STAG: Smart Tools and Apps in computer Graphics (2016)
Giovanni Pintore and Filippo Stanco (Editors)

Generalized Trackball for Surfing Over Surfaces

Luigi Malomo[1,2], Paolo Cignoni[1], Roberto Scopigno[1]

[1]Visual Computing Lab, ISTI - CNR, Italy
[2]University of Pisa, Italy

Abstract

We present an efficient 3D interaction technique: generalizing the well known trackball approach, this technique unifies and blends the two common interaction mechanisms known as panning and orbiting. The approach allows to inspect a virtual object by navigating over its surrounding space, remaining at a chosen distance and performing an automatic panning over its surface. This generalized trackball allows an intuitive navigation of topologically complex shapes, enabling unexperienced users to visit hard-to-reach parts better and faster than with standard GUI components. The approach is based on the construction of multiple smooth approximations of the model under inspection; at rendering time, it constrains the camera to stay at a given distance to these approximations. The approach requires negligible preprocessing and memory overhead and works well for both mouse-based and touch interfaces. An informal user study confirms the impact of the proposed technique.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

Interacting with a 3D environment has always been a very common and basic task in 3D graphics and many different solutions are described in literature and used in interactive applications. The *trackball/arcball* tool is one of the most common solutions for inspecting 3D models by controlling a virtual camera. The implementation of this tool is present in almost all 3D interactive systems and it offers orbit operation to rotate around a pivot point, combined with pan and zoom operations. This approach provides to both experienced and novice users a powerful instrument to drive the visualization of 3D models. Even if very successful, this approach presents some flaws. By design this tool focuses the analysis on an exocentric inspection task, so, after a series of orbit+pan operations, the user can get *lost*. This happens because the trackball pivot (the camera look-at point) can be located considerably distant from the virtual object surface under inspection so subsequent rotations move the virtual camera in unexpected positions. This issue is usually patched by introducing the focus operation: experienced users know when to reset the trackball pivot to a convenient location, but a novice may found the circumstances extremely disorienting. Moreover, in an ideal inspection session the user should be able to drive the camera to obtain an automatic fly over its surface. Using the standard trackball this kind of inspection requires the user to continuously perform a series of pan, orbit, focus and zoom operations:

- *pan* actions allow to move the camera in the xy-plane to frame another portion of the object;
- then, *orbit* the camera to look at the surface with a comfortable angle;
- then, perform a *focus* operation to *retarget* the region of interest;
- finally, *zoom* to adjust the distance from the object, depending on the level of detail required in the inspection.

This sequence of operation is usually repeated several times, which breaks the continuity of the inspection process. Moreover, in the case of topologically complex objects presenting many cavities and protrusions, this approach can often take the camera eye to cross the object surface, resulting in an undesirable effect.

These issues occur because the interaction interface is not *environment-aware*: the standard trackball moves and reorients the virtual camera, without taking into account the specific object shape. We propose an innovative manipulation technique called **Generalized Trackball**. We introduce additional constraints which limit the camera freedom but, at the same time, avoid the drawbacks reported above and provide an improved user experience. Our main goal is to drive the camera to keep the virtual object surface always framed and at a fixed distance. Orbit, pan and focus operation are all merged in single panning operation that smoothly moves the camera as if it is hovering above the surface (similarly to the HoverCam solution [KKS*05]). Our technique allows unexperienced users to explore with accuracy every detail of a generic 3D model, relying on an input interface with just two degrees of freedom. Moreover, another important constraint in our design has been the need to reduce as much as possible the resources requirements (memory and processing). In particular we do not want to force the application to run an extensive and costly preprocessing of the mesh to be rendered, because one of our design goal was to develop a solution

easily portable on low-performance mobile systems (smartphones or tablets).

We started from the same inspiration principles of the HoverCam paper, but the technique presented here allows to go beyond the limitations of that previous solution and proved to offer an efficient free navigation control. More precisely, the contributions of this paper include:

- a clean formalization of the behavior of the camera in the space surrounding the object based on the definition of Curvature Bounded Surface;
- a mesh-less adaptive multi resolution representation of the surface that allows to efficiently compute an approximation of the Curvature Bounded Surface (Section 3.1) that leads to smooth camera movements;

In the following sections we give a brief overview of the state of the art, introduce our approach and present a first naive implementation solution. Then we formalize the problem, clarify our constraints in terms of use of resources and performances, and finally explore the implementation alternatives, focusing on challenging aspects. Next we describe the implementation details of our solution and finally we present the results and discuss limitations and possible improvements.

2. Related Work

The purpose of this section is not to provide a comprehensive survey over 3D camera control techniques. For a full review of current state of the art refer to [CON08, JH13].
The problem of interactive 3D camera control mainly consists of mapping inputs with limited degrees of freedom (DOF) to higher dimension transformations, matching the user control scheme to the perceptual feedback of the interface. The main solution developed to interactively observe and inspect a 3D model is the virtual trackball [CMS88, Bel88, Sho92, HSH04]. The trackball traditionally allows 3D rotations with just 2 DOFs input devices, such as the mouse. A state of the art implementation can be found in the open-source tool MeshLab [CCR08].
The safe 3D navigation technique [FMM*08] has been designed to overcome the lost-in-space effect of the standard trackball tools. It has explicit pivot indication, avoids camera-to-surface collision and provides alternative egocentric look-around metaphor for indoor environment visualization.
Other solutions provide different interaction metaphors (see [WO90]). Hachet et al. [HDKG08] realized a point-of-interest technique that allows to directly specify camera target location with a simple pointing interface and then adjust, using an overlaying widget, the desired viewing angle. That results in the camera moving from the current shot to the newly specified one.
Solutions for the exploration of complex scenes (e.g. buildings interiors) require to precalculate possible paths, e.g., by performing a cell-and-portal subdivision, created using a distance field, and supporting the creation of camera paths for scene walkthrough [AVF04]. Other techniques are targeted to virtual environment exploration and, using a traditional flying vehicle interface, avoid collisions with the surrounding geometry constraining the camera with force-fields [WDK01, MMGK09].
Viewcube [KMF*08] proposes a 3D widget overlaid onto the scene that easily allows to transform the camera to view from each one of the 27 canonical directions (top, front, left, etc.).
Another general trend is to build interaction techniques that reduces the DOFs with respect to complete free camera movement and maps simple input operation to a combination of low level transformation to comply to the specific application. This idea is used in [LC15] to allow the user to easily control a camera for cinematographic purposes. Other approaches limit the camera DOFs to provide a simpler and most effective object inspection. Rodriguez et al. [RAMG14] proposed an auto-centering virtual trackball based on real-time screen space stochastic sampling. Marton et al. [MAG*12] further reduce the DOFs by introducing a focus sliding interaction metaphor with automatic zooming. Hanson et al. [HW97] constrain virtual camera position to a patch surface that offers convenient environment exploration and/or object inspection using 2-DOFs interaction. Each keynode of the patch has a set of precomputed or authored camera properties associated (up vector, orientation, focal length, etc.) that interpolates within the surface. Other solutions provide authored views [BKF*02, BKFK06], adding some limitation to the effective view selection.

