EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

ExploreMaps: Efficient Construction and Ubiquitous Exploration of Panoramic View Graphs of Complex 3D Environments

M. Di Benedetto[1], F. Ganovelli[1], M. Balsa Rodriguez[2], A. Jaspe Villanueva[2], R. Scopigno[1] and E. Gobbetti[2]

[1]Visual Computing Group - ISTI-CNR, Italy
[2]Visual Computing Group - CRS4, Italy

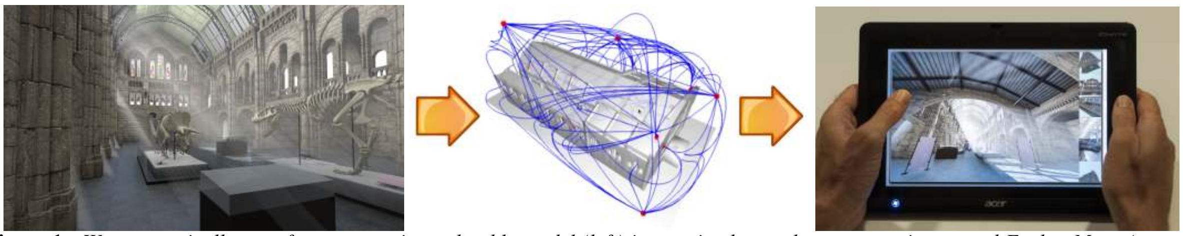


Figure 1: *We automatically transform a generic renderable model (left) into a simple graph representation named ExploreMaps (center), where nodes are nicely placed point of views that cover the visible model surface and arcs are smooth paths between neighboring probes. The representation is exploited for providing visual indexes for the 3D scene and for supporting, even on low-powered mobile devices, interactive photorealistic exploration based on precomputed imagery (right).*

Abstract

We introduce a novel efficient technique for automatically transforming a generic renderable 3D scene into a simple graph representation named ExploreMaps, where nodes are nicely placed point of views, called probes, and arcs are smooth paths between neighboring probes. Each probe is associated with a panoramic image enriched with preferred viewing orientations, and each path with a panoramic video. Our GPU-accelerated unattended construction pipeline distributes probes so as to guarantee coverage of the scene while accounting for perceptual criteria before finding smooth, good looking paths between neighboring probes. Images and videos are precomputed at construction time with off-line photorealistic rendering engines, providing a convincing 3D visualization beyond the limits of current real-time graphics techniques. At run-time, the graph is exploited both for creating automatic scene indexes and movie previews of complex scenes and for supporting interactive exploration through a low-DOF assisted navigation interface and the visual indexing of the scene provided by the selected viewpoints. Due to negligible CPU overhead and very limited use of GPU functionality, real-time performance is achieved on emerging web-based environments based on WebGL even on low-powered mobile devices.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—

1. Introduction

With the widespread availability of mobile graphics terminals and WebGL-enabled browsers, 3D graphics over the Internet is thriving. Thanks to recent advances in 3D acquisition and modeling systems, high-quality 3D models are becoming increasingly common, and are now potentially available for ubiquitous exploration. In current 3D repositories, such as Blend Swap, 3D Café or Archive3D, 3D models available for download are mostly presented through a few user-selected static images. Online exploration is limited to simple orbiting and/or low-fidelity explorations of simplified models, since photorealistic rendering quality of complex synthetic environments is still hardly achievable within the real-time constraints of interactive applications, especially on on low-powered mobile devices or script-based Internet browsers. Moreover, navigating inside 3D environments, especially on the now pervasive touch devices, is a non-trivial task, and usability is consistently improved by employing assisted navigation controls [CON08]. Spatially indexed photographic imagery, popularized by Internet systems such as Google

StreetView and Bing Maps StreetSide, is, on the other hand, proving to be an effective mean for generating compelling interactive virtual tours of real locations. These systems achieve visual quality by presenting captured imagery rather than synthetic reconstructions, and simplify interaction by exploiting connections among captured images to guide the user. Designing such guided walkthroughs for synthetic 3D environments manually is, however, a hard and time-consuming task.

In this work, we introduce an approach aimed at automatically providing a richer experience in presenting 3D models on dedicated web sites. The method builds on a novel efficient technique for transforming a generic renderable 3D scene into a simple graph representation, dubbed ExploreMaps, where nodes are nicely placed panoramic views, called probes, and arcs are smooth panoramic video paths connecting neighboring probes. Our GPU-accelerated unattended construction pipeline distributes probes so as to guarantee complete coverage of a generic scene, before clustering them using perceptual criteria, determining preferential viewing orientations, finding smooth good looking connection paths, and reordering probes in a linear arrangement suitable for thumbnail-bar presentation. Probe images and path videos are then computed with off-line photorealistic renderers, overcoming real-time rendering limitations. At run-time, the graph is exploited both for generating visual scene indexes and movie previews, and for supporting interactive exploration through a low-DOF assisted navigation interface. Usability and sense of presence are increased by leaving orientation and field of view free when looking at the scene from a probe position, and gently converging to the closest target "good orientation" during transitions. Due to negligible CPU/GPU usage, real-time performance is achieved on emerging WebGL environments even on low-powered mobile devices (see Fig. 1). Our core contributions are:

- **View sampling:** a novel, unattended, GPU accelerated visibility-driven technique for sampling renderable surface models into a set of panoramic views, and a novel approach for determining a good set of representative panoramic views for general 3D scenes, extending the scope of current viewpoint orbiting solutions [SLF*11];
- **Path creation:** a novel method for determining view connectivity, a purely visibility-guided GPU accelerated method for defining smooth good-looking connecting trajectories, and a technique for exploiting the graph structure to reorder the views in a linear arrangement well suited for exploration through thumbnail bars.
- **Ubiquitous exploration:** a mobile WebGL-friendly system able to guide users and sustain interactive performance while presenting photorealistic environments.

The method is robust, completely automatic, and applicable to many kinds of complex renderable scenes, materials, and lighting environments. The major limitation of the approach is, by design, the restriction of viewing positions to the precomputed probes and paths. This limitation, however, allows us to replace the problem of streaming 3D scenes with the streaming of images and videos, precomputed with movie-quality rendering systems without hard timing constraints, and the problem of 3D navigation in space with a much simpler graph exploration, improving usability through a low-DOF assisted navigation interface and a graph-based visual indexing of the scene.

2. Related work

In the following, we will briefly discuss the approaches that are most closely related with our work. For further details, we refer the reader to established surveys on massive model rendering [DGY07, GKY08, YGKM08], image-based rendering [SCK07], camera control [CON08], view planning [SRR03], and mobile graphics [CPAM08].

