including global political boundaries, roads and motorways, and buildings. With the advent of Google Earth and Microsoft Virtual Earth, 3D urban visualization and navigation has become mainstream. There are several scales at which urban environments can be meaningful to users; at the city scale, neighborhoods, street level, a single building or home, and a single apartment or room inside a structure. However, typical 3D software applications do not account for the scale of the environment within their navigation tools. Some applications simply do not provide a means for users to modify the navigation tools in the ways needed to support multiscale navigation. Those that do, bury these options in several layers of dialog boxes. Even so, users will need to continuously tweak the settings as the scale of the environment changes, adding significant effort to the users task. With the intent of providing a seamless multiscale 3D navigation user experience going from the planetary scale down to an individual building and then moving about inside the building (see Figure 1), we have developed a set of tools that automatically sense the size of the environment and adjust the viewing and travel parameters accordingly. Through simple mouse-based controls, a user can not only navigate through complex environments, but also fly around objects to inspect them, regardless of size, shape, or scale. To help achieve this goal, we introduce a new GPU-based environment-size algorithm to sense and respond, with control-display (C:D) ratio manipulation and collision-avoidance, to the scale of the surrounding environment.

2 Related Work

Virtual 3D environments can be immersive, such as Virtual Reality, or desktop based, such as Google Earth. Interaction is typically controlled through a 6 degree-of-freedom device, such as a bat [Ware and Osborne 1990], or a 2D device, such as a mouse. While we review work covering all combinations of environments and controllers, our current research focuses on the presentation of a virtual 3D environment in a desktop setting, using a mouse for input.

Navigation in 3D virtual space has been the focus of a significant body of research. Many have investigated mimicking real-world navigation experiences to help users explore virtual worlds. Early works focused on using metaphors to help users better understand the way in which their interactions would take them through virtual space. Egocentric navigation describes the interaction where a user wants to move through a space, while exocentric navigation pertains to a user moving around an object. [Ware and Osborne 1990] proposed three general interaction paradigms for 3D virtual environments: eyeball-in-hand, scene-in-hand, and flying vehicle control (flying). Flying was found to best support egocentric navigation, while scene-in-hand best suited exocentric inspection. Despite the many metaphors available to help users understand navigation, the problem has not been reduced to an intuitive and easy task [Fitzmaurice et al. 2008]. Navigation aids have also been developed, to help users better understand the layout and context of their environment. [Darken and Sibert 1993] provides an overview of many such techniques, including landmarks, breadcrumbs, and maps.

Another school of thought proceeds that the authors of a virtual environment know best what the points of interest are, and should therefore assist users in reaching them. [Haik et al. 2002] showed that by sacrificing freedom of navigation, users completed navigation tasks more quickly and were able to locate and view selected features of a scene without getting lost or disoriented. ShowMotion [Burtnyk et al. 2006] and Magallanes [Abásolo and Della 2007] provide authored viewpoints to users, while StyleCam [Burtnyk et al. 2002] and the work of [Hanson and Wernert 1997; Hanson et al. 1997] limited camera movement to two dimensional motion along 3D surfaces. Path-based camera constraints have been presented [Salomon et al. 2003; dos Santos et al. 2000], with some even allowing users some local control to deviate from the path [Galyean 1995; Hanson et al. 1997; Elmqvist et al. 2008]. Finally, others have explored determining optimal camera positions based on content-based constraints [Drucker and Zeltzer 1995; Bares and Lester 1999; Bares et al. 2000], and even moving the camera by manipulating objects being viewed [Gleicher and Witkin 1992]. While these systems focus on navigation through large 3D data sets, they are not sensitive to the scale of the environment. For example, when moving to a landmark of a very small object does not change the C:D ratio of the freeform camera tools.

While authored solutions do minimize the possibility of confusion in a 3D virtual environment, they require the author to determine what will be of interest to the user. To avoid this overhead, but still benefit from constrained navigation, more general solutions have been sought after. The ViewCube [Khan et al. 2008] is a widget that supports constrained exocentric inspection of an object, ensuring the entire object is visible in the viewport. HoverCam [Khan et al. 2005] allows users to inspect objects exocentrically by panning and zooming, as the camera automatically reorients itself to view the surface perpendicularly.

[Igarashi et al. 1998; Cohen et al. 2000] provided tools for users to define constrained camera paths, [Ropinski et al. 2005] automatically generated paths for roads in cities, and [Li and Ting 2000] developed a system for navigating through environments using probabilistic path planning. Finally, collision detection has been used as a means of guiding users along uneven terrain [Steed 1997]. Force-fields have also been investigated as a means of both preventing collisions and guiding users through obstacles [Li and Hsu 2004; Li and Chou 2001; Xiao and Hubbold 1998]. While these techniques have been shown benefit users, each have addressed either exocentric or egocentric navigation, but not both. Speed-coupled flying with orbit is a technique developed by [Tan et al. 2001] that allows users access to both egocentric and exocentric navigation, but requires a discrete modal change of interaction to do so.