Recently two other papers have been published on this subject [MRB*14] and [Bou14]; both of them target the main idea of a camera orbiting around the scene surface, but they adopt a screen space approach that builds partial local approximation of the visible scene. On the other hand our techniques is based on a global approach that builds up a multi resolution representation that allows to consider also global, non visible, portions of the scene to determine the camera paths in a more robust and continuous way.

The HoverCam approach [KKS*05] is targeted to exocentric inspection task and is the technique most similar to our solution. HoverCam allows to interactively navigate a model with a hovercraft metaphor: the user can move the camera as if it's hovering above the surface, while staying at a fixed distance from the closest point on the model and looking at it. But it adopts local computation to avoid the known distance field drawbacks. As briefly stated in the introduction, we follow a similar approach, but our design is more flexible and cheaper in terms of computational resources (because we start with an accurate mathematical model and then derive an approximate implementation that accomplishes the prescribed behavior and the performances), as better demonstrated in the following. Moreover our approach, being based on smooth approximation of the original surface, does not suffer of the jerkyness in the camera movement that affected HoverCam, please compare the accompanying video with the ones of [MMGK09].

3. Defining the Generalized Trackball

Inspecting an almost spherical object using orbit and zoom operations of a standard trackball is very intuitive. Orbiting the trackball allows the camera to focus on any surface region while the zooming operation controls how close to look at it. Starting from this trackball interaction scheme we can provide an alternative interpretation for the manipulation technique, involving two parametric surfaces, camera eye and camera target surfaces. We define S_E as the spherical surface with radius r_E which is the space of possible camera eye points E. S_T as the spherical surface of radius r_T of possible camera

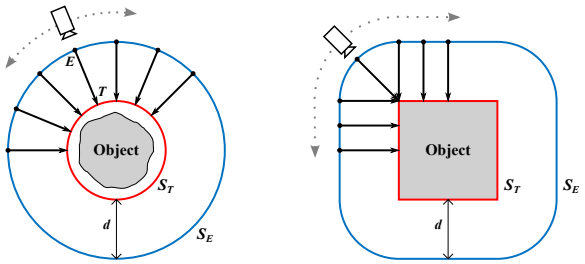


Figure 1: *The standard trackball allows to orbit around an object on a spherical surface. We can generalize this idea by forcing the camera to move over a surface closer to the original shape of the object. In blue the surface of possible camera eye positions. In red the surface of the corresponding camera targets.*

target points T, that wraps the 3D model (see Fig. 1 left). We also define a one-to-one correspondence between each point E and the associated target T, obtaining a function f that for each point on S_E returns one point on S_T. For every camera position the distance between E and T is $d = r_E - r_T$. We can then redefine the orbit operation: displacement of screen-space input maps to movement of eye point E over the surface S_E; as point E changes, the target is set to $T = f(E)$. Zooming operation changes the radius r_E of surface S_E, thus modifying the distance d between eye and target.

This interpretation is easy to formalize for spherical surfaces. Our goal is to extend this design to arbitrarily complex objects: this means that we have to define a generic surface S_T that optimally approximates the model shape, a new space of possible eye points S_E and a function that maps E to T. A simple and straightforward solution consists in defining S_T equal to the 3D model surface (see Fig. 1 right). Then we define S_E as the offset surface of distance d for the 3D model (the locus of points whose minimum distance from the model geometry is d). The new f maps each point E to the closest point on the surface T. With this solution every eye-target vector $\vec{ET}$ is normal to both S_T and S_E surfaces (when normal is defined). This technique is ideal for convex shapes but for generic 3D models presenting concavities it does not work.

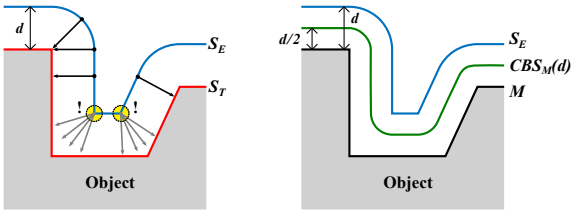


Figure 2: *Left: Naive trackball generalization for a concave object surface. Right: Curvature-Bounded Surface (green) generated for the same surface.*

3.1. Curvature-Bounded Surface

Considering the simple concave shape of Fig. 2 we can notice that the surface S_E presents some *singularities* (the highlighted regions) where the normal is undefined and there is no obvious mapping from the camera eye point to a camera target on S_T. The HoverCam technique [KKS*05] solves this issue using an algorithmic solution that is based on local input movement and geometric calculations. We propose instead a formal approach relying on a well defined surface that is C^1 and bounded in curvature: the **Curvature-Bounded Surface** (CBS), defined as

$$CBS_M(d) = offset(-\frac{d}{2}, offset(d, M))$$

where d is the distance and M is the 3D model surface. The *offset* function builds the surface defined by all the points at distance d from M (a distance field isosurface). Using the Mathematical Morphology terminology [Ser83, LL88] our CBS can be obtained from M with a *dilate* operation of value d followed by an *erode* operation of $\frac{d}{2}$ (see Fig. 2, right).