While in recent years, research efforts have produced systems capable of rendering moderately complex environments on the web and/or mobile devices [MLL*10, NKB10, BGM*12, GMB*12, BGMT13], real-time constraints limit achievable quality to moderate geometric complexity, simple shading models and/or baked illumination. We focus instead on supporting photorealistic views of complex scenes through precomputation, and on exploiting them both for visual scene indexing and constrained navigation.

Using image-based techniques to remove limitations on scene complexity and rendering quality for interactive applications, as well as to improve application usability is an old idea, that dates back at least to the branching movies of the 80s [Lip80] and the navigable videos and environment maps of the 90s (e.g., [Che95, KFL01]). More recently, these approaches have flourished in the context of applications that exploit camera pose (location, orientation, and field of view) and sparse 3D scene information to create new interfaces for exploring physical places by browsing large collections of photographs or videos [SSS06, Vin07, KCSC10, TKKT12, SS12]. While much of earlier research has focused either on authored paths or on pre-acquired large photo/video collections, with an emphasis on view interpolation, image-based rendering from captured samples, or interfaces for navigation among large sets of precomputed images, we focus instead on how to efficiently and automatically *create* a set of representative views and connections starting from a given 3D environment, and on how to increase the sense of presence during constrained navigation. Since we restrict the possible camera positions (but not orientations and fields of view), we can side-step the complex problem of computing pixel-accurate viewpoint interpolations in general shading environments [SKG*12]. Our method is therefore applicable to scenes including effects such as participating media, reflections, and refractions. The question of what are *good* views of a 3D object has been addressed by many researchers in perception, computer vision and computer graphics. In this work, we optimize view placement (and preferential orientations) by combining a perceptual metric [SLF*11] with our new criterion based on viewpoint stability and scene coverage.

Placing viewpoints to guarantee a complete and accurate sampling is an extensively studied computer vision problem [SRR03]. Reconstructing a "virtual" object is a different problem, since there are no physical constraints on the field-of-view, position, orientation, and motion of the virtual sensors, and acquisition errors are limited to the discretization done by rasterization. View selection in this context has mostly been studied in image-based rendering literature. Most techniques use a fixed set of views [Rap98, ABB*07], leading to gaps or holes for non-trivial objects. Fleishman et al. [FCOL00] and Vazquez et al. [VFSH02] introduced adaptive techniques for covering polygonal models, but require the prior definition of a "walking zone" for bounding camera positions. Our approach, applicable to generic scenes, bears similarities with search-based methods, which use optimization criteria based on expected entropy [WDH*06], visibility [WM03], or silhouettes [Abi95]. We propose a novel visibility-based approach based on panoramic images and an optimized GPU implementation, combined with a view clustering and selection scheme. Voronoi-based sampling is employed by Wilson and Manocha [WM03], which, however, use a greedy optimization framework based on the detection of "skin edges" and a simplified encoding in textured depth meshes, limiting the approach to architectural scenes.

3. Creating the ExploreMaps graph

The input scene is assumed to have a geometry, used for view discovery, as well as a shaded representation, used for high quality rendering. The only assumptions made on the geometric representation is that its bounding box is known, that it can be rasterized producing depth and normal buffers, and (optionally) has a preferential up vector ($+Y$ by default). The explicit definition of the up direction can be removed by employing an unsupervised upright direction solver for man-made objects [FCODS08, JWL12]. The shaded representation, instead, is any description of the same scene that can be given as input to an external high-quality renderer. In the simplest form, it consists of the geometric representation enriched with lights and material properties.

```
VirtualExploration() {
  v = PlaceFirstProbe();
  V += MaximizeSeenSurface(v);
  while (not IsComplete(V)) {
     v = PlaceNewProbe();
     V += MaximizeSeenSurface(v);
  }
}
```

Listing 1: *Placing a set of probes to cover all visible surfaces*

```
MaximizeSeenSurface(v){
  do{
     Snew = NewSurface(v);
     b = Barycenter(Snew);
     v = MoveTo(b);
  } while (not Converged());
  return v;
}
```

Listing 2: *Moving a probe to maximize the new surface seen*

3.1. Discovering the scene

We start by formally defining the problem of exploring a scene given only its bounding box and an abstract method for rasterizing the geometry. Let $\mathbf{v} = \{v_0, \ldots, v_{n-1}\}$ be a set of n probes, S the entire input surface to be discovered, $S(v_i) \subseteq S$ the portion of surface *seen* by probe v_i, and $S(\mathbf{v}) = \bigcup_{\forall v_i \in \mathbf{v}} S(v_i)$. With this notation, the exploration problem is posed as the problem of finding a set of probes $\mathbf{v}$ such that $S \subseteq S(\mathbf{v})$. We say that a set of probes satisfying this condition is *complete*, which means it sees the whole input surface. Note that we do not have prior knowledge of S, which must be *discovered* by the algorithm.

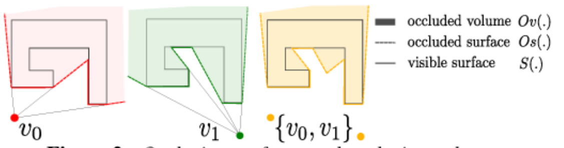


Figure 2: *Occlusion surfaces and occlusion volumes.*

Incremental algorithm. Our incremental algorithm, see Listing 1, starts with an empty set of probes and, at each iteration, adds a new probe and optimizes its position so that it sees a portion of the surface not visible from previous probes, until the coverage is complete[†]. Fig. 2 illustrates a 2D example of an object, with two probes, the portion of surface seen, and the *occluded surfaces* $Os(v_i)$, i.e., the surfaces joining the discontinuities of $S(v_i)$. The occluded volume for a probe, $Ov(v_i)$ is the (infinite) region of 3D space non visible from v_i and the occluded volume from a set of probes is the intersection of the single viewing volumes $Ov(\mathbf{v}) = \bigcap_{v_i \in \mathbf{v}} Ov(v_i)$. Finally the occluded surfaces of a set of probes, $Os(\mathbf{v})$ is the union of the single occluded surfaces intersected with the occluded volume: $Os(\mathbf{v}) = \bigcup_{v_i \in \mathbf{v}} Os(v_i) \cap Ov(\mathbf{v})$. The occluded surfaces are typically used as candidate locations to place the next probe [WM03] as a way to look "behind the corner". It is easy to prove that if we found a set of probes $\mathbf{v}$ such that the occluded surface is empty, it means that all the reachable surface is seen by some probe, that is: if $Os(\mathbf{v}) = \emptyset$ then $S(\mathbf{v}) = S$. Since we are interested only on the surface reachable from the outside, placing the first probe outside the scene bounding box is a sufficient initialization (PlaceFirstProbe()).