As graphics hardware has advanced, techniques to off-load computation to the GPU have been developed. In particular, there is a growing body of research into image-based solutions to solve such problems as calculating proximity, collisions, and intersections. Early work [Baciu and Wong 1997; Baciu et al. 1998; Myszkowski et al. 1995] used multipass rendering of depth maps to calculate collisions between convex objects. [Fan et al. 2004] rendered six view directions inside a convex object to determine when another object intersects. [Kolb et al. 2004] implemented fragment shaders to render depth maps and encode normal vectors to simulate colliding particles in real time. [Vassilev et al. 2001] used a similar approach to animate cloth colliding against moving avatars. [Winter and Stamminger] used depth maps to calculate collisions between user controlled avatars and virtual environments.

Many points of user confusion, when navigating in 3D, are related to the multiscale nature of a 3D virtual environment. An effect described as desert fog occurs when a user is either too close to a low-detail object or too far from a high-detail object, causing confusion and disorientation resulting from a loss of context [Jul and Furnas 1998]. In addition to the importance of detail to provide context, speed of motion has also been identified as being important to effectively navigate 3D environments. [Mackinlay et al. 1990] developed point of interest movement where the user selects a point on the geometry and begin to move towards it; their rate of travel decreasing as they approach the object. This concept was extended by [Ware and Fleet 1997] who varied the velocity of a user flying through an environment interactively, slowing their speed as they neared geometry, and speeding up as they became more distant. Their approach was based on sampling a subset (15 rows) of the depth buffer in the viewing direction.

3 Image-Based Environment Representation

Many systems rely on the CPU to perform geometric processing as the viewer navigates about an environment, but this approach can be prohibitive for sufficiently complex scenes. Instead, we exploit the speed of modern graphics hardware and rasterize the environment geometry into a compact image-based representation. As we will show, this representation is appropriate for all environment queries that our algorithms require. In our representation, each image-space position maps to a 3D vector that points out into the scene from the viewer's position. The value at each image-space position in the map defines the distance to environment geometry in that direction.

Our approach renders environment geometry using a shader that calculates distance to the viewer for each fragment. Each fragment is a unique position in the image-based representation. Distances are normalized by the current frustum near and far plane distance. The distance d is used as the depth component in the image for each fragment. To be shown visually, we also set the red colour component to $1.0 - d$ (so "redness" corresponds to closeness to the viewer).

We call our representation the *cubemap* as the scene is rendered with a camera that faces each of six canonical directions whose projection planes correspond to the faces of a cube. We use a perspective camera with 90 degree field of view positioned at the viewer's eye. The combined frusta of the cameras completely cover the space until their clipping limits meet. The worldspace position of the pixel $x, y \in [-1, +1]$ in one of the i cubemap sub-images can be recovered using the following:

$$pos(x, y, i) = dist(x, y, i) \cdot norm\,(front(i) \mid right(i)x \mid up(i)y)\,, \quad (1)$$

where $front, right, up$ are unit vectors whose directions correspond to the projection for face i. We note for clarity that only the camera's worldspace position (and not its orientation) is needed to update the cubemap; our implementation uses fixed vectors for $front, right, up$ such that the faces unwrap as a cube, whose front face is always oriented towards the positive x-axis (see Figure 2).

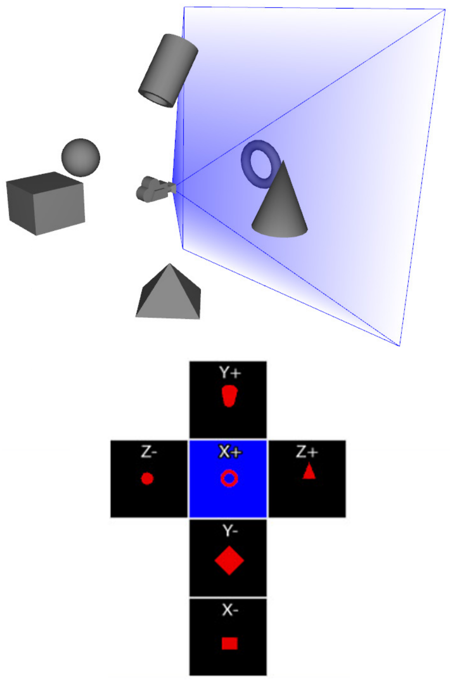


Figure 2: Illustration of scene geometry depth information rendered to worldspace axis-aligned faces of the cubemap. The face pointing toward the positive x-axis is highlighted in blue.

The cubemap can be updated at every frame, or simply when there is camera movement. The cubemap does not require precomputation, or maintenance of any additional data structures. This is an important property for environments that are dynamic or are in the process of being authored. Since our applications of the cubemap either use samples to estimate a single statistic or average multiple samples, we do not rely on having high-resolution cubemap face textures, or use highly-detailed geometry.