The CBS has a lot of useful properties. First and most importantly the CBS for a surface M at distance d has the curvature upper-bounded by the value $c = \frac{2}{d}$ since the curvature radius cannot be less than $\frac{d}{2}$. This implies that the surface $\in C^1$. The continuity is obvious since the CBS is an offset surface. It is derivable because our CBS has upper-bounded curvature. As a consequence every point on the CBS has a defined normal.

Given the parametric surface $CBS_M(d)$, a point P on it and his associated normal $\vec{N}_P$, the point

$$P' = P + \vec{N}_P \cdot \frac{d}{2} \tag{1}$$

lies on the offset surface of M with distance d. Moreover, it can be proved that P' is the point on the isosurface closest from P.

Our *Generalized Trackball* (GT) is thus defined by means of the CBS. Given a 3D mesh model with surface M we define the space of possible camera eye point as $S_E = offset(d, M)$ and the surface of associated target points as $S_T = CBS_M(d)$. With this technique we map screen-space input movement to displacement of target point T over S_T, instead of E. When T changes, we retrieve the new eye point E using the Equation 1 which becomes:

$$E = f^{-1}(T) = T + \vec{N}_T \cdot \frac{d}{2} \tag{2}$$

Here, the f proposed in the trivial solution is inverted to make it a proper function defined for all values $T \in S_T$.

We also know that given a point P and a model surface M, $\exists\ \bar{d} \mid P \in CBS_M(\bar{d})$. Accordingly we can define a *CBS field* that for every point returns the value $\bar{d}$ of the corresponding CBS surface.

We can now provide an algorithm for the GT. The generalized orbit operation (pan) moves the target T over the corresponding CBS of distance d. With the zoom operation, we change the value of d to d'. To find the corresponding $T' \in CBS_M(d')$, we perform an ascent (or descent) along the CBS field gradient until we found the value d' and thus the point T'. For each new value of T, we compute the corresponding eye point E using the Equation 2 (see Fig. 3).

4. Data structures for efficient implementation

The GT formalized so far provides an exocentric camera manipulation technique extremely robust and coherent. But we should take into account constraints for its actual implementation, like the real-time performance requirement.
Implementing a volumetric field with realtime performance is a challenging task, even using state of the art algorithms and spatial indexing structures (e.g., [LH07]). Starting from a model surface, one idea is to generate a series of offset surfaces and compute their corresponding CBS. Then we could take a number of samples from each of the isosurfaces and set the obtained values into an optimized structure that will become our CBS field. Unfortunately, this kind of structures are not optimal for our purposes, since the field gradient (isosurface normals) would present noise. For visual representations (shaded surface), slightly noisy normals can be tolerated, but when they are used as view directions even subtle variations can result in significant interaction disruption. In addition, we must consider that such data structures, even if compressed, occupy a substantial amount of memory and their use would require a time-consuming preprocessing phase.
To address performance requirements we adopted a new strategy. We observed that an exact implementation of the formalized mathematical model is not needed; the only requirement is to provide visual continuity during the inspection process.

4.1. KD-tree distance field and normal field

For each point in space we need the distance from the model surface M and a normal directed to the closest region. The key idea for an efficient implementation is to have an approximate distance field and a continuous vector field of normals that is quite coherent with the former. We realized both using an auxiliary data structure that relies on a KD-tree.

Given a 3D model, a preprocessing phase creates a point cloud PC_r that approximates its surface with uniformly distributed point samples (Fig. 4). The point generation is performed with a fast poisson-disk sampling strategy, which ensures that the distance between any two points is greater than a given sample radius r [CCS12]. This preprocessing time is negligible with respect to the file loading time.
The KD-tree built over PC_r is extended to approximate a distance field with an implicit representation, thus let us call it **KD-tree distance field**. For every point in space, we can perform a nearest-neighbor query (*NN*) on the KD-tree, get the closest point within PC_r, and return the distance from it. This approach is really fast, but the point sampling discretization produces many *singularities* on the isosurface produced (Fig. 5). The latter are also present in the case of concave surface regions (Fig. 2).
To eliminate this issue, we propose an alternative distance field approximation that *smooths* all singularities and produces almost C^1 isosurfaces. Instead of getting the closest point within the PC_r we perform a k-nearest-neighbors (k-*NN*) query to get k closest sample points. Field values are computed by averaging the distances d_i from each of the k points with a gaussian based method. Precisely:

$$d = \sqrt{\frac{\sum_{i=1}^{k} \omega_i \cdot d_i^2}{\sum_{i=1}^{k} \omega_i}} \quad \text{with} \quad \omega_i = e^{-g \cdot \left(\left(\frac{d_i}{d_{\min}} \right)^2 - 1 \right)}$$

where ω_i is the gaussian-like weight for each of the k distances, $d_{\min}$ is the minimum distance among all distance values, and g is the scale factor used to tweak the gaussian weighting.

