Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2009)
E. Grinspun and J. Hodgins (Editors)

# Visibility Transition Planning for Dynamic Camera Control

Thomas Oskam[1] Robert W. Sumner[2] Nils Thuerey[1] Markus Gross[1,2]

[1]Computer Graphics Lab, ETH Zurich
[2]Disney Research, Zurich

**Abstract**

*We present a real-time camera control system that uses a global planning algorithm to compute large, occlusion free camera paths through complex environments. The algorithm incorporates the visibility of a focus point into the search strategy, so that a path is chosen along which the focus target will be in view. The efficiency of our algorithm comes from a visibility-aware roadmap data structure that permits the precomputation of a coarse representation of all collision-free paths through an environment, together with an estimate of the pair-wise visibility between all portions of the scene. Our runtime system executes a path planning algorithm using the precomputed roadmap values to find a coarse path, and then refines the path using a sequence of occlusion maps computed on-the-fly. An iterative smoothing algorithm, together with a physically-based camera model, ensures that the path followed by the camera is smooth in both space and time. Our global planning strategy on the visibility-aware roadmap enables large-scale camera transitions as well as a local third-person camera module that follows a player and avoids obstructed viewpoints. The data structure itself adapts at run-time to dynamic occluders that move in an environment. We demonstrate these capabilities in several realistic game environments.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation, K.8 [Personal Computing]: Games

## 1. Introduction

In any virtual environment, computer game, or other interactive application, natural camera motion is crucial for a positive user experience. Successful navigation through content strongly depends on appropriate camera movement. When considering automatic camera control, four critical criteria include:

**Real-time.** Static or precomputed viewpoints are unsuitable since the user's actions are not known a priori. Instead, the camera must adapt in real-time.

**Collision-free.** As the camera moves through a scene, it must not collide with objects in the environment as this lends an unrealistic feel and lessens the sense of immersion.

**Smooth.** Teleporting the viewpoint from one place to another may disorient the user since the continuity of the view is broken.

**Visible.** Ultimately, the camera's goal is to look at something. Thus, visibility is of utmost importance: the player or other focus target must be kept in view and unobstructed.

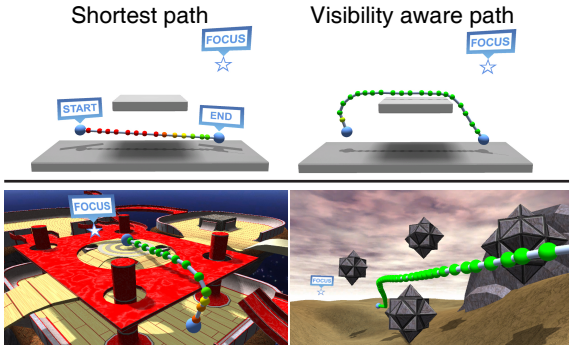


**Figure 1:** *Our system for visibility transition planning computes long camera transitions in real-time that keep a focus point in view. A comparison between a shortest path and our visibility-aware path is shown on the top. Our method can perform complex camera movements that follow a fast moving player, and adapt to dynamic environments (bottom).*


SCA 2009, New Orleans, LA, August 1–2, 2009.
© 2009 ACM 978-1-60558-610-6/09/0008 $10.00

Classical third-person camera models that follow a player deal with these criteria using algorithms that are inherently local in nature. Small adjustments in position and orientation are made based on objects in the camera's immediate vicinity. Such local camera control is effective in some situations, but if the character dashes quickly behind corners, the locality assumptions are broken and the camera may be forced to pass through an object or teleport to an unobstructed view. Furthermore, local camera models are not designed to perform large-scale transitions between viewpoints.

Our research alleviates the aforementioned problems by computing smooth, collision-free transitions between arbitrary start and end points, while emphasizing the visibility of a third focus point. We refer to this problem as *visibility transition planning*, since collision-free movement through an arbitrary three-dimensional environment is naturally solved via path planning. However, a visibility transition may deviate significantly from the shortest path in order to ensure that the focus point is visible, as shown in Fig. 1. A typical application might be the switch from a third-person view of an avatar in a virtual city environment to an overhead view while ensuring a clear view of the avatar throughout the camera motion. Maintaining unbroken focus helps the user to better understand position and context. Although large-scale transitions and local camera control may seem like disparate problems, we propose that the two are closely connected. A local camera control module that deals with visibility in a sophisticated way must move the camera from one point in space to another in order to maintain or regain visibility (Fig. 2). For certain configurations, this movement can only be resolved with global knowledge of the scene.

Visibility computations in arbitrary three-dimensional environments are notoriously complicated and time-consuming. The difficulty of this problem is exacerbated by several factors. The global nature of the transition planning problem means that a potentially huge number of visibility evaluations must be considered. Neither the start point, nor the end point, nor the focus point are known in advance, making precomputation a non-trivial task. Finally, the algorithm has strict, real-time requirements, with only milliseconds available for all computations.

We present an algorithm for visibility transition planning that can compute large, collision-free camera transitions in real-time in dynamic environments. This achievement rests on several key insights. We develop a visibility-aware roadmap data structure that allows the precomputation of a coarse representation of all collision-free paths through an environment, together with an estimate of the pair-wise visibility between all portions of the environment. Once the start, end, and focus point have been specified, our runtime system executes a path planning algorithm using the precomputed roadmap values to find a coarse path that is optimal in terms of visibility up to the resolution of the roadmap. Next, the path is refined by computing a sequence of GPU-assisted occlusion maps along the coarse path. The same path-planning code is executed with these occlusion maps to enhance visibility on a fine scale. An iterative smoothing algorithm together with a physically-based camera model ensure that the path followed by the camera is smooth in both space and time. All run-time computation is output sensitive, so that the required time depends on the final path length. The visibility-aware roadmap data structure adapts dynamically to occluders that move in an environment, supporting opening and closing doors, falling boulders, and other occlusion situations in real-time.

In terms of impact, our contribution includes the first camera control system that can generate large, collision-free camera transitions customized for the visibility of a focus point in real-time. This functionality opens up new opportunities for game designers, such as dynamic target switching between multiple characters and fly-throughs that demonstrate a suggested path through an environment from arbitrary start and end points. Additionally, we present a third-person camera routine that optimizes for visibility yet never passes through scene geometry. Existing local camera models cannot make such claims, since some geometric configurations can only be resolved using global knowledge.