In general, the sampling resolution of an object at a distance d from the camera is $2d/cubeMapResolution$. In our implementation, we found a 64x64 sample size for each cube face gave us an effective resolution of 10 centimeters for objects 3.2 meters away, which we found to be sufficient for typical environments. We next show three particular applications of this environment representation.

3.1 Scale Detection

In navigation tools that use some notion of absolute speed (such as the walk tool), it is necessary that speed be related to environment size. In some applications, the scale of the environment is generally uniform and known, so speed parameters can be hand-tuned to match that. However, it may not always be known (such as imported geometry with incomplete or incorrect units), or may vary greatly across the environment. In such situations, we can use the cubemap's distance values to estimate the scale of the local environment the viewer is in. These estimates can be used to modulate speed parameters.

In our implementation, we have found simply using the minimum distance from the cubemap to estimate scale produces acceptable and consistent navigation behaviour for a multitude of environments and scales, especially when there is little or no prior knowledge of the environment geometry.

We have also experimented with using the mean of all distance values in the cubemap as a scale estimate. Unfortunately, it is necessary to use a robust estimate, since the distance samples are likely to contain a significant number of structured outliers. For example, a room containing open doors or windows will contain distance samples from the environment outside the room. We implemented Tukey's one-step biweight algorithm [Hoaglin et al. 2000] to find an average distance value that is less affected by these structured outliers. We have also explored using Gaussian mixture models to cluster distance samples into different categories. However, we have found that in practice these robust averaging approaches produce less predictable navigation behaviour.

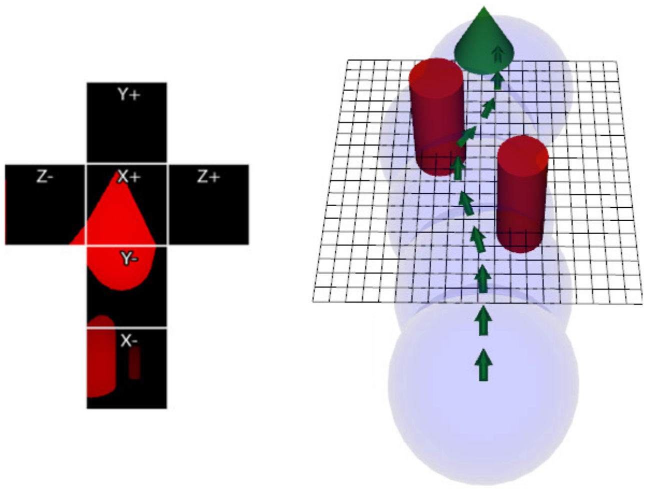


Figure 3: Camera path is automatically modified to avoid collision with nearby geometry within the collision radius.

3.2 Smooth Collision Resolution

To prevent the viewer from colliding with the environment and avoid the overhead of typical geometric collision detection methods, we again take advantage of the cubemap (see Figure 3). For every distance sample, we apply a soft collision penalty force if the distance is within a threshold δ. When δ is constant across samples, the viewer's collision boundary is then a sphere with radius δ. The net penalty force is then:

$$\frac{1}{N_x N_y 6} \sum_{x,y,i} w\,(dist(x, y, i)) \cdot norm\,(pos(x, y, i) - eye). \quad (2)$$

where $w(dist) = sign(dist)e^{\frac{-min(|dist|-\delta, 0)^2}{2\sigma^2}}$ is a soft penalty function and σ is a softness parameter. Bound radius δ can be modulated by scale estimate as described in the previous subsection, but there may still be cases when it is too large and may prevent the viewer from navigating through small holes. To resolve this in click-to-fly navigation mode, we slowly shrink the bound radius when the viewer is far from their destination, but is not making any progress towards it (i.e. is stuck due to collision).

3.3 Dynamic Viewing Frustum

Due to the limited numerical precision of the depth buffer (32-bit, or possibly lower) using static values for the near and far plane distances of the viewing frustum will cause undesirable clipping at specific scales. To resolve this, our approach is to dynamically modify the near and far plane distances of the viewing frustum using our cubemap approach.

We can rely on the maximum and minimum distances obtained by our image-based representation to select optimal distances for the near and far planes of the viewing frustum. The intuition behind this approach is that we wish to keep the distance of visible scene geometry within a threshold between the near and far planes. As the camera moves around the scene, the near and far planes should be dynamically updated to contain the changing minimum and maximum distances of scene geometry. In addition, if the current normalized minimum distance from the cubemap is 1.0 (indicating no geometry was rendered to the cubemap), then the near/far plane distances should be decreased/increased to "search" for the geometry in the scene. In our system, we use the (non-normalized) minimum