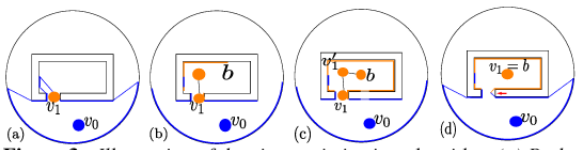


Figure 3: *Illustration of the view optimization algorithm. (a) Probe v_1 is added on the occluded surface $Os(v_0)$; (b) the new surface portion seen by v_1 and its barycenter b are shown; (c) the probe is moved to b, and visible surface and barycenter are recomputed; (d) the probe ends up on the barycenter of the surface seen.*

Finding a new probe position. The position for placing a new probe (PlaceNewProbe()) is chosen at random in the current occluded surface. This is a common choice for many view planning strategies, as it guarantees that some new portion of the surface that will be seen. We then start an iterative optimization algorithm, described in Listing 2. At each step, the algorithm tries to maximize the area of the surface seen by the new probe and unseen by previous ones. This is done by

[†] Note that, while in this paper we focus on complete automation, the method could also start with user-defined set of probes, e.g., to force inclusion of semantically relevant/nice shots in the graph

iteratively moving the probe position towards the barycenter of the surface visible from its current position (MoveTo(b)). If this is not possible because the segment connecting the current position and b intersects the surface, the probe is put halfway between the current position and the intersection point. This strategy, reminiscent of Lloyd relaxation [Llo82], ends when an equilibrium position is reached, i.e., when the barycenter of the surface seen is the same as current position itself (up to a small threshold). The approach implicitly assumes that the probe is at the interior of a closed surface, e.g., inside a building, since, otherwise, the location of the barycenter of the unseen surface would tend to be too close to the surface itself, or even collapse onto it. We therefore artificially bound the scene with a spherical background object, used by the discovery algorithm, but ignored in the final panoramas. Figure 3 illustrates the positioning of a probe, from its placement on the occluded surface to the convergence of the algorithm. The situation depicted in the example is fairly common. We can see that a probe "entering" in a room tends to position itself at the room's center. Note that not all the surface seen by the new probe during the first steps of the optimization will be visible also from the final position (see the portion indicated by a red arrow in Fig. 3.(d)).

3.2. Optimizing the set of probes

Our exploration algorithm finds a set of probes that globally guarantee the full coverage of the reachable surface (see Figure 4, left). While in principle we could build a graph over this set, we infer a new graph that also takes into account perceptual criteria, in order to enhance user experience. This is done by clustering the set of probes obtained by the exploration phase and replacing each cluster with a representative probe so that the resulting graph is almost equivalent in terms of coverage but *better* in terms of browsing.

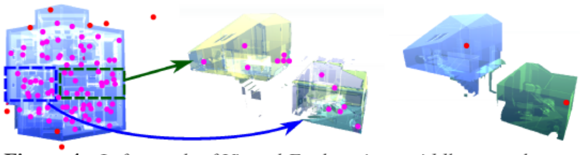


Figure 4: *Left: result of Virtual Exploration; middle: two clusters highlighted; right: synthesis of the clusters.*

Probe Clustering. The clustering is performed by applying the Markov Cluster Algorithm (a.k.a. MCL [VD08]), with default power and inflation parameters ($e = 2$, $r = 2$) to the coverage graph. MCL is a well known randomized algorithm that finds out *natural* clusters of nodes in a graph. The key idea is that when you take random walks starting from a node, it is more likely that you stay in the same cluster of the starting node than you jump to another cluster. MCL can use weights on the arcs (i.e., the weight determines how likely the arc is crossed in the random walk), and we assign the weights as the amount of overlap in visible surface between the two probes connected by the arc. This will tend to cluster together probes that see the same area. The algorithm terminates when it reaches a steady state, obtaining a number of clusters $c_0 \dots c_m$. Two examples of these clusters are shown on Figure 4 (middle). In a second phase, we find the *synthesis* of each cluster c_i, i.e., a single new probe that sees most of the surface seen by all the probes in the cluster, while providing a significant panorama in terms of perceptual criteria.

Stability Criterion. Secord et al. [SLF*11] introduce criteria for assessing the quality of a view with the purpose of automatically defining the best point of view in a perceptual sense. However, there are important differences with our case, because we must support general object viewing, including exploration of indoor scenes, while the metric proposed in [SLF*11] is only concerned with the object-in-hand paradigm, assuming that the object silhouette is entirely visible. This means that attributes concerning the projected surface area, silhouette and semantic (see [SLF*11]) are hardly applicable in our case, and the linear goodness function they propose would reduce to:

$$G(v) = \alpha\ MaxDepth(v) + \beta\left(1 - \int H(z)^2 dz\right)$$

where $Maxdepth$ is the maximum depth value, H is the normalized histogram of the depth values and α and β are $[0,1]$ weights to combine the two metrics. However, it is easy to verify that these criteria alone could lead to unnatural positions. A clear example is a point of view "behind" a column, from where there may be large maximal depth and uniform depth distribution, but where the feeling is to hide behind a column and not exploring the scene. Therefore, we weight $G(v)$ with the *stability* $St(v)$ of the point of view v, leading to a goodness function

$$G_s(v) = G(v) \cdot St(v)$$

We say that a point of view is stable if the view does not change much with respect to the neighborhood. In the previous example, it is clear that a position behind a column (that is, near to an occluder) is not stable because a small displacement would unveil a large part of the scene. In order to formalize this idea, we define a distance function between two points of view v and w as $\Delta(v,w) = max\{\phi(v,w), \phi(w,v)\}$, where $\phi(v,w) = Diff(P_w(S(v)), D(w)))$, $S(v)$ is the 3D surface seen from v, $P_w(.)$ is the projection of a 3D surface on w, $D(.)$ is the depth map from a point of view and $Diff(.,.)$ is the number of different depth values in the comparison between two depth maps. We thus define the stability of a point as $St(v) = max_{w \in N(v)}\{\Delta(v,w)\}$, where $N(v)$ is a set of point near v. In our system we used the points distributed on the sphere centered at v with radius equal to the distance of the *near plane* used to produce the panoramas.