Technically, we contribute the visibility-aware roadmap data structure as well as a strategy to successfully divide calculations between precomputation, run-time CPU calculation, and GPU computation in order to perform visibility transition planning in real-time. Precomputation requires only seconds, run-time calculations only milliseconds, and data structures only a few megabytes. We demonstrate how to update this data structure dynamically to support moving occluders. Finally, we show how the roadmap can be used to achieve proactive camera movement that actively seeks a position to prevent the player from escaping visibility. In developing these algorithm, a shortcoming in any single area can eclipse the benefit of the entire system, yielding it useless. Thus, systemic and practical issues are extremely important, and their solution non-obvious. The realization of our camera-control system rests on many components that all work together interdependently without ever sacrificing high-performance.

## 2. Background

A thorough overview of the state-of-the-art in camera control for computer graphics is presented by Christie, Oliver, and Normand [CON08], and we review only a small portion of this work here. Many classical third-person camera models examine the local vicinity in order to resolve occlusions. A common camera model used in computer games casts a ray from the player to the camera and teleports the camera to closest intersection in order to maintain visibility, leading to a noticeable jump in view. An alternate scheme makes occluding geometry transparent, which avoids camera jumps but detracts from the environment's

realism. Halper, Halbing, and Strothotte [HHS01] present a more sophisticated local camera model that resolves occlusions using point-based visibility with z-buffer hardware back projection. While this method is able to resolve certain local situations, it cannot find large transitions around multiple obstacles. Li and Cheng [LC08] focus on finding an unoccluded camera position, but sometimes teleport when the camera cannot be readily moved to this vantage. These algorithms handle some situations well, but their local nature leads to inherent limitations. The camera may not adequately follow a fast moving character or resolve a complicated visibility situation, resulting in discontinuous jumps or passing through scene geometry (Fig. 2). In contrast, our algorithm is global in nature which permits both large-scale camera transitions as well as a third-person camera that follows an avatar in a natural fashion without discontinuous jumps or collisions with objects in the scene.

More high-level approaches to camera control focus on virtual cinematography, in which cinematographic knowledge is incorporated into the choice of camera position and scene composition. For example, Bares and his colleagues [BGL98] present a constraint-based camera planner for shot composition that models different cinematic styles, and He, Cohen, and Salesin [HCS96] encode film making heuristics into a hierarchical finite state machine that controls camera selection and placement. Our work complements these approaches by offering a sophisticated system for camera transitions. Cinematographic rules may provide a sequence of shot placements and focus targets, while our system controls the actual camera motion between such shots. Drucker and Zeltzer [DZ94] present an alternative for advanced camera control with visibility based path optimization. However, their approach focuses on the creation of offline animations and is not suitable for real-time applications.

A key aspect of our camera system is the focus on visibility during large camera transitions. Visibility problems are fundamental to computer graphics. As shown by surveys on the topic [COCSD03, Bit02, Dur00], many visibility algorithms strive to identify which objects, lights, or other portions of a scene are visible from a given vantage. Runtime visibility calculations can be accelerated via precomputation, in which a visibility relationship between "viewcells" in space and scene objects is established (cf. [Lai05]). Our algorithm relies heavily on precomputation for real-time performance. In our setting, however, the focus target is not an object within the scene but rather a point that can be placed anywhere within the ambient space. Our visibility-aware roadmap (Section 3) extends the idea of precomputed visibility with a data structure that estimates the visibility of every region of space with respect to every other region of space.

The problem of visibility transition planning is related to the topic of motion planning in the robotics literature where optimal paths are found for robot navigation [LaV06, MS07].

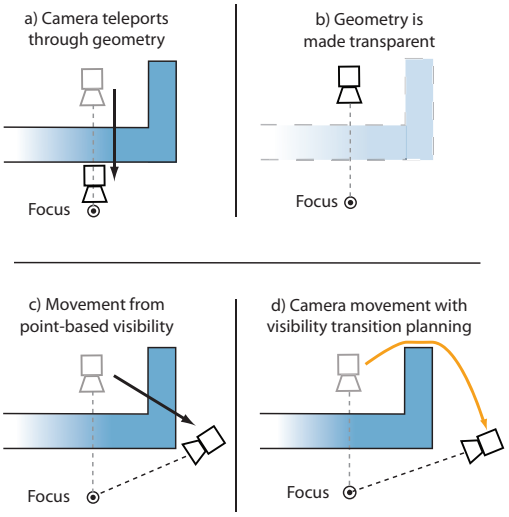


**Figure 2:** *This simple example illustrates the different behaviors of commonly used methods for camera control. (a) The raycast can result in sudden jumps. (b) The obstructing geometry can be made transparent. (c) With point-based visibility [HHS01], the camera may pass through geometry. (d) Our method will avoid obstacles correctly.*

Many motion planning algorithms employ a roadmap construction in which the free configuration space of the robot is mapped to a graph data structure, reducing the planning problem to a graph search [LaV06]. Some algorithms incorporate a notion of visibility with a sparse set of guard nodes whose visibility region can be defined by unobstructed straight lines in configuration space [VM05] or the ability of a local planner to navigate without intersection [SLN00]. The related problem of target tracking strives to compute the motion of a robot observer in order to maintain visibility of a moving target. Sophisticated algorithms address this problem in planar environments with polygonal obstacles (e.g., [MCSBH04, BLAJH04, BLAJH06]). However, direct extension of this work to full 3D motion is non-trivial, partly because visibility relationships are significantly more complex [BAJH07]. Other work on 3D tracking does not deal with occlusions [VSK*02], utilizes only a robot's visual sensors rather than global scene information [BAJH07], or presents interesting theoretical results without demonstrating a system that matches the strict efficiency demands of games [MCMS*07, Laz01]. More generally, camera control for real-time graphics has a different set of requirements than robot navigation. A camera control system must operate in complex, 3D environments. Visibility is critically important, and computed paths should always take visibility into consideration. The start, end, and focus point may move continuously. The system must be extremely efficient, with only milliseconds to compute a camera transition and react to scene updates. While the robotics literature has explored