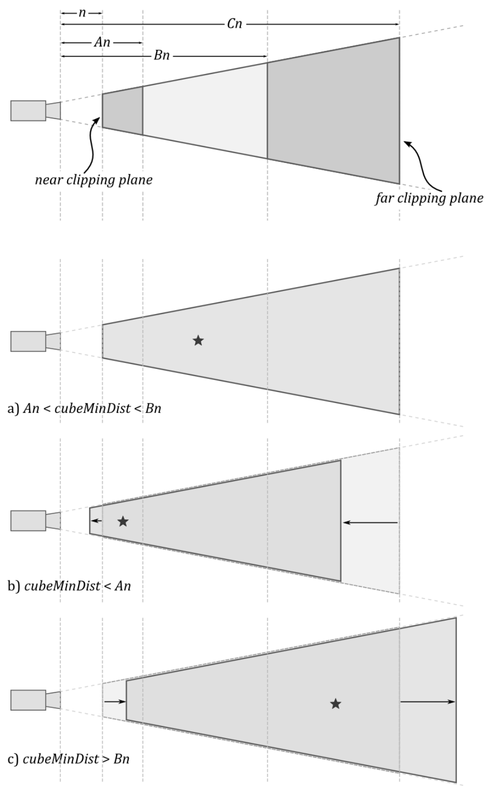


Figure 4: Our frustum technique dynamically changes the near and far plane clipping bounds. (a) The frustum does not change. (b) The frustum is shortened. (c) The frustum is extended.

distance in the cubemap, $cubeMinDist$, to determine if changes in the near and far planes are necessary (see Figure 4). We update the near plane distance n at each frame using the following assignment:

$$n = \begin{cases} \alpha n & \text{if } cubeMinDist < An \\ \beta n & \text{if } cubeMinDist > Bn \\ n & \text{otherwise} \end{cases} . \quad (3)$$

In our implementation, we found the following variable assignments produced satisfactory results: $\alpha = 0.75, \beta = 1.5, A = 2, B = 10, C = 100$. We clamp n and Cn (the near and far plane distances) for unreasonably small or large values.

In the case of navigating the Earth, we have prior knowledge of the structure of the scene (a sphere with much smaller objects embedded on its surface), and we could have, for example, used the length of the tangent line from the camera position to the globe to define the far plane distance. The problem with such an ad-hoc assignment is that it does not work at all scales, indeed once sufficiently close to the Earth's surface the result is unacceptable. By comparison, our cubemap-dependent approach to determine near and far plane distances is robust, working both far from the Earth and up-close. It requires neither ad-hoc assignments or prior knowledge of the geometry, making it a general, multiscale approach to managing the viewing frustum.

3.4 Proxy Objects

Our system can render various types of primitives (sphere, cylinder, box or capsule) as a geometric approximation of a more complicated scene object. We call these objects *proxy objects*. When loading each object into the scene, the dimensions of the proxy object are automatically calculated using the object's vertex data. We list some of the benefits of using proxy objects in our system:

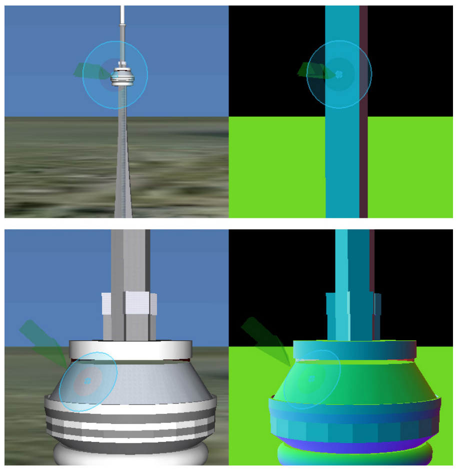


Figure 5: Top row: At a significant distance from the object (left), the low-resolution proxy object (right) is used for cubemap depth rendering and for 3D cursor orientation and position. Bottom row: At close distances, the original object geometry is used.

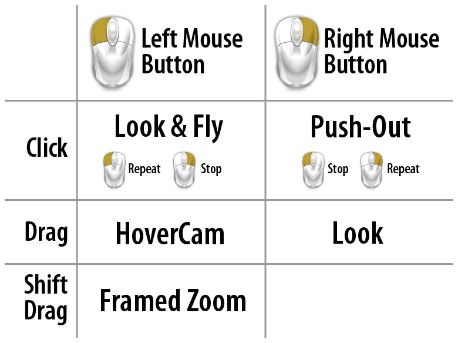
Performance: Updating the cubemap is a GPU-intensive operation. Except when very close to a particular scene object, using a low-detail geometric approximation will result in the cubemap yielding similar if not equivalent scale detection/collision resolution results, while taking less time to update.

Analytic Computation: Knowing the type of primitive, its dimensions, and local coordinate frame, we can efficiently evaluate a precise distance from the camera to the proxy object, as well as nearest point on the proxy geometry. This can enable specific navigation behaviours with a proxy object without the need to update the cubemap. Coarse-level collision detection can also be performed.

