Interactive Axis-Based 3D Rotation Specification Using Image Skeletons

Xiaorui Zhai[1][0000−0002−4244−9485], Xingyu Chen[1,2][0000−0002−3770−4357], Lingyun Yu[3][0000−0002−3152−2587], and Alexandru Telea[4][0000−0003−0750−0502]

[1] Bernoulli Institute, University of Groningen, The Netherlands
[2] School of Computer and Communication Engineering, University of Science and Technology Beijing, China
[3] Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China
[4] Utrecht University, The Netherlands
x.zhai@rug.nl, xingyu.chen@rug.nl, Lingyun.Yu@xjtlu.edu.cn, a.c.telea@uu.nl

Abstract. Specifying 3D rotations of shapes around arbitrary axes is not easy to do. We present a new method for this task, based on the concept of natural local rotation axes. We define such axes using the 3D curve skeleton of the shape of interest. We compute effective and efficient approximations of such skeletons using the 2D projection of the shape. Our method allows users to specify 3D rotations around parts of arbitrary 3D shapes with a single click or touch, is simple to implement, works in real time for large scenes, can be easily added to any OpenGL-based scene viewer, and can be used on both mouse-based and touch interfaces.

Keywords: Skeletonization · 3D Interaction · Image-based techniques.

1 Introduction

Interactive manipulation of 3D scenes is a key part of many applications such as CAD/CAM modeling, computer games, and scientific visualization [12]. 3D rotations are an important manipulation type, as they allow examining scenes from various viewpoints to *e.g.* select the most suitable one for the task at hand. Two main 3D rotation types exist – rotation around a *center* and rotation around an *axis*. The first one can be easily specified via classical (mouse-and-keyboard) [28] or touch interfaces [27] by well-known metaphors such as the trackball. The latter is also easy to specify if the rotation axis coincides with one of the world-coordinate axes. Rotations around arbitrary axes are considerably harder to specify, as this requires a total of 7 degrees of freedom (6 for specifying the axis and one for the rotation angle around the axis).

Users often do not need to rotate around *any* 3D axis. Consider the case when one wants to examine a (complex) 3D shape such as a statue: It can be argued that a good viewpoint will display the statue in a 'natural' position, i. e., with the head upwards. Next, a 'natural' way to rotate this shape is around its vertical symmetry axis. This keeps the shape's global orientation (which helps understanding the shape) but allows one to examine it from all viewpoints.

Several methods support the above exploration scenario by first aligning a shape's main symmetry axis with one of the world coordinate axes and then using a simple-to-specify rotation around this world axis [6]. This scenario falls short when (a) the studied shape does not admit a *global* symmetry axis, although its parts may have local symmetry axes; (b) computing such (local or global) symmetry axes is not simple; or (c) we do not want to rotate along an axis which is first aligned with a world axis.

To address the above, we propose a novel interaction mechanism: Given a shape viewed from an arbitrary 3D viewpoint, we allow the user to choose a part of interest of the shape. Next, we propose a fast and generic method to compute an approximate 3D symmetry axis for this part. Finally, we interactively rotate the shape around this axis by the desired angle. This effectively allows one to rotate the viewpoint to examine shapes around a multitude of symmetry axes that they can easily select. Our method can handle any 3D shape or scene, *e.g.*, polygon mesh or polygon soup, point-based or splat-based rendering, or combination thereof; is simple to implement and works at interactive rates even for scenes of hundreds of thousands of primitives; requires no preprocessing of the 3D geometry; and, most importantly, allows specifying the rotation axis and rotation angle by a single click, therefore being suitable for both classical (mouse-based) and touch interfaces. We demonstrate our method on several types of 3D scenes and exploration scenarios.