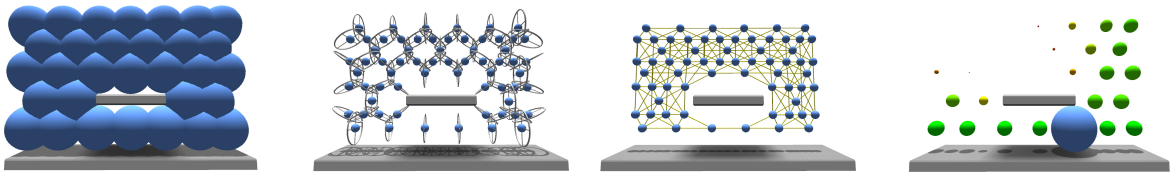


**Figure 3:** *Overview of the creation algorithm for the roadmap. Based on an initial geometry of the environment, we first compute a spatial discretization. A graph is then built from the overlap regions. Finally, for each pair of spheres, a visibility probability is computed with a Monte-Carlo raytracing algorithm.*

navigation in great detail, exiting work does not meet the requirements of a high-performance game camera.

Our work is motivated by the navigation system of Salomon and colleagues [SGLM03]. In their approach, a roadmap is created in which nodes represent an avatar's position, and edges connect nodes between which a local planner can successfully navigate. Niederberger, Radovic, and Gross [NRG04] present a navigation system using a shortest-path search on a triangulated height-field terrain. We build upon these ideas in several ways in order to develop an algorithm that is appropriate for camera control. First, we focus on the full ambient space, rather than just walkable surfaces, and incorporate the notion of dense visibility into the roadmap computation, yielding our visibility-aware roadmap for complex 3D environments. Rather than a minimal number of guard node visibility points, we favor a dense visibility roadmap in which each node corresponds to a local volume of space that overlaps with the volume of adjacent nodes. Movement within and between these volume bounds is guaranteed to be collision free, and an estimate of the percentage of visibility of all other nodes within a falloff distance is always know. Our runtime planning algorithm uses the precomputed visibility values to find a coarse, global path, and a refinement method makes fine-scale adjustments to improve the path shape and incorporate sub-sphere visibility information.

## 3. Visibility-Aware Roadmap

The ultimate goal of our visibility transition planning algorithm is to compute long, collision-free camera transitions through a complex environment, guided by the visibility of a focus point. Since the start, end, and focus points are specified only at runtime, they can be located anywhere in the scene, and may even change continuously. The planning algorithm may explore arbitrary parts of the environment's free space while continually querying the visibility of the focus point in order to compute the best camera transition. To meet the strict real-time constraints of games, training simulations, and other interactive environments, our system requires a data structure that makes these run-time queries as fast as possible.

Motivated by these requirements, we have developed a data-structure based on spheres and portals, which we refer to as a visibility-aware roadmap (Fig. 3). Although we explored other possible spacial data structures such as BSP-trees, KD-trees, and octrees, [PH04], the simplicity, both algorithmically and geometrically, of a spherical division proved to be most efficient. The entire free space of an environment is tessellated with overlapping spheres. A visibility probability value is computed between every pair of spheres and stored within the data structure. Portals are defined by the circle of overlap between any two spheres. The roadmap is a graph structure derived from the spheres and portals by placing a node at the center of each portal and connecting this node to all other portal nodes associated with either of the two overlapping spheres. By traveling entirely within the network of spheres (transitioning from sphere to sphere via the portals), the camera is guaranteed never to collide with scene geometry. As soon as a focus point is fixed, an estimate of the visibility of the focus point from any query point within the scene is known immediately by looking up the precomputed visibility probability between the query point's sphere and the focus point's sphere.

Unlike existing sphere tree representations [Bra04], our system approximates the ambient space with a flat hierarchy of overlapping spheres using an iterative sphere placement algorithm. First, the scene geometry is embedded within a three-dimensional grid with spacing $\Delta x$. Any grid cells that intersect scene geometry are marked as occupied. A candidate sphere of maximal size is constructed at each unoccupied grid cell. To favor uniformity in sphere and portal size, radii are restricted to the range $[\Delta x/3, 3\Delta x]$ and any candidate with a portal area less than $\Delta x/4$ is discarded. A seed sphere is selected at random. Then, in each step of the iteration, the algorithm selects from the candidates the sphere that maximally overlaps the previously selected spheres, creating the largest portals. Grid cells whose center lie within the new sphere are marked as occupied and the corresponding candidates are deleted. The process repeats until no candidate

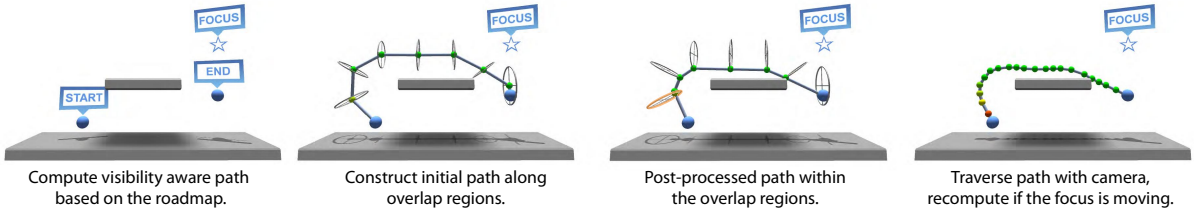


**Figure 4:** *The different steps to compute a visibility aware path based on the roadmap.*

spheres remain. The size of $\Delta x$ depends on the size of features (e.g., doorways, tunnels, etc.) within the environment and should be chosen to be small enough so that all desired features are resolved by the grid. Although not strictly hierarchical, a finer grid (smaller $\Delta x$) can first be used in areas with smaller features (e.g., the inside of a house) followed by a coarser grid on the surrounding scene (e.g., the streets of a village). Although randomized sampling works well for motion planning [YL02], we found that deterministic sphere placement makes it particularly easy for game designers to select appropriate parameter values. However, our method could be adapted to include random sampling of sphere centers, either on the whole domain or, in order to maintain control over the refinement levels, within individual grid cells.