Smooth Normal for Distant Objects: Our interface incorporates a 3D cursor whose appearance is consistent between scales. The cursor sits on the surface of scene geometry, orienting itself on the plane perpendicular to the surface normal. A somewhat distant scene object with variation in the normal direction will cause the cursor to erratically orient in different directions as it is moved along the object. By using the proxy object, the cursor orientation remains consistent (see Figure 5).

4 Multiscale 3D Navigation

In the environment examples we provide here, there are several fairly distinct scales in which the camera can operate. Specifically: orbit-level, city-level, neighbourhood-level, and building interior-level. We present the implementation-specific challenges, in producing a system that works at these scales later, in the Discussion section. In the following subsections, we detail the components of our system that the user interacts with and utilizes to navigate through our multiscale scene.



	Left Mouse Button	Right Mouse Button
Click	Look & Fly Repeat Stop	Push-Out Stop Repeat
Drag	HoverCam	Look
Shift Drag	Framed Zoom	

Figure 6: Interaction scheme for multiscale 3D navigation.

We developed a mouse-driven interaction scheme to drive Multiscale 3D Navigation. A left-mouse-buttom (LMB) click initiates a fly towards the intersection with geometry under the mouse cursor. Subsequent LMB clicks retarget the fly. While in flying mode, moving the mouse around the viewport allows the user to look around the scene while still following the original fly trajectory. To break flying mode, a user clicks the right-mouse-button (RMB). When stationary, a LMB drag initiates HoverCam mode, allowing the user to perform exocentric inspection of geometry in the scene. Conversely, a RMB drag performs an egocentric look operation. A RMB click initiates push-out mode, where the collision bubbles around objects expand, forcing the camera out and away from geometry in the scene. A single LMB click cancels push-out mode. Finally, holding shift while dragging with the LMB allows the user to perform a framed zoom-in on scene geometry (see Figure 6).

4.1 Look-and-Fly

The *look-and-fly* navigation method allows the user to travel through the scene at a scale-dependent rate, while avoiding collision with obstacles. The user is able to freely change the viewing direction of the camera with the mouse during flight. The centre of the screen acts as a "deadzone", a small area in which if the mouse is present the viewing direction will not change. When the mouse leaves the deadzone, the cursor's direction from the centre of the screen defines the angle the camera's view direction should rotate. The distance of the cursor from the deadzone controls the rate of rotation.

The user initiates look-and-fly by left clicking. The destination point of the flight is set to the current 3D cursor position p_{Cursor}, which remains fixed. During flight, we then use the following function to displace the camera position p_{Cam} over a time interval $t_\triangle$ between frame updates:

$$p_{Cam} = p_{Cam} + (p_{Cursor} - p_{Cam}) \cdot t_\triangle \cdot \frac{cubeDistMin}{2}, \quad (4)$$

where $cubeDistMin$ is the (non-normalized) distance of the closest point in the scene, provided by the cubemap. Scaling the speed of flying motion relative to the distance of nearby geometry provides a consistent sense of speed when navigating between scales.

While flying, the cubemap is also used to perform collision detection with the scene. Because of this, the path of flight will not always be a straight line from start to finish - the camera will avoid obstacles that enter its local vicinity. In fact, because of collision detection, the camera will never reach the destination point given by p_{Cursor}, instead being repelled once geometry enters the collision avoidance radius δ. The collision radius δ is assigned a new value when flying is initiated, by using the distance from start to finish (we use $\delta = \frac{\|p_{Cursor} - p_{Cam}\|}{4}$).

4.2 Push-Out

As a complement to look-and-fly, we implement the push-out technique to perform the opposite behaviour. Moving the camera to a position where it has perspective of a larger scale has a trivial solution in the case of Earth navigation - simply move the camera position further from the Earth's centre. However in our case, the ability to navigate within structures is a key component of our system. The simple approach of translating the camera away from the Earth will potentially penetrate the roof, resulting in a camera path that is unnatural, and further does not provide a behaviour that is consistent between scales.

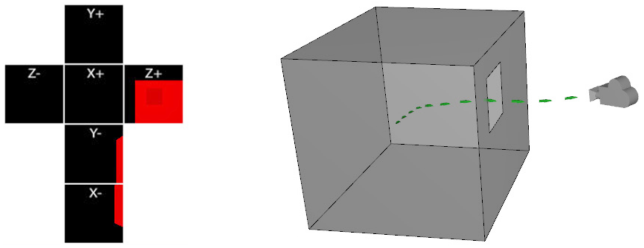


Figure 7: By using push-out within an enclosure, the camera moves along a curved path that takes it outside through an open hole.

Instead, our system uses the nearest exit point when the camera is inside a structure. We exploit the collision resolution offered by our image-based representation to achieve this. Specifically, when the user right clicks the mouse to initiate *push-out*, we dilate the collision avoidance radius δ. Specifically, we assign it $\delta = 3.0 \cdot cubeMinDist$. In the case of the camera being enclosed by scene geometry, this gets the camera flying toward an exit point (such as a doorway or window), if present. In the case of the camera being exterior to geometry, as is the case when orbiting the Earth, or viewing a city from above, this moves the camera in a direction generally away from the geometry. In both of these cases, the camera moves in a natural way to take the user between scales, using the same general approach. If the user wants more rapid movement to a coarser scale, pressing the right mouse button additional times multiplies the radius δ by a constant factor (we use $\delta = 3.0 \cdot \delta$) once the behaviour is initiated.