Experimentally we found that $k = 16$ provides a good trade-off between performance and smoothness. Fig. 6 shows a section of an isosurface generated with the KD-tree distance field using *16-NN*

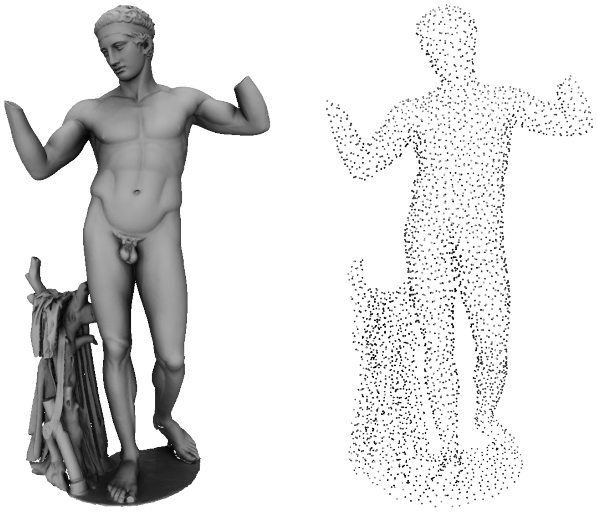


Figure 4: *Poisson-disk sampling of a 3D model.*

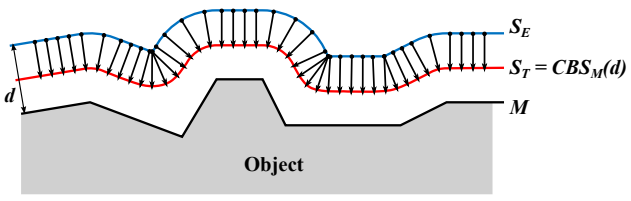


Figure 3: *Camera eye-target vectors for the Generalized Trackball. The CBS is the space of possible targets.*

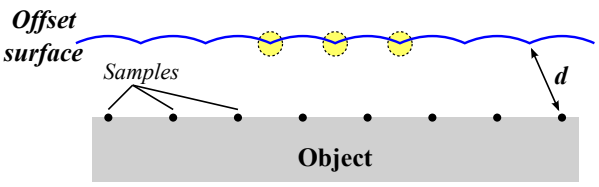


Figure 5: *Singularities (in yellow) on the offset surface generated from a KD-tree distance field.*

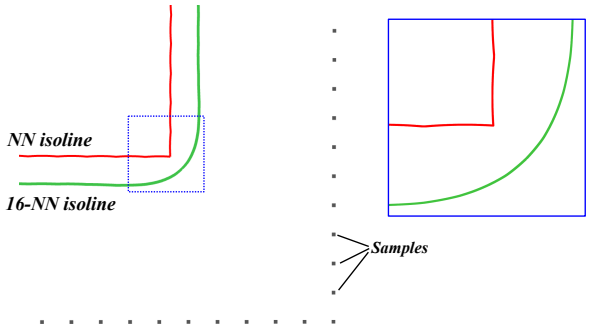


Figure 6: *Isosurface sections generated with NN (red) and 16-NN queries (green).*

queries, compared to the same obtained with simple *NN* procedure.

Similarly, we implemented a **KD-tree normal field** that approximates the gradient of the distance function. To be exact, we did not provide a vector field consistent with the distance field gradient, but we created an *ad-hoc* continuous function that returns for each point a vector directed towards the closest region of the PC_r. In practice, we take the k closest points from the query point Q, we average them obtaining P_{avg}, and finally we yield the normalized vector $\vec{P_{\text{avg}}Q}$. The function that computes the average is the following:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{k} \hat{\omega}_i \cdot P_i}{\sum_{i=1}^{k} \hat{\omega}_i}$$

For normal calculation we couldn't use the same weighting strategy with respect to distance field. Since each of the k points contributes to normal evaluation, the most distant point must have a null contribution. That's because, varying the query point Q, the *k-NN* points changes discretely: in particular the most distant point is dropped in favor of a new different one. In these cases we obtain normal discontinuities, since a point is exchanged in the normal evaluation function, while his weight and all other terms remain the same. Weights $\hat{\omega}_i$ need to be different to solve this issue:

$$\hat{\omega}_i = 1 - \left(\frac{d_i^2 - d_{\min}^2}{d_{\max}^2 - d_{\min}^2} \right)$$

where $d_{\max}$ is the maximum distance among the k distance values.

4.2. Approximated CBS

The approximate distance and normal fields presented above allow to efficiently implement our approach based on CBS. Let us remember that the camera eye space is the distance field isosurface with distance d, which in our case is:

$$S_{E_d} = \{E \mid D_{\text{field}}(E) = d\}$$

with D_{field} being the KD-tree distance field function. The camera target space is then defined by an approximate version of the Curvature Bounded Surface $\widetilde{CBS}(d)$ defined it in terms of our distance and normal field:

$$S_{T_d} = \widetilde{CBS}(d) = \{P \mid P = Q - \vec{N_{\text{field}}}(Q) \cdot \frac{d}{2} \text{ and } Q \in S_{E_d}\}$$

with $\vec{N_{\text{field}}}$ being the KD-tree normal field function. Since we eliminated the *singularity* issue with the approximate fields implementation, we can now have for the GT a proper function $f : \mathbb{R}^3 \to \mathbb{R}^3$ that maps each camera eye point in a corresponding target point:

$$f(E) = T = E - \vec{N_{\text{field}}}(Q) \cdot \frac{D_{\text{field}}(E)}{2}$$

5. Interaction

In the following we describe the implementation of the GT base operations.

Panning. The *pan* operation moves the camera remaining at a fixed distance d from the object surface, i.e. the camera is *floating* above the 3D mesh and looking always at the surface. Given a camera eye point $E \in S_{E_d}$ and a corresponding target $T = f(E)$, for a mouse based (or touch) drag input we want to obtain a new eye point $E' \in S_{E_d}$ and target $T' = f(E')$ such that the vector $\vec{TT'}$ is *consistent* with the input displacement vector.
We can't use the input to displace E on the S_{E_d} isosurface because small movements in the regions corresponding to 3D model concavities will result in the camera changing abruptly its orientation, producing an evident visual discontinuity.
Since there is no direct method that given a displacement vector allows to move the point T along the implicit CBS surface S_{T_d}, we developed an iterative algorithm to find the new eye point E' that maps to the desired camera target $T' = f(E')$.