Probe synthesis. The synthesis step starts from the position of the probe within the cluster that has the largest coverage of the surface seen by all probes in the cluster. We then perform a local randomized search using a *simulated annealing* approach, which applies small random perturbations of a probe's position, with an objective function that combines the coverage and the perceptual metric using a weighted average. Upon convergence, using the same perceptual metric, we determine a small set of orientations for each probe that will serve both as preferred arrival orientations when the user move to the probe, and as the *look-at* orientation used when generating thumbnails to represent the probes in a navigation application. Best orientations for each view are selected by finding the dominant peaks in the goodness function with mean-shift clustering [CM02]. The up vector for the dataset, if present, is used to avoid upside-down views.

3.3. Connecting probes

Once we have the new set of probes, we have to define which probes must be connected with an arc and the geometric path that connects them. One simplistic choice would be to connect two probes if they are mutually visible from each other. This solution is definitely not acceptable, since it would drastically reduce navigation possibilities. For instance, see Fig. 5, a probe in a room may not necessarily directly see a probe out of the window, nonetheless the transition from the first to the second makes perfect sense. More generally, view planning does not even guarantee that nearby probes are in general mutually visible. A smarter choice would be connect probes that see a common point in the surface. Even though this strategy would perform much better, it is easy to see that a number of common cases break the same rule (see, e.g., the same example in Fig. 5). We have thus decided to *connect two probes if there exists a point in space that is visible from both of them*. which generally leads to a meaningful set of arcs (see Figure 5 right and accompanying video). The technique for finding smooth feasible paths is explained in Sec. 4.2.

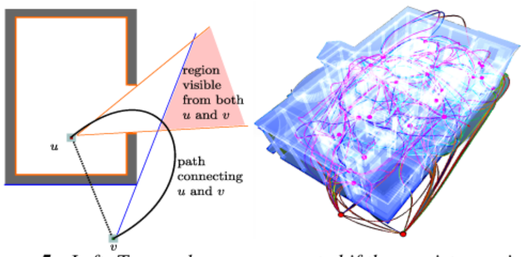

region visible from both u and v
u
path connecting u and v
v

Figure 5: *Left: Two probes are connected if there exists a point in space that is visible from both of them. Right: the path graph found for the German house example.*

3.4. Reordering probes

The final result of our exploration algorithm is a graph representation of the input scene in terms of probes and paths. We finalize graph construction by reordering nodes in a scene-coherent way, by computing a weighted minimum linear arrangement (MLA) of the probes, i.e., a permutation Π of the probes such that the cost $\sum_{ij \in E} w_{ij} |\Pi(i) - \Pi(j)|$ is minimal, where w_{ij} is the reciprocal of the length of the path connecting probe i to probe j. Intuitively, this ordering will attempt to cluster together the nodes that are close-by and connected by paths (see Fig. 6). We use this ordering at run-time to present nodes in a thumbnail bar using a logical exploration order. Since the MLA is known to be NP-hard, we heuristically approximate the solution using a multilevel solver [SRB06].

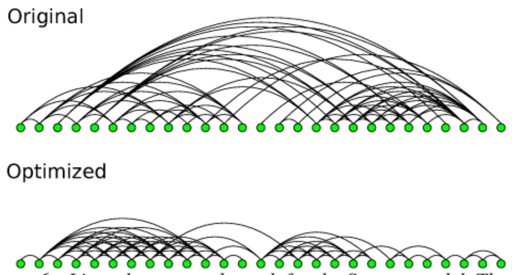


Figure 6: *Linearly arranged graph for the Sponza model. The optimized layout (bottom) reorders probes in a more coherent manner, moving nearby probes that are closely connected by short paths. This ordering is used for thumbnail-bar navigation (see Sec. 5).*

4. Efficient GPU Implementation

The different phases of ExploreMap construction have been mapped to an efficient and robust GPU implementation, whose major components are probe placement (Sec. 4.1) and path generation (Sec. 4.2).

4.1. Probe placement

The structure of the GPU implementation of our exploration algorithm is illustrated Fig. 7. Although the GPU algorithm follows the scheme presented in Listing 1 and Listing 2, there are several technicalities that need to be addressed.

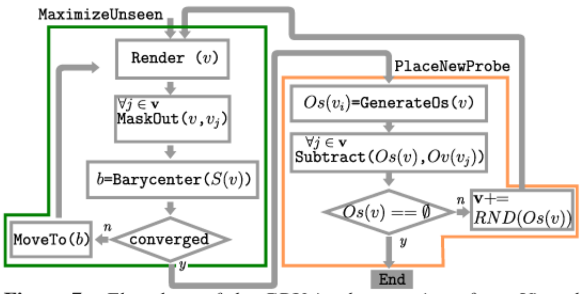

MaximizeUnseen
Render (v)
$\forall j \in \mathbf{v}$ MaskOut(v,v_j)
b=Barycenter($S(v)$)
converged
n
y
MoveTo(b)
PlaceNewProbe
$Os(v_i)$=GenerateOs(v)
$\forall j \in \mathbf{v}$ Subtract($Os(v)$,$Ov(v_j)$)
$Os(v) == \emptyset$
n
$\mathbf{v}$+= $RND(Os(v))$
y
End

Figure 7: *Flowchart of the GPU implementation of our Virtual Exploration algorithm.*

A panoramic view is composed of 6 renderings from the probe position so as to create an environment cube map, and the surface $S(v)$ is the set of 3D points obtained by unprojecting all the fragments belonging to the renderings in v. In the following, when we refer to operations on the (panoramic) views, the intended meaning is to all their 6 sub-views.

Render. The function `Render(v)` performs the rendering of the scene for each view with a *render-to-texture* (RTT) approach, producing a *depth map* $D(v)$ and a *normal map* $N(v)$ of the acquired surface. These two maps are then analyzed with a full-screen quad pass to generate a *discontinuity* (or *jump*) map $J(v)$, a map of boolean values where *true* indicates the border of the occluded surface $Os(v)$. We perform a full-screen pass to find out in parallel, for each acquired pixel, if it is on a discontinuity by testing if its projection in world space is more distant than a threshold τ_{pln} from the plane defined by its neighbor pixels and their respective normals. This is slightly more sophisticated than just testing the difference in the depth map or the distance in world space, for it takes into account the local orientation of the surface with respect to the view direction, as shown in Fig. 8.