This technique is not robust in that it will not find a complex path away from geometry (e.g., within a complex maze), however it is reliable in producing simple motion paths where an exit point is observable from the camera's current position.

4.3 An Improved Hovercam

The original implementation of HoverCam navigation mode [Khan et al. 2005] required calculation of the point on the environment closest to the viewer along with a surface normal at that point. This calculation was costly and relied on pre-computation of sphere tree data structures. It is possible to use our cubemap to find the closest point without explicitly testing the geometry, which is sufficent for re-implementation of the HoverCam behaviour. Since the closest point is guaranteed to lie on the surface visible to the viewer, it is the worldspace position of the pixel in the cubemap with the smallest distance value. The HoverCam approach is further simplified in that the process of searching for a new closest point in the direction of movement (to avoid both object collision, and unnatural, hook-shaped camera paths) is handled automatically by the cubemap, which searches all directions into the scene at once.

The original implementation also suffered from temporal coherence issues when sampling the closest point from noisy geometry. To ensure smoothness of HoverCam navigation, G^1 surfaces are desirable. Since geometry-based smoothing is costly, we use an image-based smoothing technique to approximate local smoothing of geometry. We have implemented a smoothing algorithm that uses a box-filter that specifically targets depth values in the cubemap whose normalized values are less than 1. Other algorithms for smoothing depth values in the cubemap exist (such as gaussian, or a bilateral filter for edge preservation) which we have not yet evaluated.

Our system allows interaction with any of multiple independent scene objects. The user performs HoverCam behaviour by dragging the left mouse button on a particular scene object. Just the scene object selected is rendered into the cubemap in order to find its closest point (as it may not provide the true closest point in the scene to the camera). This approach avoids the issue of the camera switching between viewing the object of interest, and another separate object in the scene which may be closer. For example, consider examining the base of a tower (the object of interest), it would be undesirable to have HoverCam suddenly switch the view to the Earth immediately below the camera. While we render only the object of interest in the cubemap to obtain its closest point specifically, we still peform collision detection with all scene objects by rendering each into the cubemap.

4.3.1 Dynamic Up Vector

Up until this point we have not discussed how we update the up vector for the camera, which is an important parameter. Also, in the case of HoverCam behaviour, different up vector models vary the general behaviour of the HoverCam as a navigational tool. In the system we present here, we use the “global” up vector model, on a per-scene-object basis.

In our system, each independent scene object has its own unique origin and coordinate frame. We assume the scene object upon importation is oriented such that its vertical direction points along the positive y-axis. The influence of a scene object's up vector is based on the object's proximity to the camera.

In the case of the Earth, we would like that the camera always point to the North Pole when the camera is at an orbit-level scale in the scene, but normal to the surface when up close. To achieve this, our Earth object is a subclass of a generic scene object class. We overload the up vector method to specify a more-complex up vector assignment. The up vector points towards the North Pole but as the camera moves closer to the surface the up vector direction gradually changes to point normal to the Earth's surface.

4.4 Framed Zooming

While holding shift, a left-click drag sets up a *framed zoom*. The framed zoom allows the user to specify a particular region of the scene they would like to see from a closer viewpoint. The extent in which the camera zooms in will be determined by the framing circle, whose radius is determined by projecting a ray from the current mouse position through the plane defined by the position and normal of the 3D cursor. The framing circle appears as a stencil, darkening areas of the scene that will not be visible when rendered at the destination (see Figure 8). The transparent green arrow provides a visual affordance of the destination position and orientation of the camera when the left mouse button is released.

If p_{Cursor} and n_{Cursor} are the position and normal of the 3D cursor, the framing circle has radius r, and the horizontal and vertical FOVs of the camera are θ_H and θ_V, the new camera position p_{Cam} is given by

$$p_{Cam} = p_{Cursor} + n_{Cursor} \frac{r}{\sqrt{2}\min(\tan\theta_H, \tan\theta_V)}, \quad (5)$$

and the camera will be oriented to look at p_{Cursor}.

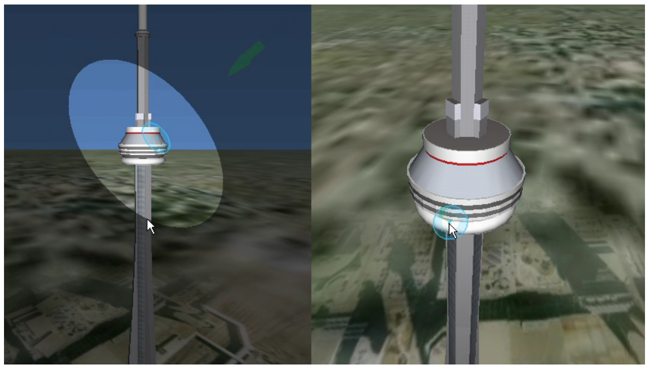


Figure 8: (Left) Holding shift, the user clicks and drags to frame a region. (Right) Upon release, the camera animates to a new position whose view tightly contains the region.