Due to performance reason, we cannot have a mesh representation of the isosurface S_{E_d}, so we need an operator that allows to retrieve points on it using only a few distance field queries. Therefore, we introduced a *distance field ray-casting operator* that performs a linear search within the field until the d value is found. For an exact distance field the implementation is easy. To find a point at distance $\bar{d}$ starting from a ray with origin P_0 and direction $\vec{D}$, the algorithm evaluates $d = D_{\text{field}}(P_i)$; if $\left|\frac{\bar{d}-d}{\bar{d}}\right| < \varepsilon$ the P_i point is returned. Else,

$$P_{i+1} = P_i + \vec{D} \cdot \left|\bar{d} - d\right| \tag{3}$$

is computed and a new iteration is performed. If after a fixed number of iterations the relative error keeps increasing, the ray casting operation is assumed to be divergent and is then stopped. This algorithm is conservative: it does not use derivatives of the field and always finds, if present, the first point in ray direction that evaluates to $\bar{d}$. The algorithm works exploiting the triangular inequality property of the distance field, which is not guaranteed in our KD-tree implementation. For this reason our ray casting operation is different and only possible from lower to higher distance values ($D_{\text{field}}(P_0) < \bar{d}$). We changed the advance factor of the linear search: the step of Equation 3 becomes

$$P_{i+1} = P_i + \vec{D} \cdot \left|\bar{d} - d\right| \cdot \frac{D_{\min}(P_i)}{d}$$

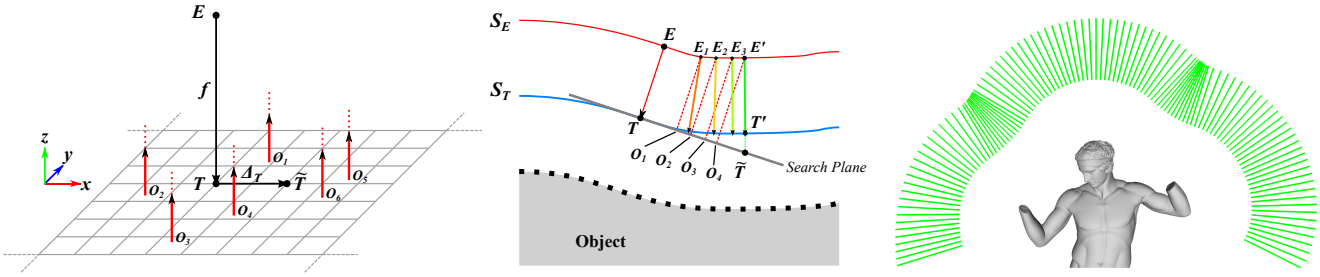


Figure 7: *Left: the space of points O_i used to perform ray casting operations. Center: simplified 2D section for the searching algorithm. Right: series of camera eye-target vectors generated with fixed step panning operations along one direction.*

where $D_{\min}$ is the distance field value computed using single nearest neighbor query. This solution accounts for field *compression*: it rescales the increment for search point along ray direction exploiting the fact that compression factor of the field decreases with growing values of d. Let's call this newly introduced operation $ray(\text{origin}, \text{direction}, d)$, where d is the distance of the target isosurface to be intersected.

We discuss GT mouse based interactions; extension to touch interfaces is rather straightforward. For simplicity's sake, assume to start from a situation in which we have a camera point E, a corresponding target $T = f(E)$ and $D_{\text{field}}(E) = d$. With a mouse drag movement the user translates the cursor from a screen-space point P_1 to another P_2. We want to translate the target T to a position $\tilde{T}$ such that $\vec{\Delta_T} = \tilde{T} - T$ is *consistent* with screen-space translation $\vec{\Delta_P} = P_2 - P_1$.

Let's fix the X and Y axes to be always aligned to screen-space x and y axes (the z axis will result identical for both world- and screen-space and it is directed from the screen surface towards the user). For an orthogonal projection the vectors are basically the same (only scaling factor between screen-space and world space coordinates must be considered). For a perspective projection instead, considering screen-space coordinates $x \in [-\text{aspect ratio}, +\text{aspect ratio}]$ and $y \in [-1, +1]$, the translation vector is computed as:

$$\vec{\Delta_T} = \vec{\Delta_P} \cdot \frac{d}{2} \cdot tan\left(\frac{\text{FOV}_y}{2}\right)$$

where FOV_y is the field of view vertical angle of the camera and $\frac{d}{2}$ is the distance between E and T. With this computation, the screen-space translation is scaled to achieve a perceptually coherent world-space displacement located on the plane orthogonal to view direction and passing through the camera target T. The problem is that $\tilde{T}$ is not guaranteed to be on the surface S_{T_d} and also we cannot invert the f function to retrieve the corresponding new eye point. The final and most important step is therefore an optimized space research algorithm that finds the desired eye point $E' \in S_{E_d}$ such that $\|\tilde{T} - (E' - d \cdot \vec{N_{\text{field}}}(E'))\|_2$ is minimum (i.e. the point E' for which its corresponding target point is the closest to $\tilde{T}$). We take the plane orthogonal to view direction that includes both T and $\tilde{T}$ and create on it a space of points defined as $O_i = T + \vec{\Delta_i}$, where $\vec{\Delta_i} = (x, y, 0)^T$ is a space of bidimensional displacement vectors. For every point on the plane we can perform a ray casting operation to obtain a point on S_{E_d} isosurface:

$$E_i = ray(O_i, \vec{N_{\text{field}}}(E), d)$$

Then, given the candidate E_i, we can obtain the corresponding target $T_i = f(E_i)$. Now we can transform the space-search into a minimization problem:

$$\min_i(\|\tilde{T} - T_i\|) \quad \text{where} \quad T_i = f(ray(O_i, \vec{N_{\text{field}}}(E), d))$$

The minimization problem was solved with the NEWUOA software [Pow06]. From these results we can retrieve the new eye point $E' = ray(T + (x, y, 0)^T, \vec{N_{\text{field}}}(E), d)$ and the new target $T' = f(E')$ (Fig. 7).

A situation that may reduce the algorithm precision or definitely break it occurs when the input displacement is excessive, e.g., when the user performs a very fast mouse move. We overcome this issue by splitting large movements in a series of length-bounded displacements, sequentially launching the procedure for each one of them.