2 Related Work

Rotation specification: Many mechanisms for specifying 3D scene rotation exist. The trackball metaphor [1, 28] is one of the oldest and likely most frequently used. Given a 3D center-of-rotation $\mathbf{x}$, the scene is rotated around an axis passing through $\mathbf{x}$ and determined by the projections on a hemisphere centered at $\mathbf{x}$ of the 2D screen-space locations $\mathbf{p}_1$ and $\mathbf{p}_2$ corresponding to a mouse pointer motion. The rotation angle α is controlled by the amount of pointer motion. Trackball rotation is simple to implement and allows freely rotating a shape to examine it from all viewpoints. Yet, controlling the actual axis around which one rotates is hard, as this axis constantly changes while the user moves the mouse. At the other extreme, world-coordinate-axis rotations allow rotating a 3D scene around the x, y, or z axes [28, 12]. The rotation axis and rotation amount α can be chosen by simple click-and-drag gestures in the viewport. This method works best when the scene is already *pre-aligned* with a world axis, so that rotating around that axis provides meaningful viewpoints.

Pre-alignment of 3D models is a common preprocessing stage in visualization [4]. Principal Component Analysis (PCA) does this by computing a shape's eigenvectors $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, ordered by the respective eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$, so that the coordinate system $\{\mathbf{e}_i\}$ is right-handed. Next, the shape can be suitably aligned with the viewing coordinate system $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ by a simple 3D rotation around the shape's barycenter [22, 13].

3D rotations can be specified by classical (mouse-and-keyboard) [28] but also touch interfaces. Yu *et al.* [27] presented a direct-touch exploration technique for 3D scenes called Frame Interaction with 3D space (FI3D). Guo *et al.* [9] extended FI3D with constrained rotation, trackball rotation, and rotation around a user-defined center. [26] used trackball interaction to control rotation around two world axes by mapping it to single-touch interaction.

Medial descriptors: Medial descriptors, also known as skeletons, are used for decades to capture the symmetry structure of shapes [19]. For shapes $\Omega \subset \mathbb{R}^n$, $n \in \{2, 3\}$ with boundary $\partial\Omega$, skeletons are defined as

$$S_\Omega = \{\mathbf{x} \in \Omega) | \exists \mathbf{f}_1 \in \partial\Omega, \mathbf{f}_2 \in \partial\Omega : \mathbf{f}_1 \neq \mathbf{f}_2 \wedge ||\mathbf{x} - \mathbf{f}_1|| = ||\mathbf{x} - \mathbf{f}_2|| = DT_\Omega(\mathbf{x}\} \quad (1)$$

where $\mathbf{f}_i$ are called the feature points of skeletal point $\mathbf{x}$ and DT_Ω is the distance transform [18] of skeletal point $\mathbf{x}$, defined as

$$DT_\Omega(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\| \quad (2)$$

The feature points define the so-called feature transform [10]
Displayed equations are centered and set on a separate line.

$$FT_\Omega(\mathbf{x} \in \Omega) = \arg\min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\| \quad (3)$$

In 3D, two skeleton types exist [21]: *Surface skeletons*, defined by Eqn 1 for $\Omega \subset \mathbb{R}^3$, consist of complex intersecting manifolds with boundary, and hence are hard to compute and utilize. *Curve skeletons* are curve-sets in $\mathbb{R}^3$ that locally capture the tubular symmetry of shapes. They are structurally much simpler than surface skeletons and enable many applications such as shape segmentation [17] and animation [2]. Yet, they still cannot be computed in real time, and require a well-cured definition of Ω as either a watertight, non-self-intersecting, fine mesh [20], or a high-resolution voxel volume [16].

3 Proposed Method

We construct a 3D rotation in five steps (Fig. 1) which can be integrated into any OpenGL-based 3D scene viewer. We start by loading the scene of interest into the viewer (a). Next, the user can employ any standard mechanisms offered by the viewer, *e.g.* trackball rotation, zoom, or pan, to choose a viewpoint of interest, from which the viewed scene shows a detail around which one would like to further rotate to explore the scene. In our example, such a viewpoint (b) shows the horse's rump, around which we next want to rotate the horse to view it from different angles.