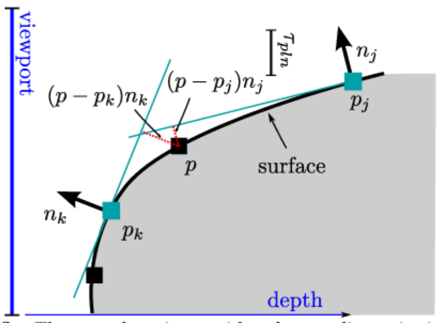


Figure 8: *The sample p is considered on a discontinuity if it is farther than τ_{pln} from any of the planes built with its adjacent samples positions and normals.*

MaskOut. Once $D(v)$, $N(v)$ and $J(v)$ have been produced, we need to find out how much of the acquired scene surface is seen *only* by the new probe. This is done by using a shadow map-like approach, treating the surface acquired by all previous probes as a shadow caster on the new, shadow receiving probe. The test, referred to as `MaskOut(`v`, `v_j`)`, is asymmetric, meaning that it only determines which fragments of $S(v)$ will be lost in the comparison against v_j. This phase starts by generating a *mask* map $M(v)$ where all pixels are set as valid, e.g., not masked. Then, with a series of full-screen passes, we try to invalidate the pixels belonging to already acquired surfaces. For this purpose, we use the depth map $D(v_j)$ of previous probes as a shadow buffer and perform the visibility test with the corresponding pixel of the current probe in $D(v)$. Note that since each view consists of 6 rendering, `MaskOut(`v`,`v_j`)` would require, in a trivial implementation, 36 renderings to be completed. We optimize the operation by skipping non-intersecting frustum pairs.

Barycenter. To compute the barycenter of non-masked parts of the scene seen by the probe (`Barycenter(`$S(v)$`)`), we consider $D(v)$ as a range map whose regular tessellation form the acquired scene surface. A full-scene pass is used to generate a *barycenter* map $B(v)$ that will contain, at each pixel (x,y), the barycenter of a surface quad obtained tessellating the samples $(x+\delta_x, y+\delta_y)$ $\delta_x, \delta_y = 0,1$ of $D(v)$. Note that only the newly seen surface must contribute to the barycenter computation, and quads over discontitnuities must not be taken into account (they are no other than the occlusion surface of the view). To this purpose, we access the discontinuity map $J(v)$ and the mask map $M(v)$ to ignore the contribution of quads that contain at least one vertex classified as a discontinuity or masked out. At the end of this process, every pixel in $B(v)$ will contain its associated, area-weighted barycenter b_q, or zero if it has been rejected due to discontinuities or masking. Calculating the barycenter of the acquired surface then consists of summing all values in $B(v)$: this is accomplished by applying a parallel *reduction* on $B(v)$ with the sum operator, easily done by generating mipmap pyramids. The barycenters of the six views, thus contained in the top-level 1×1 pixel maps, are then read back on the CPU and summed to produce the final probe barycenter.

Converged. Convergence is tested using the relative length of the movement from the current probe position to its barycenter and variation of acquired area. When one of these quantities is below a given threshold (or a max number of iterations has been reached), the probe terminates its exploration.

MoveTo. As explained, moving the position p of the probe towards the barycenter b (`MoveTo(`b`)`) requires to test that the segment connecting p and b does not intersect the surface, i.e., they are in line of sight from each other. Luckily, this test is trivial and only consists of projecting b on the depth map $D(v)$ and check if it is visible or in shadow.

Placing a new probe. The first step (`GenerateOs(`v`)`) consists of generating a sampling of the occluded surface $Os(v)$. This is done by explicitly generating a tessellation of the occluded surface by means of a geometry shader. We issue the rendering of a regularly tessellated grid with vertices associated to samples in the map of discontinuities $J(v)$. If at least one of the is a jump, the geometry shader outputs all 3 vertices by reading the depth values from $D(v)$ and unprojecting them in world space, otherwise the input triangle is discarded. These surviving triangles are are no other than a tessellation of the occluded surface $Os(v)$. The output vertex stream is recorded by the transform feedback into a buffer that is then read back in CPU memory, where triangles are densely sampled into a texture $T(v)$. With the hardware used in our tests, this proved to be faster than using tessellation and geometry shaders to generate samples directly on the GPU. The second step, (`Subtract(`$Os(v)$`, `$Ov(v_j)$`)`), consists of erasing all the samples outside the occluded volume, i.e., that are visible by at least one of the previous probes. Again, this is accomplished with a shadow map-like approach: we first test samples visibility using the depth map of previous probes, then use a geometry shader and the transform feedback to compact the samples list by eliminating visible ones. After reading back the samples contained in the transform feedback buffer, they are added to the occluded surface samples list L. The starting position of a new probe is then selected by randomly picking a sample from L. If L is empty, then the whole occlusion volume has been carved and the set of probes generated so far is *complete*.

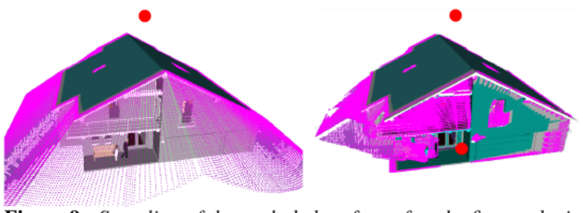


Figure 9: *Sampling of the occluded surface after the first probe is positioned (left, shown as a red sphere), and after* 7 *probes (right).*

Figure 9 shows probe positions (red spheres), surface seen (dark green), and sampling of the occluded surface (magenta) at two steps of our virtual exploration algorithm.

```
bool FindPath(v,u){
 v_curr = v;
 do{
   SetCamerasFrom(v_curr);
   RenderSceneOffset();
   EnableOcclusionQuery();
   RenderOffset(DM(u));
   nfrag = FinishOcclusionQuery();
   if (nfrag > 0) {
     p = ClosestTo_u();
     if (p == u) v_curr = u;
     else v_curr += delta * Normalize(p-v_curr);
   }
 } while( (v_curr != v) && (v_curr != u));
 return (v_curr == u);
}
```

Listing 3: *Finding a path between two probes.*

4.2. Path generation

In order to check if there is a point that sees two probes v and u, and to define a path connecting them, we adopt the following GPU accelerated strategy (see Listing 3). From the probe v, we render the scene, then enable hardware occlusion queries and render a tessellation of the depth map of u ($\mathsf{DM}(u)$), assigning to each vertex of the tessellation its distance from u. If at least a fragment of the tessellation is written in the buffer, the corresponding 3D point can be seen both from u (because part of the depth map tessellation) and from v (because the tessellation is rendered from v). We thus know that the two must be connected. In order to find a path, we can then move v by a fixed step towards the common point closest to u and iterate the algorithm, as shown in Fig. 10.

A very important detail is that each rendering of the scene must be done with an offset, in order to prevent the path being closer to the surface than the value of the near plane used when building panoramic views. We do this by using screen space impostors. Let r be value for the near plane. We first render the scene, then for each fragment we issue a quad with side r in world space, which will be used to raycast a sphere of the same radius in the fragment shader.