A final precomputation step estimates a visibility probability between all pairs of spheres using a Monte Carlo approach that selects a random point on the hemisphere of a source sphere $i$ facing a destination sphere $j$. A ray is shot toward a second random point on the opposing hemisphere of sphere $j$. The visibility probability $p_{i,j}$ between spheres $i$ and $j$ is given by the fraction of rays that reach the destination sphere before hitting an obstacle. To limit the amount of computations for very large environments, we take into account a maximal visibility distance that specifies how much of a level are typically in view. We only pre-compute the visibilities for spheres that are not further away from each other than the maximal visibility distance.

## 4. Visibility Transition Planning

Visibility transition planning refers to the problem of finding the shortest collision-free camera transition from a start position **s** to an end position **e** such that a focus point **f** is visible as long as possible. Although the extremely complex nature of visibility in arbitrary 3D environments [Laz01] makes a provably optimal solution to this problem intractable in real-time, we present an algorithm to compute an approximate solution within the strict time constraints of games. First, our runtime system executes a visibility-based path-planning algorithm on the precomputed roadmap data structure to find a coarse collision-free path through the scene. Next, a fine-scale refinement is performed by computing a sequence of GPU-assisted occlusion maps in spheres of partial visibility. A final smoothing step shortens the path length by allowing it to hug the sphere portals tightly.

### 4.1. Path Planning on the Roadmap

The first stage of our runtime system computes a coarse path from the sphere containing **s** to the sphere containing **e** along the visibility-aware roadmap. Due to its efficiency, we use the A* search, which is a generalized best-first search algorithm [DP85]. We omit a detailed description of the algorithm since it is ubiquitous in robotics and artificial intelligence and implementations are readily available. A* uses an edge cost function $C$ and finds paths of minimal cost. A path heuristic $H$ provides a lower bound on the remaining cost of a partial path and is used to accelerate the search. The typical shortest-path A* search uses edge length as the cost function and Euclidean distance as the heuristic. We augment the edge length cost with the precomputed visibility probability in order to find paths that maximize the visibility of a focus point. The cost for edge $e_{ij}$ between nodes $i$ and $j$ is given by:

$$C(e_{ij}) = d(i,j) + \alpha d(i,j)\left(1 - v(e_{ij})\right), \qquad (1)$$

where $d(i,j)$ is the length of the edge $e_{ij}$ (the Euclidean distance between nodes $i$ and $j$) and $v(e_{ij})$ is the visibility probability with respect to the focus point. Due to the construction of the roadmap, each edge lies entirely within a given sphere. Thus, we use $v(e_{ij}) = p_{k,f}$, where $p_{k,f}$ is the precomputed visibility probability between the edge's sphere $k$ and the sphere $f$ containing the focus point **f**. This value represents the probability of seeing **f** while traveling along $e_{ij}$. The parameter $\alpha$ determines the relative cost of traveling in regions where **f** is visible versus regions where it is occluded. If $\alpha$ is chosen to be larger than the maximal distance of any path through the roadmap, the algorithm will find the path that travels as quickly as possible into the visibility region. For the heuristic function $H$ of the A* search, we use the euclidean distance between the last point on the path and the target: $H(n) = d(n, \mathbf{e})$.

### 4.2. Path Refinement

The path planning algorithm yields a path $\mathcal{P}$ along the edges of the roadmap, through the roadmap's spheres. Spheres

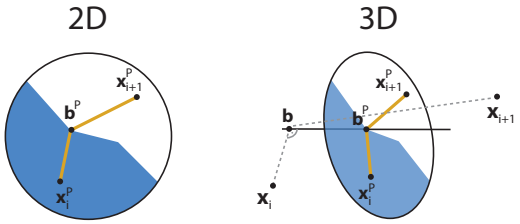


**Figure 5:** *The distance traveled in the occluded region (shown in blue above) is minimized on occlusion maps. The left side shows the 2D plane of the occlusion map, while the right hand side shows how the 3D positions of the points are computed for the actual path.*

with a visibility probability of either 0 or 1 are entirely outside or entirely inside of the visibility region with respect to the focus point, while those with a probability between 0 and 1 are in partial visibility. The focus point may be visible from some positions within a sphere of partial visibility and hidden from other positions. We perform a detailed refinement step in such spheres so that the computed path navigates along positions where the focus point is actually visible. Since the path planning edge weight favors visibility, there will be, whenever possible, few spheres of partial visibility. Thus, the path refinement need only be performed for a small number of spheres.

We can simplify the path refinement for spheres with partial visibility from a three- to a two-dimensional problem, since one dimension is determined by the line of sight to the focus point. Our system builds a detailed representation of the focus point's visibility within the sphere in the form of a 2D occlusion map, which contains per-pixel information indicating whether **f** is visible from a given position within the sphere. The occlusion map is rendered at runtime using a view frustum that is tightly fit around the sphere and originates at the focus point [HHS01]. Our system performs another A* search on this occlusion map. 2D path positions on this map that change visibility are detected and reconstructed in 3D.

Although the occlusion maps provide detailed visibility information, rendering them at runtime for every sphere during path planning would be prohibitively expensive because the A* algorithm may explore hundreds of nodes as it searches for the optimal path. Thus, our system uses the precomputed visibility probability estimates which require only a table lookup during the coarse path planning. Once the coarse path is fixed, only a few spheres will lie in partial visibility due to the nature of the search. The algorithm can afford to compute the more accurate occlusion maps on these few spheres without exceeding the camera's allotted computation budget. Thus, the computation is spent where it is needed most to build the best path.

The start and end points of the 2D path search on the occlusion map are given by projecting the points on $\mathcal{P}$ that enter and exit the sphere onto the map-plane. The entry position $\mathbf{x}_i$ and exit position $\mathbf{x}_{i+1}$ lie on the two overlap circles of the sphere and its predecessor and successor sphere, respectively. The projected positions are denoted by $\mathbf{x}_i^p$ and $\mathbf{x}_{i+1}^p$ in Fig. 5. A path planning, similar to the one described in the previous section is performed on the occlusion map pixels, where each pixel is considered connected to its eight neighbors. The distance $d(i,j)$ and visibility values for $C$ are replaced by functions computed on the occlusion map: $d(i,j)$ is the 2D Euclidean distance, and $v(x_{ik})$ is the average of the two per-pixel visibilities.