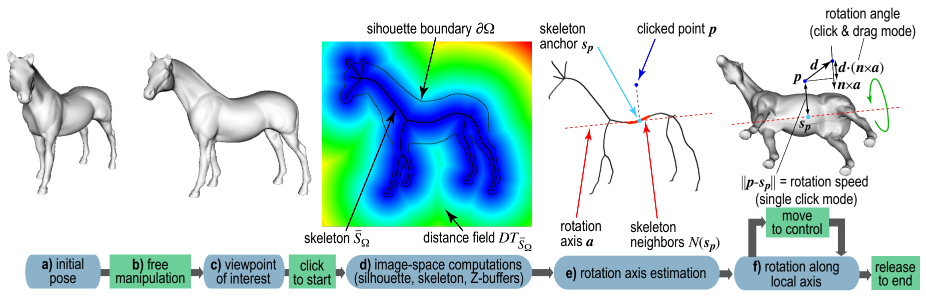
sihouette boundary ∂Ω
skeleton anchor s_p
clicked point p
rotation angle (click & drag mode)
d
$d\cdot(n\times a)$
$n\times a$
p
s_p
skeleton $\bar{S}_\Omega$
distance field $DT_{\bar{S}_\Omega}$
rotation axis a
skeleton neighbors $N(s_p)$
$\|p-s_p\|$ = rotation speed (single click mode)
move to control
a) initial pose
b) free manipulation
c) viewpoint of interest
click to start
d) image-space computations (silhouette, skeleton, Z-buffers)
e) rotation axis estimation
f) rotation along local axis
release to end

Fig. 1. Skeleton-based local rotation pipeline. Blue boxes indicate tool states. Green boxes indicates user actions.

3.1 Rotation axis computation

From the above-mentioned initial viewpoint, we next perform three image-space operations to compute the 3D rotation axis. These steps, denoted A, B, and C next, are as follows.

A. Silhouette extraction: This is the first operation in step (d) in Fig. 1. We render the shape with Z buffering on and using the standard *GL_LESS* OpenGL depth-test. Let Ω_{near} be the resulting Z buffer. We next find the silhouette Ω of the rendered shape as all pixels that have a value in Ω_{near} different from the default (the latter being 1 for standard OpenGL settings).

B. Skeleton computation: We next compute the silhouette skeleton S_Ω following Eqn. 1. This is the second operation in step (d) in Fig. 1. To eliminate spurious skeletal branches caused by small-scale noise along $\partial\Omega$, we regularize S_Ω by using the salience-based metric in [23]. Briefly put, this regularization works as follows. For every point $\mathbf{x} \in S_\Omega$ of the full skeleton computed by Eqn. 1, we compute first the so-called *importance* $\rho(\mathbf{x}) \in \mathbb{R}^+$, defined as the shortest path along $\partial\Omega$ between the two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$. As shown in [24], ρ monotonically increases along skeletal branches from their endpoints to the skeleton center, and equals, for a skeleton point $\mathbf{x}$, the amount of boundary length which is captured (described) by $\mathbf{x}$. Next, the saliency of point $\mathbf{x}$ is defined as

$$\sigma(\mathbf{x}) = \frac{\rho(\mathbf{x})}{DT_\Omega(\mathbf{x})} \qquad (4)$$

As shown in [23], the saliency is *overall low* on skeleton branches caused by small-scale details along $\partial\Omega$ and *overall high* on skeleton branches caused by important (salient) protrusions of $\partial\Omega$. Hence, we can regularize S_Ω simply by removing all its pixels having a salience value lower than a fixed threshold σ_0. Following [23], we set $\sigma_0 = 1$. Fig. 2 illustrates the regularization process by showing the raw skeleton S_Ω and its regularized version $\overline{S}_\Omega$ for a noisy shape. As visible in image (b), the salience regularization removes all spurious branches created by surface noise, but leaves the main skeleton branches, corresponding to the animal's limbs, rump, and tail, intact. Salience regularization is simple and automatic to use, requiring no parameters to be controlled by the user – for details, we refer to [23].