5 Discussion

Implementing a system that works at a dynamic range of scales - from thousands of kilometres down to centimetres - causes other specific problems to surface which we must address.

One problem facing systems which allow navigation between such scales results from the lack of floating point precision for vertex positions far from the origin of the scene. Our implementation originally used the Earth's centre as the origin. Objects on the Earth's surface, far from this origin appeared to “jitter” when viewed close-up. The non-uniform distribution of floating point values, whose relative spacing is further exaggerated when zoomed in, produces the observed effect. Note that the naive approach of keeping the camera positioned at the origin and translating the scene geometry accordingly also will not prevent this effect, the culprit being catastrophic cancellation in this case.

As our system is multiscale and relies on image-based techniques for navigation, we had to address this issue. Our solution is to define a dynamic *global origin* that all scene objects are rendered relative to. The global origin of the scene moves according to the smaller scene objects the camera is close to (specifically, the geometric structures littering the Earth's surface). Display lists for all scene objects are regenerated relative to the global origin after it is translated.

6 Conclusions

Our goal of providing a seamless navigation system for highly multiscale 3D datasets was met. The key technique enabling this advanced interaction was the cubemap. This image-based environment representation was used effectively in (a) a robust scale-detection system, (b) a smooth collision resolution heuristic, (c) a dynamic viewing frustum critical to the success of the overall goal, and (d) a proxy object rendering to facilitate object-centric navigation.

The simplified interactions for the Look-and-fly and Push-Out modes facilitated the transition between many scales within the dataset and this was made possible by the intelligent navigation adjustments implicitly made by the cubemap-based algorithms. In particular, the Push-Out mode was not constrained by a specific input point and so, nicely reflected an object-level scale change with each invocation.

Also, by implementing an image-based HoverCam algorithm, this high-level interaction technique could be provided without an expensive sphere-tree precomputation and this could work on dynamic 3D scenes as well.

Finally, the framed zoom operation, together with 3D cursor feedback provided fine control over navigation through a city model containing many faceted targets. This situation benefitted noticeable from the 3D cursor feedback increasing predictability of the resulting camera motion.

Overall, significant progress was made to create a small group of interactions to perform multiscale 3D navigation.

7 Acknowledgments

Keenan Crane for the excellent island/city/house/jug model, Justin Matejka for help with figures, Ryan Schmidt for code bits and helpful discussion, Tovi Grossman and George Fitzmaurice. We also thank Nicolas Burtnyk for originally suggesting this approach.

References

ABÁSOLO, M. J., AND DELLA, J. M. 2007. Magallanes: 3d navigation for everybody. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, New York, NY, USA, 135–142.

BACIU, G., AND WONG, W. S.-K. 1997. Rendering in object interference detection on conventional graphics workstations. In *PG '97: Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 51.

BACIU, G., WONG, W., AND SUN, H. 1998. Recode: An image-based collision detection algorithm. *Computer Graphics and Applications, Pacific Conference on 0*, 125.

BALAKRISHNAN, R., AND KURTENBACH, G. 1999. Exploring bimanual camera control and object manipulation in 3d graphics interfaces. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, 56–62.

BARES, W. H., AND LESTER, J. C. 1999. Intelligent multi-shot visualization interfaces for dynamic 3d worlds. In *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces*, ACM, New York, NY, USA, 119–126.

BARES, W., MCDERMOTT, S., BOUDREAUX, C., AND THAINIMIT, S. 2000. Virtual 3d camera composition from frame constraints. In *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, ACM, New York, NY, USA, 177–186.

BURTNYK, N., KHAN, A., FITZMAURICE, G., BALAKRISHNAN, R., AND KURTENBACH, G. 2002. Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 101–110.

BURTNYK, N., KHAN, A., FITZMAURICE, G., AND KURTENBACH, G. 2006. Showmotion: camera motion based 3d design review. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 167–174.

COHEN, J. M., HUGHES, J. F., AND ZELEZNIK, R. C. 2000. Harold: a world made of drawings. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 83–90.

DARKEN, R. P., AND SIBERT, J. L. 1993. A toolset for navigation in virtual environments. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 157–165.

DOS SANTOS, C. R., GROS, P., ABEL, P., LOISEL, D., TRICHAUD, N., AND PARIS, J. P. 2000. Metaphor-aware 3d navigation. In *INFOVIS '00: Proceedings of the IEEE Symposium on Information Vizualization 2000*, IEEE Computer Society, Washington, DC, USA, 155.