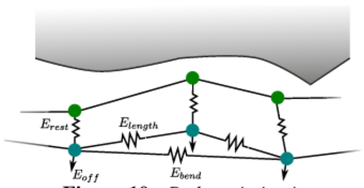


Figure 10: *Path optimization.*

The path found with this algorithm could be already used for computing the transition videos. In order to make it smoother while preventing interpenetration with the scene without drastically changing the path, we finally optimize the path by solving a energy minimization problem:

$$\min E_{rest} + E_{length} + E_{bend} + E_{off}$$

implemented with a mass-spring system. While the first three energy terms are simple springs connecting the path's nodes as shown in Fig. 10, the component E_{off} is introduced to prevent the optimized path to get too close to the surface, and it is calculated by rendering the scene from the node position, computing a repulsion force for each fragment (proportional to the inverse of square distance) and summing up to obtain a global vector force. It should be noted that we could further include a more sophisticated perceptual metric here, in order to optimize the panoramic images seen during the path. This is not currently included in our system, since we are focusing on short paths connecting probes that already meet quality constraints. Fig. 5 right shows the path graph found for the German house example.

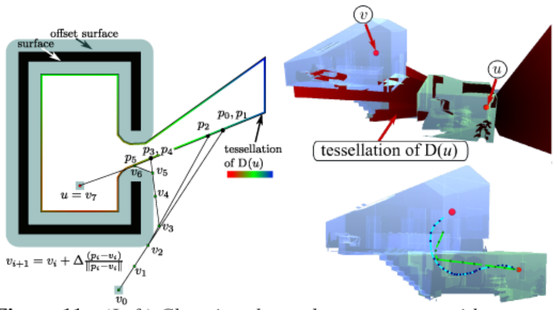


Figure 11: *(Left) Choosing the probes to connect with an arc. p is the closest point to u that is also visible from v, therefore the algorithm moves v towards p. The squares around v_i and u represent the near planes. (Right) A real case: on the left the tessellated depth map, on the right the path (green), and the smoothed path (blue).*

4.3. Creating a dataset

Once we have probes and paths, we can assemble the final dataset, which consists of one panoramic image and one thumbnail image for each probe and one panoramic video for each path. These are created on a render farm using the shaded model and an external photorealistic renderer, which renders all probes and video frames as cube maps, and all thumbnails as perspective images. Path timings are computed using a slow-in/slow-out travel strategy (in this paper, 4s for traversing the model in all presented results). Finally, an index file describing the graph structure is generated.

5. Browsing explore maps

One of the main applications of ExploreMaps is supporting ubiquitous browsing of complex illuminated models using minimal CPU/GPU resources on web-based and mobile devices. We describe here the basic features of our reference JavaScript/WebGL implementation, which can run on any WebGL-enabled web browser, as shown in Fig. 1. The view graph is exploited both to provide a visual index of the entire scene and to let users move within the environment. In automatic mode, the viewer traverses the graph by random walk. In interactive mode, the window is subdivided in three areas (see Fig. 12): the central area shows the high-res spherical panorama from the current probe position, while the right thumbnail bar (*probe bar*) shows all available probes, and the bottom context-sensitive thumbnail bar (*path bar*) shows selected probes reachable from the current position. Panoramas in the path bar correspond to the target probes of the paths leaving the current probe. They are ordered based on the angle between the current view direction and the path direction, so as to always center the probe bar on the path most aligned with the current view. Panoramas are initially oriented towards their most preferential view direction. The user is free to interactively change orientation with a dragging motion both in the central panorama and in the small panoramas appearing in the thumbnail bars. In addition, the central panorama is also zoomable. When a probe in one of the bars is selected, the path leading to it, if available, is shown in the main viewport. Clicking on the central viewport triggers a goto action. When a non-directly connected probe is selected in the probe bar, the new probe is downloaded, and presented using a cross-dissolve transition. When going to a directly connected probe, the panoramic video corresponding to the selected path is started. The view direction is then interpolated over time, during video playback, between the one at the time of clicking, which depends on the user, to the arrival one, chosen among the best precomputed view directions. This improves the quality of experience, since transitions are not repeated exactly (unless the starting position is exactly the same), and motion is consistent with the user-defined current orientation. Using precomputed video transitions with a single view direction would be too constraining, forcing the system to move the camera to the starting orientation of the video before transition, and forcing the arrival to a single fixed camera pose. Since we need a free viewpoint panoramic video, we render it by remapping frame areas on the 6 faces of the cube around the current point of view.

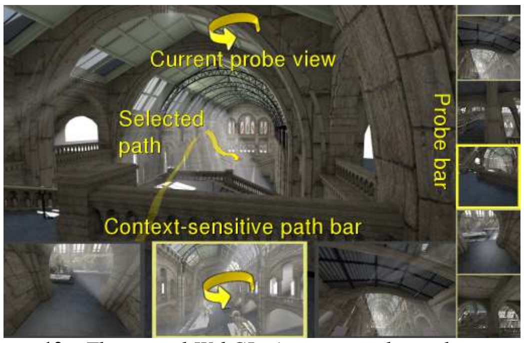


Figure 12: *The central WebGL viewer area shows the currently selected probe, while the right thumbnail bar visually indexes the scene, and the bottom thumbnail bar shows a context-sensitive subset of reachable target position.*

6. Results

The proposed method has been used to develop a complete prototype system, using C++/OpenGL for preprocessing, Blender 2.68a as external rendering engine, and Apache2 for web serving. The client application has been written in JavaScript using WebGL and HTML5. It is able to deliver results in a HTML5 canvas running in WebGL-enabled browsers (Chrome 30 in this paper).

Test models. In order to evaluate our pipeline, we downloaded several models from public repositories (Trimble 3D Warehouse, Archive3D). These websites show a few views of each model so that users can judge if they are interested in downloading it. Therefore, these sites are a perfect example of how the ExploreMaps could be used for a higher quality browsing of 3D models. Results discussed in this paper are for the models presented in Table 1, which have been selected to be representative of various model kinds, featuring complex illumination and/or geometry.