Zooming. The idea behind the zoom operation is actually very simple: the distance d defining the distance from the model surface is varied; consequently, camera eye and target points are moved to the newly obtained parametric surfaces S_{E_d} and S_{T_d}. Similarly to a standard trackball implementation, the zoom is geometric and the d value is changed with the mouse wheel notch to the new value d'. The new camera points E' and T' are computed using with the same procedure used by the panning algorithm. The only difference is that reference target point $\tilde{T}$ is equal to T and the ray casting operation is performed using the distance d' to retrieve intersection points E_i on the isosurface $S_{E_{d'}}$.
Zoom operation, like panning, works only *locally*: when the d value changes too much, the iterative procedure could not converge to the correct new camera position. Analogously to the panning case, we split distance displacement into a sequence of smaller operations.

Z-Rotation. During a panning operation, while moving along a curved surface, the view direction vector $\vec{ET}$ may vary significantly. As already noted in [KKS*05], these direction changes require an up-vector policy to be defined. In general in our technique we used the local up-vector mode, in which screen-space displacement vector tries to always match world-space translation along camera movement, resulting in up-vector drifting due to this con-

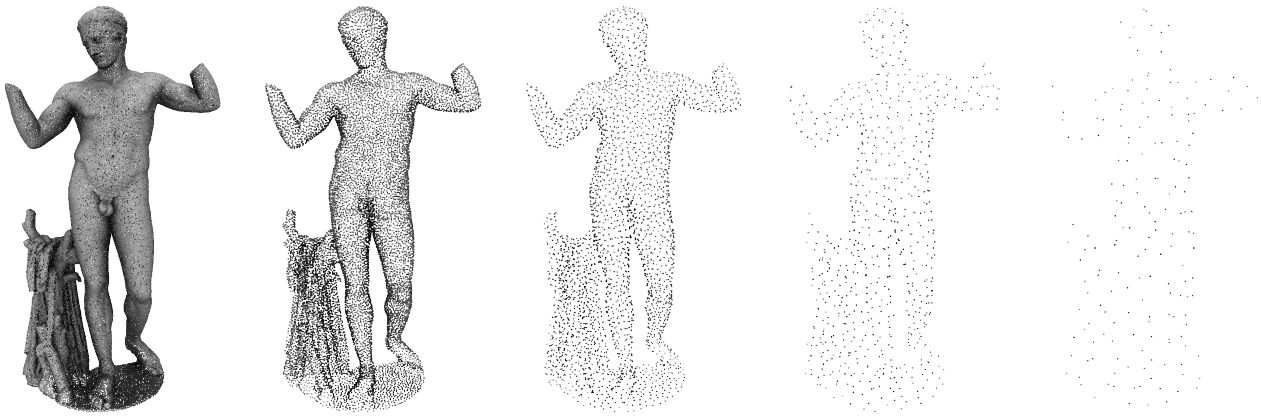


Figure 8: *Point clouds resulting from poisson-disk sampling using subsequently halved radii.*

strain. For models that have an obvious up-orientation we switch to the global up-vector policy.

6. Scale dependency

For close object inspection the method delivers exactly the expected behavior: it is extremely intuitive and produces smooth camera movements perceptually coherent with the user interaction. If we focus on far object inspection (e.g., when the entire object is fully framed), the expected interaction should be very similar to the standard trackball (the camera eye moving over a *quasi* spherical surface).

The competition between these two behaviors results in a *shakiness* effect, that has been noted as well for the HoverCam implementation and may show up also at medium distances. Moreover, at great distances the KD-tree queries become slower because the nearest-neighbor algorithm must visit a consistent portion of the tree.

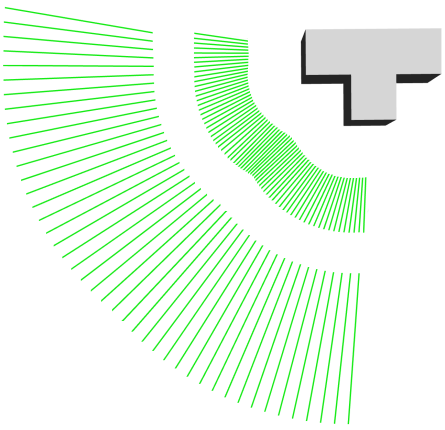
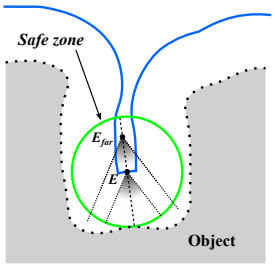


Figure 9: *Eye-target camera vectors at different distances obtained using scale dependent field approximation.*

To solve this issue we introduced a modification that copes with both the performance issue and the disturbing camera movement for distant observation. The distance field is represented at different scales, by means of scale-dependent KD-trees, constructed starting from different *poisson-sampled* mesh surfaces, each one representing a given scale (by iteratively halfening the sampling radius $r_i = \bar{r}/2^i$, see Fig. 8). In practice the optimized implementation of our KD-tree, for a structure indexing 20K points, allows to perform hundreds of thousands of field queries per second. The specific distance fields to be queried will depend on the distance of the viewer. We omit the details for the sake of reducing the paper length.

The multi scale approach allows to achieve the desired camera *smoothness* during the object inspection: in the case of close inspection, panning makes the camera hover above the surface, while in the case of distant observation the behavior is very similar to the standard trackball orbit operation. That's because the *smoothness* of the fields increases with growing distance: for low values the resulting offset surface wraps the model with close approximation while with increasing distances the surface degenerates to a sphere (see Fig. 9).



In addition to scale dependency, we added another simple modification that enables a more comprehensive view for the object. Since the camera eye point evaluates to d on the distance field we are sure that the model geometry is at least at distance d from that point. Thanks to this property, we can offset the eye point E along the negative view direction (see side figure). Applying this feature the space of camera eye points is no longer an offset surface but the perceptual feedback for the user is not negatively affected. As a simple analogy to compare the different approaches, imagine a squared room with some paintings on one wall. The classical approach enables to look at the wall at most

from the center of the room; the camera offsetting, instead, allows to look at it from the opposite wall.