Once the occlusion map path has been calculated, the 2D path positions can be reconstructed in 3D. For each pixel, the 3D position can lie anywhere on its projection ray toward the focus point within the sphere. The reconstruction of the start and end points, $\mathbf{x}_i^p$ and $\mathbf{x}_{i+1}^p$, are known from the projections of their 3D positions $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ onto the map. Next, border points $\mathbf{b}_i^p$ are identified on the 2D path. Border points are points on the path where the visibility changes from occluded to visible, or vice versa. For each occluded region on the map, such a border point is minimizing the path length in the occluded area. This implies that the occluded part is a straight line. That means that for the construction of the 3D position of the border point and the path segment in the occlusion region it is enough to project the border point to 3D. Its 3D position $\mathbf{b}_i$ is given by the closest point on its view-line segment to either $\mathbf{x}_i$, or $\mathbf{x}_{i+1}$, as shown in Fig. 5. On the other hand, the portions of the 2D path that are fully visible do not necessarily form a straight line. To avoid errors introduced by approximating visible portions of the path also by a line between $\mathbf{b}_i^p$ and its neighbor, additional points can be iteratively inserted in 2D and reconstructed in 3D as the closest point to the line formed by its 3D predecessor and successor.

### 4.3. Path Post-processing

Although the coarse planning and refinement determines the gross nature of the camera path, the actual path traversed by the camera can be freely moved anywhere within the selected portals without colliding with geometry or changing the visibility score. Our system uses these additional degrees of freedom to smooth the path, creating both shorter and more natural camera movement. The final path positions $\mathbf{x}_i$ are computed using a constrained iterative smoothing algorithm. The corrected position $\mathbf{x}_i'$ of each point $\mathbf{x}_i$ is first found as the intersection of the line from $\mathbf{x}_{i-1}$ to $\mathbf{x}_{i+1}$ with the portal's plane. If the intersection point lies outside of the portal circle, it is moved to the nearest point on the circle boundary, as shown in Fig. 6 a. Note that due to the previous refinement of the path in partially visible spheres, either of $\mathbf{x}_i$'s neighbors can be a reconstructed border point. These steps are performed iteratively for all points of the path. This update

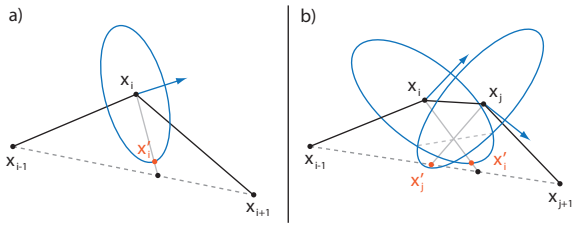


**Figure 6:** *The path post-processing computes smoothed point positions* $\mathbf{x}'$ *that lie on the portals. (a) The standard case of smoothing on a single portal. (b) The procedure for two overlapping portal circles.*

can change the projected start and end positions on the occlusion maps, so the refinement as described in Section 4.2 is re-computed in an additional pass after each iteration.

We distinguish two special cases of the post-processing. The first one is that of two overlapping portal circles. In such a situation, two neighboring points on $\mathcal{P}$, $\mathbf{x}_i$ and $\mathbf{x}_j$, can converge to a single point on the intersection line of the two circles, which prevents the points from moving further toward their smoothed positions. To resolve this problem, we perform a combined update of both points if they are closer than a small distance $\varepsilon$ (Fig. 6 b). The second special case occurs when a portal is completely contained in the cylindrical volume of its two neighboring portal circles. As the contained portal does not contribute to the overall volume of the path, we simply discard it.

The final camera trajectory is determined by Hermite interpolation [Far02] of the path between each pair of points. Two consecutive portals cannot lie on the same plane, which guarantees that a $C_1$ continuous interpolation can be found. The maximum curvature is bounded by the space between the two portals. In partially visible spheres, where the projection map forces the border point onto the margin of the sphere, there might not be enough room to add additional control points for the Hermite interpolation. In this case, in order to guarantee the $C_1$ continuity of the final path, the border point can be slightly moved toward the sphere center to create enough room for the interpolation.

## 5. Applications and Extensions

In the previous sections, we have presented the basic framework for visibility transition planning, which includes the precomputed visibility roadmap as well as a runtime planning, refinement, and smoothing strategy for visibility-aware camera transitions. This functionality is useful in many situations as-is. However, games and interactive environments often have unique, specialized requirements, and one single camera model cannot cater to all situations. One of the strongest aspects of our research is that the data structure and planning system provide the foundation and algorithmic tools to enable a variety of customized camera behavior that can be specialized to the needs of a particular game. In this section, we focus on applications and extensions enabled by our work. Because camera motion is more easily demonstrated with movement than with words or pictures, we refer frequently to the video that accompanies this publication.

**Large Camera Transitions.** Once a large camera transition has been computed, the most basic task is to move the camera along the path from the start point to the end point. All camera movement is realized using a physically-based simulation to determine the camera's six degrees of freedom: position, pitch, yaw, and roll. A spring is created for each degree of freedom between the actual position or orientation and the desired one (e.g., a location along the computed path). Our model is based on forward dynamics [BMH98] and is integrated over time with an explicit leapfrog scheme. This physical camera model ensures smooth camera movement, at the cost of small deviations in the exact camera path. In order to avoid collisions that could result from these deviations, a small safety buffer around scene geometry is included during sphere creation.

We demonstrate such a camera move in the *Urban* environment (Video Part 2), including a transition that zooms out from a close-up view in the streets to a top down view from above. While the shortest path quickly hides the focus point behind a skyscraper, the visibility aware transition keeps the target in focus until the last possible moment. Such transitions could assist in way-finding tasks in virtual environments by providing a natural facility to connect egocentric and allocentric viewpoints without losing context [BB08].