Preprocessing. Table 1 shows the performance of the preprocessing phase for the test models on a PC with Windows 7 64-bit OS, Intel i7 @3GHz CPU, 12GB RAM, equipped with a nVidia GTX 670. First of all, we ran an experiment to prove that the number of probes, clusters, and transitions highly depend on the shape of the scene and are only marginally influenced by the randomized initialization. We processed the same model 10 times and then computed the Hausdorff distance between the probes of all the pair of graphs, obtaining that the average distance between two graphs for the same model is 0.9% of the diagonal of the bounding box, the maximum distance is 10% and the number of probes varies in an interval of 10% around the expected value. As expected, for instance, a scene like Sponza needs less probes than the Medieval Town, because there are less separated spaces. The important thing, however, is that the processing time for each phase is linear with respect to the size of the elements involved. Thus, the time for Virtual Exploration is linear in the number of probes, the time for the synthesis is linear in the number of clusters, and the times for path determination and smoothing are linear in the number of paths. The number of paths is generally much higher for urban models, because it is more likely that two probes are visible from a common point, while it is smaller for closed environments like the German house. The synthesis task requires the acquisition and evaluation of nearby locations to estimate viewpoint stability, thus requiring more renderings. Also, finding and smoothing the paths are time consuming tasks, especially smoothing, which runs a local search optimization and requires 6 renderings for each evaluation of the objective function. However, note that our pipeline processed each of these models in times ranging from about 3 to a little less than an hour. It is interesting to observe that the biggest model in terms of polygon count was also one of the fastest to process. This happens because time depends on how complex the scene is, i.e., how many probes and paths do we need to see it all. Moreover, note that these phases may be easily distributed. More specifically, the synthesis only concerns a single cluster and path related operation involve pairs of probes. The dominant time of processing thus ends up being the rendering of panoramas and panoramic videos, which, as for movies, can be parallelized on a render cluster. In this paper, we used 32 8-core PCs, for rendering times ranging from 40 minutes to 7 hours/model.

Browsing. Our prototype client has been tested on a variety

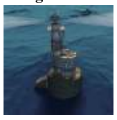
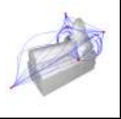
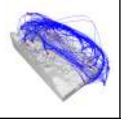
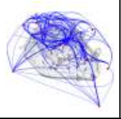
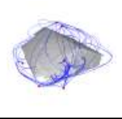
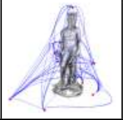
	Museum	**Sponza**	**Sibenik**	**Lighthouse**	**Lost Empire**	**Medieval Town**	**German Cottage**	**Neptune**
								
								
Input								
#tri	1,468,140	262,267	69,853	48,940	157,136	14,865	79,400	2,227,359
Output								
#probes	70	36	92	57	74	78	140	79
#clusters	17	10	21	17	25	30	23	19
#paths	127	29	58	81	206	222	102	93
Time (s)								
Exploration	154	23	63	15	41	34	163	38
Clustering	17	3	27	8	13	14	118	14
Synthesis	144	35	449	453	284	395	427	279
Path	7	1	31	12	22	80	23	13
Path smoothing	3,012	122	81	89	482	199	185	150
Thumbn.	11	3	7	5	8	10	7	6
Thumbn. pos	2	2	1	1	4	4	2	1
Total	3,347	189	659	583	854	736	925	501
Storage (MB)								
Probes	59	28	72	59	86	103	79	43
Paths	248	146	113	159	371	376	390	120

Table 1: *Selected input datasets and associated processing statistics. Probes are stored as 6x1024x1024 JPEG images, while paths are stored as 6x256x256@25fps webm videos. The numbers are for the graph instance used for the accompanying video. We verified that the number of probes, clusters, and transitions are only marginally influenced by the randomized initialization by processing the same models 10 times with different initialization. The Hausdorff distance between the probes of all the pair of graphs is* 0.9% *of the diagonal of the bounding box, the maximum distance is* 10% *and the number of probes varies in an interval of* 10% *around the expected value.*

of devices. The accompanying video shows its performance in a mobile setting, using a Nexus 4 phone (Qualcomm Snapdragon S4 Pro 4-core; 1280x768 screen) and a Acer Iconia 500 tablet (AMD Fusion C-60 and Radeon HD6290; 1280x768 screen) connected to a wireless network. Our tests demonstrate our ability to sustain interactive performance on photorealistic environments. During navigation, the user can look around within a single probe, and move to distant ones without loosing the sense of location. The frame rate during probe exploration typically exceeds 50 fps, while the frame rate during video transitions drops down to about 20 fps due to video decompression and texture updates. Even though our WebGL application is not a full-fledged viewer, it shows the potential of this automated browsing approach. While some of the models could be explorable on such a mobile device in full 3D, this could definitely not be done while presenting the same quality images (see, for instance, volumetric illumination effects in the Museum example of Fig. 1). The accompanying video also illustrates the definite advantage of using thumbnails for quick scene browsing and panoramic videos for transitions with free selection of both the starting orientation and the arrival one.

7. Conclusions

Our main contribution is an automated GPU-accelerated technique for transforming general renderable scenes into a panoramic view graph representation, exploited for creating automatic scene indexes and movie previews of complex scenes, as well as for supporting interactive exploration through a low-DOF assisted navigation interface. Real-time performance is achieved on WebGL-based environments even on low-powered mobile devices.

Admittedly, there are still a few limitations that need to be addressed. On the construction side, the sampling rate (that is, the viewport size of the cameras during the exploration phase), is fixed and inferred from the scene bounding box. For models where there are interesting details at very different scales (teapot in a stadium problem), the sampling rate should be made adaptive. Furthermore, shading information is not taken into account in the placement of probes, while it could be important for certain models, especially for the selection of best views. Currently, the proposed system considers a binary visibility model: if a surface exists, and is not fully transparent, it acts as a blocker. Semi-transparency handling could be included by considering the degree of opacity of a path during the path construction phase, penalizing paths that go through semitransparent surfaces with respect to paths going through free space. Domain-specific knowledge could also be incorporated in the system, e.g., to achieve human-scale exploration in architectural models. The incremental nature of our probe placement technique should also be exploited to let users optionally provide user-defined set of probes, e.g., to force inclusion of semantically relevant/nice shots in the graph. On the browsing side, our implementation aims at being a proof of concept, and many improvements are possible. In particular, there is no way to aim at a point of interest and move to the probe that has a best view of it, which may be frustrating for the user. Moreover, the view graphs should also be exploited to provide location awareness through the automatic generation of overhead views.

There are many potential applications for the ExploreMaps. In particular, we aim to open the way to a richer experience in presenting 3D models on dedicated web sites, no more limited to few still images or very constrained orbiting interfaces. Furthermore we can turn construction CAD into navigable previews for presentation to stakeholders/potential owners. Thanks to our unattended processing pipeline, we envisage the implementation of a public web service allowing users to upload 3D models and make them browsable in WebGL-enabled browsers, making for 3D models what Flickr/YouTube made for images and videos.

Acknowledgments. This work is partially supported by the EU FP7 Program under the DIVA (290277) and VASCO (607737) projects. We also acknowledge the contribution of Sardinian Regional Authorities. We thank Roberto Combet for blender wizardry. See accompanying video for model credits.