7. Results and Evaluation

Performance. The main advantage of point-sampling is that the number of samples is regardless of the actual resolution detail of the model (the number of faces), so the computational cost is uniform across models and thus can be easily forecasted. For models counting 1-2M of triangles, we found that a minimum sampling radius equal to $1/100$th of the bounding volume diameter is a good value. Except for meshes with great variance of primitives size, this allows to interactively observe with great detail even the single geometry primitives.
Experimental data show that the linear search *ray*-operator converges within 2-7 iterations, each performing two queries to distinct KD-trees field (to implement scale dependency). The NEWUOA optimization process takes, on average, 80 iterations to converge. During interaction, at most 30K KD-tree *NN* queries per second are required.
The memory overhead for the proposed technique is given by the point-clouds and the KD-trees built on them. With the parameters shown above, the point-cloud of the most dense sampling level counts 10-30K points that implies a small memory footprint (less than 2Mb). For computationally limited devices it is possible increase the overall efficiency by simply widening the minimum sampling radius. The limitation of such a choice could be seen only on very complex models that would not be possible to visualize on such devices.

An empirical evaluation of the quality and speed of the manipulation experience can be assessed by looking at the accompanying video.

Evaluation. In order to assess the effectiveness of the GT we designed the following user study based on two tasks that requires the close inspection of a 3D model. The user was requested to perform the task twice with the two different interaction mechanisms (standard trackball and GT). We choose the following two tasks:

- Given a 3D model of a car, we asked to the user to closely inspect it by selecting for each wheel a single view that should frame the wheel completely and show it from a direction parallel to the wheel axis;
- Given a 3D model of a piece of art (a medieval stone cross with detailed carving), we asked to the user to inspect the model and frame, similarly to previous test, four features but with a specific order.

The interaction mechanisms was randomly chosen for each user. Instruction were presented to the users by means of a printed description that included images of the target views to be reproduced. We used 12 subjects, half of the subjects were 3D expert, with long experience in many different environments (like high-end 3D modeling tools, visualization systems, etc.) and the other half had no experience in 3D manipulation (except console gaming experience). The Table 1 reports the average times, in seconds, needed for an user to perform the tasks. The results of the user study clearly show that the GT was significantly simpler to use. Our approach allowed to complete the tasks in a time that was on average 24% shorter for both naive and expert user. Looking at the times we can note that the 'Cross' task was significantly harder than the 'Car' task: average times of both naive and expert users and with both interaction mechanisms were higher. In this case the relative improvement using our approach is higher for harder task: 27%. We could say that probably when the interaction is more difficult the benefits of a more sophisticated mechanism are more evident. We also collected feedback from our test subjects and in all cases the GT was the preferred choice. The most common motivation provided is that our interaction scheme provides a more pleasant experience and feels more natural.

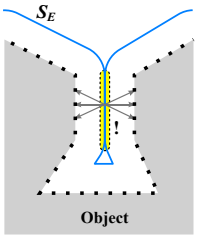
	Standard Trackball			Generalized Trackball		
	expert	naive	all	expert	naive	all
Car	45.3	88.1	66.7	33.9	74.2	54.1
Cross	71.0	115.5	93.3	52.5	83.6	68.8
Total	58.1	101.9	80.0	43.2	78.9	61.0

Table 1: *Average times for completing the tasks (standard trackball vs. GT) for naive and expert users.*

8. Conclusions

We presented the Generalized Trackball, a solution to control in a very intuitive and direct manner the interactive inspection of 3D objects with complex shape. The solution proposed is effective, efficient and can be applied to both mouse-based platforms and mobile devices adopting a touch-based interface. Our approach features a very small memory overhead and negligible preprocessing times; it is based on a robust mesh-less adaptive representation that works on any kind of model without introducing topology constraints. We assessed the system with naive and expert users, confirming the significance of the proposed technique in shortening interaction times.

Future extensions. We envision several possible improvements to the GT approach. First of all, despite GT supports a very natural and intuitive exocentric object inspection, there is still a theoretically unsolved issue. For objects with a complex topology, presenting many holes and cavities, the distance field gradient may be undefined because in some internal regions where there can be points equally distant from different portions of the object surface (see side figure). The normal field that we use achieves normal smoothing in correspondence with concave object regions, but in this situations we cannot provide a consistent value because there is no actual value. For this reason, if following exactly the presented framework, the camera could remain stuck, zoom-out could not be performed and the pan operation behaves like an egocentric technique. In this case only a zoom-in operation can take the camera eye to a distance value where the field achieves again the expected behavior. For practical purposes, to recover from these situations, we simply move towards the last admissible position traversed by the user at the target distance. While there can be situations in which this approach could lead to non optimal paths, in practice is simple to be implemented



and works well.
In the future we would like to solve this issue in a more clean way, by using a zoom-out operation that allows the camera eye to follow the undefined gradient region (using the outer medial axis of the model detected with the distance field). In this way we will be able to execute an operation similar to the *push-out* mode of the Multiscale 3D Navigation technique [MMGK09].

Another possible extension would be to explore alternative approximations for the object surface and the distance field. One thing in particular that would be helpful is to produce a parametric representation for the mesh surface that smoothly approximates the original topology. Perhaps this approach may speed-up the camera manipulation performance and, hopefully, provide direct methods for the offset surfaces calculations.

Acknowledgment

The research leading to these results was funded by EU FP7 project ICT FET Harvest4D (http://www.harvest4d.org/, G.A. no. 323567).