**Global Planning for Local Camera Control.** Our local camera model builds upon our global planning system in order to implement a camera that follows a moving target while striving to maintain visibility. It is analogous to the robotics problem of maintaining visibility and a fixed distance from a moving target [MCSBH04], except in a complex 3D environment. If visibility is broken because the user quickly ducks behind an obstacle, an unoccluded vantage is chosen via a ray-cast. While a naïve method might teleport the camera to the unoccluded point or make the occluding geometry artificially transparent (Fig. 2), our path planner finds a collision-free path to this position that regains visibility as quickly as possible. We demonstrate this facility in our *Island Village* level (Figs. 9 b and c, Video Part 3).

The planning computations for the camera model are constantly active in a thread running parallel to the game engine, as this decouples the complexity of the visibility planning from the application's frame rate constraints. For rendering the occlusion maps, this parallelization requires synchronization of the GPU with the main rendering step. The different states of our local camera model are shown in Fig. 7.

**Camera Target Switching.** Taken together, the previous two applications enable a target switching, where the camera's focus point switches between multiple players playing

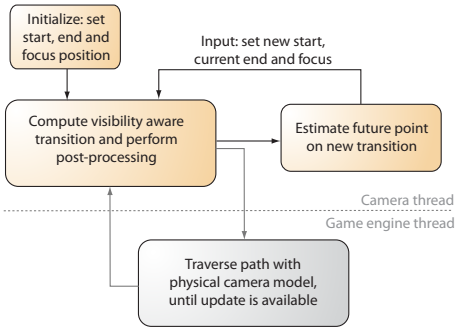


**Figure 7:** *Implementation of the local camera model. By predicting a future start point on a given path based on the time it took to compute it, a new start position can be estimated. The new transition path is computed in parallel while the camera travels along the current path.*

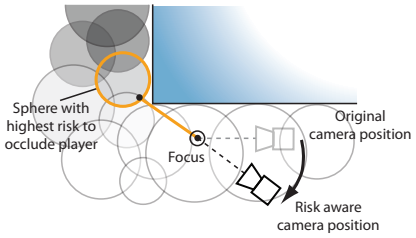


**Figure 8:** *Proactive camera movement: our algorithm finds the closest sphere (indicated in orange) that has reduced visibility from the original camera position. After detecting a high escape risk, the camera's viewing direction is aligned towards this sphere. Spheres with reduced visibility from the original camera position are shown in darker shades of gray.*

at the same time. Such a system could be used in a *spectator mode* where an outside spectator is observing the progress of a multi-player game. We demonstrate this facility in the *Woodland* level (Fig. 9 a, Video Part 4). The level design includes two houses and a connecting underground tunnel, which creates indoor regions that are completely invisible from the outside. As the focus dynamically changes between the two players, our visibility transition planning algorithm results in camera transitions that bring the new player into focus as quickly as possible. The smooth transitions give the viewer a sense of the environment's layout and the distance between the two players, which would not be conveyed if immediate camera cuts were used instead.

**Risk Prediction.** A purely reactive camera controller cannot keep a fast moving player in view at all times. In order to anticipate situations in which the player might escape the camera's region of visibility, our system uses an algorithm inspired by the target tracking work of Bandyopadhyay and colleagues [BLAJH06] and extended to 3D. Our data structure and path planning code permit an efficient implementation that adjusts the camera position in an effort to prevent the player from escaping view. This risk prediction algorithm uses the visibility-aware roadmap and planning architecture to find a path to the sphere closest to the player which is not visible from the camera's current position. This path represents the highest risk of escape. The camera's current position is rotated to a new vantage so that this escape route is in view. The transition planning system is used to move the camera to this new vantage point. An overview of this process is shown in Fig. 8, as well as in the *Arena* level in Part 5 of the video.

**Dynamic Occluders.** Often, real-time environments contain dynamic elements, such as closing doors or moving obstacles, and it is crucial that the camera takes these objects into account when moving through the scene. Fully dynamic environments where every environmental feature can move are uncommon since they invalidate acceleration structures necessary for collision detection, lighting, and other computations. Thus, we target the most common and advantageous case where the environment contains a small number of dynamic occluders.

To enable the computation of visibility transitions in the presence of dynamic occluders, our system performs an on-the-fly update of the information in the roadmap. Moving objects are approximated by one or more bounding spheres. When computing the A* search on the roadmap, our algorithm marks all connections between two portal circles that touch the bounding spheres as being occupied in order to prevent the camera from colliding with the moving object. To update the visibility information, our system projects the bounding sphere of the object onto the spheres of the roadmap from the direction of the focus point. All roadmap spheres within this cone are marked as having an occluded view of the focus point. During path refinement, the actual geometry of the moving object is included in the occlusion maps, making the camera take into account the actual shape of the object.

We demonstrate this ability to handle dynamic environments in two examples shown in Fig. 10, as well as Part 7 in the video. The first one covers a typical game setting where a door to a house is closing while the camera is entering it. Our system dynamically adapts to the changed visibility setting, and takes a detour through the window of the house. In another scene we demonstrate how the camera can find collision free and unoccluded paths through multiple moving objects.

**Additional Search Criteria.** In some situations an application might have additional constraints for the search path. For example, a path through a game level might require the camera to move along the ground, in a similar way that a player would do, instead of taking a short cut through the

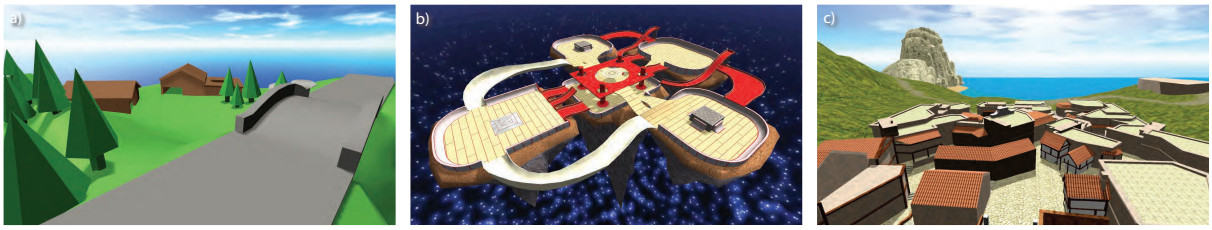


**Figure 9:** *Three of our game environments: (a) A woodland scene with an underground corridor, where we can perform complex transitions between two players. (b) A muti-player arena level. (c) A more complex island village environment.*