References

[ABB*07] Andujar C., Boo J., Brunet P., Fairén M., Navazo I., Vázquez P., Vinacua A.: Omni-directional relief impostors. *Computer Graphics Forum 26*, 3 (Sept. 2007), 553–560. 3

[Abi95] Abidi B.: Automatic sensor placement. In *Proc. Intelligent Robots and Computer Vision* (1995), pp. 387–398. 3

[BGM*12] Balsa Rodriguez M., Gobbetti E., Marton F., Pintus R., Pintore G., Tinti A.: Interactive exploration of gigantic point clouds on mobile devices. In *Proc. VAST* (2012), pp. 57–64. 2

[BGMT13] Balsa Rodriguez M., Gobbetti E., Marton F., Tinti A.: Compression-domain seamless multiresolution visualization of gigantic meshes on mobile devices. In *Proc. ACM Web3D* (2013), pp. 99–107. 2

[Che95] Chen S.: Quicktime VR: An image-based approach to virtual environment navigation. In *Proc. SIGGRAPH* (1995), pp. 29–38. 2

[CM02] Comaniciu D., Meer P.: Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 5 (2002), 603–619. 5

[CON08] Christie M., Olivier P., Normand J.-M.: Camera control in computer graphics. *Computer Graphics Forum 27*, 8 (2008). 1, 2

[CPAM08] Capin T., Pulli K., Akenine-Moller T.: The state of the art in mobile graphics research. *IEEE Computer Graphics and Applications 28*, 4 (2008), 74–84. 2

[DGY07] Dietrich A., Gobbetti E., Yoon S.: Massive-model rendering techniques: A tutorial. *IEEE Computer Graphics and Applications 27*, 6 (nov/dec 2007), 20–34. 2

[FCODS08] Fu H., Cohen-Or D., Dror G., Sheffer A.: Upright orientation of man-made objects. In *ACM Trans. Graph.* (2008), vol. 27, p. 42. 3

[FCOL00] Fleishman S., Cohen-Or D., Lischinski D.: Automatic camera placement for image-based modeling. *Computer Graphics Forum 19*, 2 (2000), 101–110. 3

[GKY08] Gobbetti E., Kasik D., Yoon S.: Technical strategies for massive model visualization. In *Proc. Solid and Physical Modeling Symposium* (2008), pp. 405–415. 2

[GMB*12] Gobbetti E., Marton F., Balsa Rodriguez M., Ganovelli F., Di Benedetto M.: Adaptive Quad Patches: an adaptive regular structure for web distribution and adaptive rendering of 3D models. In *Proc. ACM Web3D* (2012), pp. 9–16. 2

[JWL12] Jin Y., Wu Q., Liu L.: Unsupervised upright orientation of man-made models. *Graphical Models* (2012). 3

[KCSC10] Kopf J., Chen B., Szeliski R., Cohen M.: Street slide: browsing street level imagery. In *Proc. SIGGRAPH* (2010), pp. 96:1–96:8. 2

[KFL01] Kimber D., Foote J., Lertsithichai S.: Flyabout: spatially indexed panoramic video. In *Proc. ACM Multimedia* (2001), pp. 339–347. 2

[Lip80] Lippman A.: Movie-maps: An application of the optical videodisc to computer graphics. *Proc. SIGGRAPH 14* (July 1980), 32–42. 2

[Llo82] Lloyd S.: Least squares quantization in pcm. *IEEE Trans. Inf. Theory 28*, 2 (mar 1982), 129 – 137. 4

[MLL*10] Maglo A., Lee H., Lavoué G., Mouton C., Hudelot C., Dupont F.: Remote scientific visualization of progressive 3D meshes with X3D. In *Proc. ACM Web3D* (2010), pp. 109–116. 2

[NKB10] Niebling F., Kopecki A., Becker M.: Collaborative steering and post-processing of simulations on HPC resources: Everyone, anytime, anywhere. In *Proc. ACM Web3D* (2010), pp. 101–108. 2

[Rap98] Rappoport D.: Image-based rendering for non-diffuse synthetic scenes. In *Proc. Rendering techniques* (1998), p. 301. 3

[SCK07] Shum H., Chan S., Kang S.: *Image-based rendering*. 2007. 2

[SKG*12] Sinha S., Kopf J., Goesele M., Scharstein D., Szeliski R.: Image-based rendering for scenes with reflections. *ACM Trans. Graph. 31*, 4 (2012), 100. 2

[SLF*11] Secord A., Lu J., Finkelstein A., Singh M., Nealen A.: Perceptual models of viewpoint preference. *ACM Trans. Graph. 30*, 5 (Oct. 2011). 2, 4

[SRB06] Safro I., Ron D., Brandt A.: Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms 60*, 1 (2006), 24–41. 5

[SRR03] Scott W. R., Roth G., Rivest J.-F.: View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv. 35*, 1 (Mar. 2003), 64–96. 2, 3

[SS12] Sankar A., Seitz S.: Capturing indoor scenes with smartphones. In *Proc. UIST* (2012), pp. 403–412. 2

[SSS06] Snavely N., Seitz S. M., Szeliski R.: Photo tourism: exploring photo collections in 3d. In *Proc. SIGGRAPH* (2006), pp. 835–846. 2

[TKKT12] Tompkin J., Kim K. I., Kautz J., Theobalt C.: Videoscapes: exploring sparse, unstructured video collections. *ACM Trans. Graph. 31*, 4 (2012), 68. 2

[VD08] Van Dongen S.: Graph clustering via a discrete uncoupling process. *SIAM Journal on Matrix Analysis and Applications 30*, 1 (2008), 121–141. 4

[VFSH02] Vazquez P.-P., Feixas M., Sbert M., Heidrich W.: Image-based modeling using viewpoint entropy. In *Proc. CGI* (2002). 3

[Vin07] Vincent L.: Taking online maps down to street level. *Computer 40* (December 2007), 118–120. 2

[WDH*06] Wenhardt S., Deutsch B., Hornegger J., Niemann H., Denzler J.: An information theoretic approach for next best view planning in 3-d reconstruction. In *Proc. ICPR* (2006), pp. 103–106. 3

[WM03] Wilson A., Manocha D.: Simplifying complex environments using incremental textured depth meshes. *ACM Trans. Graph. 22*, 3 (2003), 678–688. 3

[YGKM08] Yoon S., Gobbetti E., Kasik D., Manocha D.: *Real-time Massive Model Rendering*, vol. 2 of *Synthesis Lectures on Computer Graphics and Animation*. August 2008. 2