References

[AVF04] ANDUJAR C., VAZQUEZ P., FAIREN M.: Way-finder: guided tours through complex walkthrough models. *Computer Graphics Forum 23*, 3 (2004), 499–508. 2

[Bel88] BELL G.: Bell's trackball. Written as part of the "flip" demo to demonstrate the Silicon Graphics ws, 1988. 2

[BKF*02] BURTNYK N., KHAN A., FITZMAURICE G., BALAKRISHNAN R., KURTENBACH G.: *StyleCam: Interactive stylized 3D navigation using integrated spatial & temporal controls*, vol. 4. ACM, 2002, pp. 101–110. 2

[BKFK06] BURTNYK N., KHAN A., FITZMAURICE G., KURTENBACH G.: Showmotion: camera motion based 3d design review. *Proc. of 2006 Symp. on Interactive 3D graphics and games* (2006), 167–174. 2

[Bou14] BOUBEKEUR T.: Shellcam: Interactive geometry-aware virtual camera control. In *Image Processing (ICIP), 2014 IEEE International Conference on* (Oct 2014), pp. 4003–4007. 2

[CCR08] CIGNONI P., CORSINI M., RANZUGLIA G.: Meshlab: an open-source 3d mesh processing system. *ERCIM News*, 73 (April 2008), 45–46. 2

[CCS12] CORSINI M., CIGNONI P., SCOPIGNO R.: Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transaction on Visualization and Computer Graphics 18*, 6 (2012), 914–924. 4

[CMS88] CHEN M., MOUNTFORD S. J., SELLEN A.: A study in interactive 3-d rotation using 2-d control devices. In *Proc. of 15th annual conference on Computer Graphics* (1988), SIGGRAPH '88, ACM, pp. 121–129. 2

[CON08] CHRISTIE M., OLIVIER P., NORMAND J.-M.: Camera control in computer graphics. *Computer Graphics Forum 27*, 8 (2008), 2197–2218. 2

[FMM*08] FITZMAURICE G., MATEJKA J., MORDATCH I., KHAN A., KURTENBACH G.: Safe 3d navigation. In *Proc. Symp. on Interactive 3D graphics and Games* (2008), I3D '08, ACM, pp. 7–15. 2

[HDKG08] HACHET M., DECLE F., KNODEL S., GUITTON P.: Navidget for easy 3d camera positioning from 2d inputs. In *IEEE Symp. on 3D User Interfaces, 3DUI 2008.* (march 2008), pp. 83 –89. 2

[HSH04] HENRIKSEN K., SPORRING J., HORNBÆK K.: Virtual trackballs revisited. *IEEE Transactions on Visualization and Computer Graphics 10*, 2 (Mar. 2004), 206–216. 2

[HW97] HANSON A. J., WERNERT E. A.: Constrained 3d navigation with 2d controllers. In *Proc. 8th IEEE Visualization Conf.* (1997), VIS '97, IEEE Comp. Soc. Press, pp. 175–ff. 2

[JH13] JANKOWSKI J., HACHET M.: A Survey of Interaction Techniques for Interactive 3D Environments. In *Eurographics 2013 - State of the Art Reports* (2013), Sbert M., Szirmay-Kalos L., (Eds.), The Eurographics Association. 2

[KKS*05] KHAN A., KOMALO B., STAM J., FITZMAURICE G., KURTENBACH G.: Hovercam: interactive 3d navigation for proximal object inspection. In *Proc. 2005 Symp. on Interactive 3D Graphics and Games* (2005), I3D '05, ACM, pp. 73–80. 1, 2, 3, 6

[KMF*08] KHAN A., MORDATCH I., FITZMAURICE G., MATEJKA J., KURTENBACH G.: Viewcube: a 3d orientation indicator and controller. In *Proc. of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2008), I3D '08, ACM, pp. 17–25. 2

[LC15] LINO C., CHRISTIE M.: Intuitive and efficient camera control with the toric space. *ACM Trans. Graph. 34*, 4 (July 2015), 82:1–82:12. 2

[LH07] LEFEBVRE S., HOPPE H.: Compressed random-access trees for spatially coherent data, 2007. 4

[LL88] LEYMARIE F., LEVINE M.: *Curvature morphology*. McGill University, 1988. 3

[MAG*12] MARTON F., AGUS M., GOBBETTI E., PINTORE G., RODRÍGUEZ M. B.: Natural exploration of 3d massive models on large-scale light field displays using the fox proximal navigation technique. *Computers & Graphics 36*, 8 (2012), 893 – 903. Graphics Interaction Virtual Environments and Applications 2012. 2

[MMGK09] MCCRAE J., MORDATCH I., GLUECK M., KHAN A.: Multiscale 3d navigation. In *Proc. 2009 Symp. on Interactive 3D Graphics and Games* (2009), I3D '09, ACM, pp. 7–14. 2, 9

[MRB*14] MARTON F., RODRÍGUEZ M. B., BETTIO F., AGUS M., VILLANUEVA A. J., GOBBETTI E.: Isocam: Interactive visual exploration of massive cultural heritage models on large projection setups. *ACM J. Comput. Cult. Herit. 7*, 2 (June 2014), 12:1–12:24. 2

[Pow06] POWELL M.: The newuoa software for unconstrained optimization without derivatives. In *Large-Scale Nonlinear Optimization*, Di Pillo G., Roma M., (Eds.), vol. 83 of *Nonconvex Optimization and Its Applications*. Springer US, 2006, pp. 255–297. 6

[RAMG14] RODRÍGUEZ M. B., AGUS M., MARTON F., GOBBETTI E.: Humors: Huge models mobile rendering system. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies* (New York, NY, USA, 2014), Web3D '14, ACM, pp. 7–15. 2

[Ser83] SERRA J.: *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983. 3

[Sho92] SHOEMAKE K.: Arcball: a user interface for specifying three-dimensional orientation using a mouse. In *Proc. of Graphics Interface '92* (1992), Morgan Kaufmann Publishers Inc., pp. 151–156. 2

[WDK01] WAN M., DACHILLE F., KAUFMAN A.: Distance-field based skeletons for virtual navigation. In *Proc. IEEE Visualization '01* (2001), IEEE Comp. Soc., pp. 239–246. 2

[WO90] WARE C., OSBORNE S.: Exploration and virtual camera control in virtual three dimensional environments. In *Proc. 1990 Symp. on Interactive 3D Graphics* (1990), I3D '90, ACM, pp. 175–183. 2