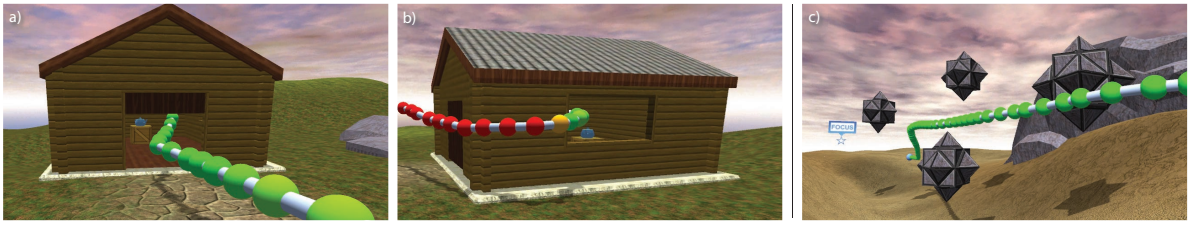


**Figure 10:** *Visibility transitions in dynamic environments: (a) When the door to this house is open, our algorithm computes a path through the door. (b) As soon as the door closes, an alternative route around the side of the house is automatically found. (c) This camera transition avoids multiple moving obstacles.*

air. We can modify our search framework to include additional constraints by modifying the cost function of the A* search. For the example above, we could include a height constraint, to make the camera move within a certain region above ground. To do this, we add an additional penalty term to Equation 1. This gives us a modified cost function

$$C'(e_{ij}) = C(e_{ij}) + \alpha^2 d(i,j) h(e_{ij}) \;, \qquad (2)$$

where $h(e_{ij})$ evaluates to zero if the edge $e_{ij}$ is within a given height range and increases to one if the edge is above or below this region. The weight of $\alpha^2$ ensures that this constraint is prioritized over the visibility. We demonstrate this constrained search in the *Island Village* (Video Part 6) with paths along the ground instead of above the houses. This constrained search could easily be extended by a variety of penalty terms, such as a weight term specified by a level designer to make the camera prefer landmarks in the environment seen from a certain angle or region.

**Statistics.** Run-time and precomputation statistics of the environments are given in Table 1. The roadmaps we compute for the different environments have between 819 and 3102 spheres. The largest roadmap, for the *Island Village* level, requires only 6.36MB. The precomputation times, the major part of which is computing the sphere visibilities, vary from 8.1 to 61.4 seconds and directly depend on the number of spheres and visibility distance. While precomputation time is unimportant in many disciplines, it is crucial in the design of interactive environments. Level designers must continually iterate during the design process, testing gameplay with each modification to the level's layout. A long precomputation time would hinder this iteration, making the camera control system unusable. Thus, our short precomputation time is critical for a practical system. The average path search and post-processing times depend on the length of the transitions. For the local models used in the *Arena*, this takes 1.8ms on average, while the long transition with dynamic objects demonstrated in the *Tea House* example requires almost 30ms. The local model must be extremely fast so that the camera will react adroitly to player movement. However, the longer transitions require several seconds for the camera to travel along the long path (five seconds in the tea house example) and the transition planning is comparatively short. Furthermore, these path computations are performed in a thread parallel to the game engine, which avoids any changes in the frame rate.

## 6. Conclusion

We have presented an algorithm that enables sophisticated camera transitions by phrasing the problem as a global search on a precomputed visibility-aware roadmap together with local runtime refinement. The balance between precomputation and runtime calculation allows our system to generate collision-free paths that emphasize visibility while maintaining real-time performance. Because it is global in nature, our method can generate large-scale camera transitions between distant points in a scene, even in complicated geometric situations. In addition, we have demonstrated a vari-

| Environment | Urban | Woodland | Arena | Tea House | Valley | Island Village |
|---|---|---|---|---|---|---|
| **Polygons** | 90 | 1778 | 15536 | 8918 | 8388 | 34721 |
| **Grid resolution** | 30x44x41 | 75x25x67 | 75x14x68 | 40x6x40 | 75x21x76 | 65x21x83 |
| **Spheres** | 1035 | 2152 | 2008 | 819 | 1106 | 3102 |
| **Roadmap nodes** | 5911 | 8730 | 3991 | 3999 | 7953 | 24989 |
| **Visibility distance** | 100% | 100% | 30% | 100% | 100% | 25% |
| **Precomputation [sec.]** | 8.1 | 61.4 | 27 | 35 | 51 | 28 |
| **Memory roadmap [MB]** | 3.61 | 4.29 | 2.92 | 2.07 | 5.10 | 6.36 |
| **Method** | Transitions | Switching | Local | Dynamic | Dynamic | Transitions |
| **Avg. Path Length [# nodes]** | 9.1 | 4.8 | 3.5 | 12.0 | 9.6 | 14.7 |
| **Avg. Search [ms]** | 4.46 | 7.44 | 1.80 | 27.67 | 25.1 | 14.48 |
| **Avg. Post-processing [ms]** | 1.20 | 2.41 | 0.41 | 2.34 | 6.61 | 0.63 |

**Table 1:** *Roadmap sizes, precomputation times and run-time performance measurements for the different parts of our algorithm.*

ety of applications of this basic functionality. Our approach can handle dynamic occluders, enhance third-person camera models, perform risk prediction, and respect additional constraints. The computation time is output sensitive. Large camera transitions require tens of milliseconds, while shorter ones used in our third-person camera are computed in one to four milliseconds.

Limitations of our current system direct us to areas of future work. Currently, we support only single-point focus targets. However, we plan to extend the algorithm to keep an entire object in view by aggregating the visibility estimations for multiple focus targets during the path search, and superimpose the occlusion maps during the refinement phase. Currently, our system computes a fairly dense sampling of the ambient space using the roadmap spheres. Accurately sampling small regions of the environment may require tiny spheres. Such a sampling leads to more accurate visibility estimations, better risk predictions, and superior camera paths. However, extremely dense roadmaps may impede performance. We plan to explore level-of-detail for the visibility-aware roadmap, where "highways" are traveled for large distances, and "local roads" are used for more fine-scale movement. Additionally, the current framework only supports dynamic *occluders* and cannot add new connections to the roadmap. While our system works well for environments with a limited number of dynamic objects, performance might degrade for large changes in the scene. In the future, we plan to explore techniques that propagate environmental changes directly onto the data structure, such as elastic roadmaps [YB06], in order to allow large-scale dynamic scene changes.