DRUCKER, S. M., AND ZELTZER, D. 1995. Camdroid: a system for implementing intelligent camera control. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 139–144.

ELMQVIST, N., TUDOREANU, M. E., AND TSIGAS, P. 2008. Evaluating motion constraints for 3d wayfinding in immersive and desktop virtual environments. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, 1769–1778.

FAN, Z., WAN, H., AND GAO, S. 2004. Simple and rapid collision detection using multiple viewing volumes. In *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, ACM, New York, NY, USA, 95–99.

FITZMAURICE, G., KHAN, A., PIEKÉ, R., BUXTON, B., AND KURTENBACH, G. 2003. Tracking menus. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 71–79.

FITZMAURICE, G., MATEJKA, J., MORDATCH, I., KHAN, A., AND KURTENBACH, G. 2008. Safe 3d navigation. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 7–15.

GALYEAN, T. A. 1995. Guided navigation of virtual environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 103–ff.

GLEICHER, M., AND WITKIN, A. 1992. Through-the-lens camera control. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 331–340.

HAIK, E., BARKER, T., SAPSFORD, J., AND TRAINIS, S. 2002. Investigation into effective navigation in desktop virtual interfaces. In *Web3D '02: Proceedings of the seventh international conference on 3D Web technology*, ACM, New York, NY, USA, 59–66.

HANSON, A. J., AND WERNERT, E. A. 1997. Constrained 3d navigation with 2d controllers. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, IEEE Computer Society Press, Los Alamitos, CA, USA, 175–ff.

HANSON, A. J., WERNERT, E. A., AND HUGHES, S. B. 1997. Constrained navigation environments. IEEE Computer Society, Los Alamitos, CA, USA, vol. 0, 95.

HOAGLIN, D., MOSTELLER, F., AND TUKEY, J. 2000. *Understanding Robust and Exploratory Data Analysis*. John Wiley & Sons, New York, NY, USA.

IGARASHI, T., KADOBAYASHI, R., MASE, K., AND TANAKA, H. 1998. Path drawing for 3d walkthrough. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 173–174.

JUL, S., AND FURNAS, G. W. 1998. Critical zones in desert fog: aids to multiscale navigation. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 97–106.

KHAN, A., KOMALO, B., STAM, J., FITZMAURICE, G., AND KURTENBACH, G. 2005. Hovercam: interactive 3d navigation for proximal object inspection. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 73–80.

KHAN, A., MORDATCH, I., FITZMAURICE, G., MATEJKA, J., AND KURTENBACH, G. 2008. Viewcube: a 3d orientation indicator and controller. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 17–25.

KOLB, A., LATTA, L., AND REZK-SALAMA, C. 2004. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, New York, NY, USA, 123–131.

LI, T., AND CHOU, H. 2001. Improving navigation efficiency with artificial force field. In *In Proceedings of 2001 14th IPPR Conference on Computer Vision, Graphics, and Image Processing*.

LI, T.-Y., AND HSU, S.-W. 2004. An intelligent 3d user interface adapting to user control behaviors. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, ACM, New York, NY, USA, 184–190.

LI, T.-Y., AND TING, H.-K. 2000. An intelligent user interface with motion planning for 3d navigation. In *VR '00: Proceedings of the IEEE Virtual Reality 2000 Conference*, IEEE Computer Society, Washington, DC, USA, 177.

MACKINLAY, J. D., CARD, S. K., AND ROBERTSON, G. G. 1990. Rapid controlled movement through a virtual 3d workspace. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 171–176.

MYSZKOWSKI, K., OKUNEV, O., AND KUNII, T. 1995. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer 11(9)*, 497.

ROPINSKI, T., STEINICKE, F., AND HINRICHS, K. 2005. A constrained road-based vr navigation technique for travelling in 3d city models. In *ICAT '05: Proceedings of the 2005 international conference on Augmented tele-existence*, ACM, New York, NY, USA, 228–235.

SALOMON, B., GARBER, M., LIN, M. C., AND MANOCHA, D. 2003. Interactive navigation in complex environments using path planning. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 41–50.

STEED, A. 1997. Efficient navigation around complex virtual environments. In *VRST '97: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 173–180.

TAN, D. S., ROBERTSON, G. G., AND CZERWINSKI, M. 2001. Exploring 3d navigation: combining speed-coupled flying with orbiting. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, 418–425.

VASSILEV, T., SPANLANG, B., AND CHRYSANTHOU, Y. 2001. Fast cloth animation on walking avatars. 260–267.

WARE, C., AND FLEET, D. 1997. Context sensitive flying interface. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 127–ff.

WARE, C., AND OSBORNE, S. 1990. Exploration and virtual camera control in virtual three dimensional environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 175–183.

WINTER, M., AND STAMMINGER, M. Depth-buffer based navigation.

XIAO, D., AND HUBBOLD, R. 1998. Navigation guided by artificial force fields. In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 179–186.