In this work, we have focused on visibility, since achieving an unoccluded view of an object is the most fundamental goal of camera control. In future research, we wish to include higher-level goals into our system such as cinematographic rules that influence perceptual aspects of the computed transitions. While our formulation of the search accommodates additional search criteria, more work is needed to properly incorporate aesthetic guidelines to convey different styles or moods. One particularly exciting avenue of research is to generalize from artist-created camera paths during transition planning so that greater control over the character of the transitions is given to the designer.

**Acknowledgments**

We would like to thank Serkan Bozyigit for his work on dynamic occluders, Michael Spreng for porting the precomputation to C++, and Theodor Mader for his help with the graphics engine. We are grateful to Propaganda Games / Disney Interactive Studios for providing the *Island Village* level. We are thankful to the members of the ETH computer graphics lab, Disney Research Zurich, and the anonymous reviewers for their helpful observations and suggestions.

**References**

[BAJH07] BANDYOPADHYAY T., ANG JR. M., HSU D.: Motion planning for 3-d target tracking among obstacles. In *Proc. Int. Symp. on Robotics Research* (2007).

[BB08] BYRNE P., BECKER S.: A principle for learning egocentric-allocentric transformation. *Neural Computation 20*, 3 (2008), 709–737.

[BGL98] BARES W. H., GRÉGOIRE J. P., LESTER J. C.: Realtime constraint-based cinematography for complex interactive 3d worlds. In *AAAI '98/IAAI '98: Proceedings of Artificial intelligence/Innovative applications of artificial intelligence* (1998).

[Bit02] BITTNER J.: *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University, October 2002.

[BLAJH04] BANDYOPADHYAY T., LI Y., ANG JR. M., HSU D.: Stealth tracking of an unpredictable target among obstacles. In *Algorithmic Foundations of Robotics VI*, Erdmann M. et al., (Eds.). Springer-Verlag, 2004, pp. 43–58.

[BLAJH06] BANDYOPADHYAY T., LI Y., ANG JR. M.,

HSU D.: A greedy strategy for tracking a locally predictable target among obstacles. In *Proc. IEEE Int. Conf. on Robotics & Automation* (2006), pp. 2342–2347.

[BMH98] BROGAN D. C., METOYER R. A., HODGINS J. K.: Dynamically simulated characters in virtual environments. In *IEEE Computer Graphics and Applications* (1998), vol. 15(5), pp. 58–69.

[Bra04] BRADSHAW G.: Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics 23* (2004), 1–26.

[COCSD03] COHEN-OR D., CHRYSANTHOU Y. L., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. In *IEEE Transactions on Visualization and Computer Graphics* (July 2003), vol. 9, pp. 412–431.

[CON08] CHRISTIE M., OLIVIER P., NORMAND J.-M.: Camera control in computer graphics. *Computer Graphics Forum 27*, 8 (2008), 2197–2218.

[DP85] DECHTER R., PEARL J.: Generalized best-first search strategies and the optimality of a*. *J. ACM 32(3)* (1985), 505–536.

[Dur00] DURAND F.: A multidisciplinary survey of visibility, 2000.

[DZ94] DRUCKER S. M., ZELTZER D.: Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface '94* (Banff, Alberta, Canada, 1994), pp. 190–199.

[Far02] FARIN G.: *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

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

[HHS01] HALPER N., HELBING R., STROTHOTTE T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Comput. Graph. Forum 20*, 3 (2001).

[Lai05] LAINE S.: A general algorithm for output-sensitive visibility preprocessing. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), pp. 31–40.

[LaV06] LAVALLE S. M.: *Planning Algorithms*. Cambridge University Press, 2006.

[Laz01] LAZEBNIK S.: *Visibility-Based Pursuit Evasion in Three-Dimensional Environments*. Tech. Rep. CVR TR 2001-01, Beckman Institute, University of Illinois, 2001.

[LC08] LI T.-Y., CHENG C.-C.: Real-time camera planning for navigation in virtual environments. In *Proceedings of Smart Graphics* (2008), pp. 118–129.

[MCMS*07] MURRIETA-CID R., MUPPIRALA T., SARMIENTO A., BHATTACHARYA S., HUTCHINSON S.: Surveillance strategies for a pursuer with finite sensor range. *Int. J. Rob. Res. 26*, 3 (2007), 233–253.

[MCSBH04] MURRIETA-CID R., SARMIENTO A., BHATTACHARYA S., HUTCHINSON S.: Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *Proceedings IEEE International Conference on Robotics and Automation* (2004), pp. 479–484.

[MS07] MASEHIAN E., SEDIGHIZADEH D.: Classic and heuristic approaches in robot motion planning—a chronological review. In *Proceedings of World Academy of Science, Engineering and Technology* (2007), vol. 23.

[NRG04] NIEDERBERGER C., RADOVIC D., GROSS M.: Generic path planning for real-time applications. In *Proceedings of Computer Graphics International* (2004), pp. 299–306.

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[SGLM03] SALOMON B., GARBER M., LIN M. C., MANOCHA D.: Interactive navigation in complex environments using path planning. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics* (2003), pp. 41–50.

[SLN00] SIMÉON T., LAUMOND J.-P., NISSOUX C.: Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics 14*, 6 (2000).

[VM05] VARADHAN G., MANOCHA D.: Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Robotics: Science and Systems* (2005), The MIT Press, pp. 25–32.

[VSK*02] VIDAL R., SHAKERNIA O., KIM H. J., SHIM D. H., SASTRY S.: Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *Robotics and Automation, IEEE Transactions on 18*, 5 (2002), 662–669.

[YB06] YANG Y., BROCK O.: Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. In *Proceedings of Robotics: Science and Systems* (Philadelphia, USA, August 2006).

[YL02] YANG L., LAVALLE S.: An improved random neighborhood graph approach. vol. 1, pp. 254–259.