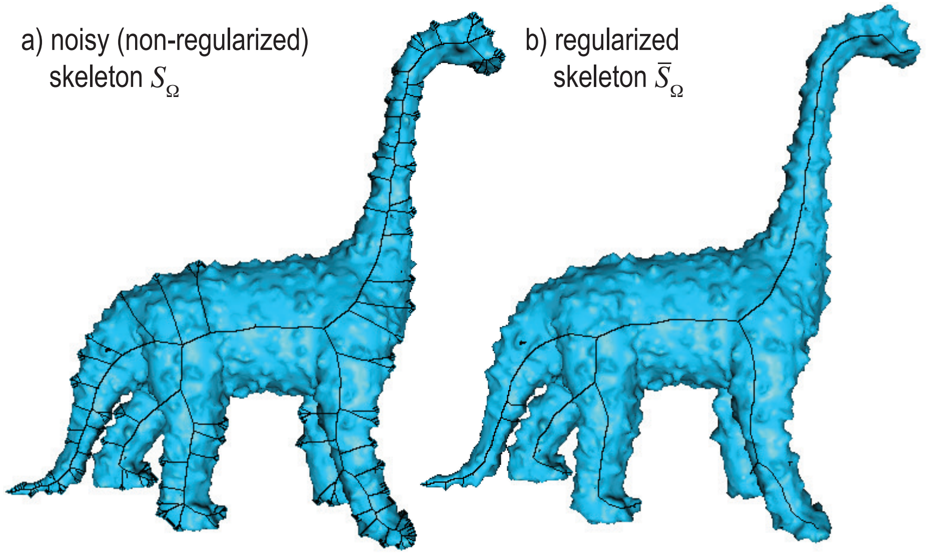
a) noisy (non-regularized) skeleton S_Ω
b) regularized skeleton $\overline{S}_\Omega$

Fig. 2. Raw skeleton S_Ω with noise-induced branches (a) and saliency-based regularized version $\overline{S}_\Omega$ (b).

C. Rotation axis computation: This is step (e) in Fig. 1. Let $\mathbf{p}$ be the pixel under the user-manipulated pointer (blue in Fig. 1e). We first find the closest skeleton point $\mathbf{s}_p = \text{argmin}_{\mathbf{y}\in\overline{S}_\Omega}\|\mathbf{p}-\mathbf{y}\|$ by evaluating the feature transform (Eqn. 3) $FT_{\overline{S}_\Omega}(\mathbf{p})$ of the regularized skeleton $\overline{S}_\Omega$ at $\mathbf{p}$. Figure 1d shows the related distance transform $DT_{\overline{S}_\Omega}$. In our case, $\mathbf{s}_p$ is a point on the horse's rump skeleton (cyan in Fig. 1e). Next, we find the neighbor points $N(\mathbf{s}_p)$ of $\mathbf{s}_p$ by searching depth-first from $\mathbf{s}_p$ along the pixel connectivity-graph of $\overline{S}_\Omega$ up to a maximal distance set to 10% of the viewport size. $N(\mathbf{s}_p)$ contains skeletal points along a single branch in $\overline{S}_\Omega$, or a few connected branches, if $\mathbf{s}_p$ is close to a skeleton junction. In our case, $N(\mathbf{s}_p)$ contains a fragment of the horse's rump skeleton (red in Fig. 1e). For each $\mathbf{q} \in N(\mathbf{s}_p)$, we next estimate the depth $\mathbf{q}_z$ as the average of $\Omega_{far}(\mathbf{q})$ and $\Omega_{near}(\mathbf{q})$. Here, Ω_{far} is the Z buffer of the scene rendered with front-face culling on and the standard *GL_LESS* OpenGL depth-test, giving thus the depth of the nearest backfacing-polygons to the view plane.

Fig. 3 illustrates this. First (a), the user clicks above the horse's rump and drags the pointer upwards. Image (b) shows the resulting rotation. As visible, the rotation axis (red) is centered inside the rump, as its depth $\mathbf{q}_z$ is the average of the near and far rump faces. Next (c), we consider a case of overlapping shape parts. The user clicks left to the horse's left-front leg, which overlaps the right-front one. Image (d) shows the resulting rotation. Again, the rotation axis (red) is centered inside the left-front leg. In this case, $\Omega_{far}(\mathbf{q})$ contains the Z value on the backfacing part of the left-front leg, so $(\Omega_{near}(\mathbf{q}) + \Omega_{far}(\mathbf{q}))/2$ yields a value roughly halfway this leg along the Z axis. Separately, we handle non-watertight surfaces as follows: If $\Omega_{far}(\mathbf{q})$ contains the default (maximal) Z value, this means there's no backfacing surface under a given pixel $\mathbf{q}$, so the scene is not watertight at that point. We then set $\mathbf{q}_z$ to $\Omega_{near}(\mathbf{q})$.

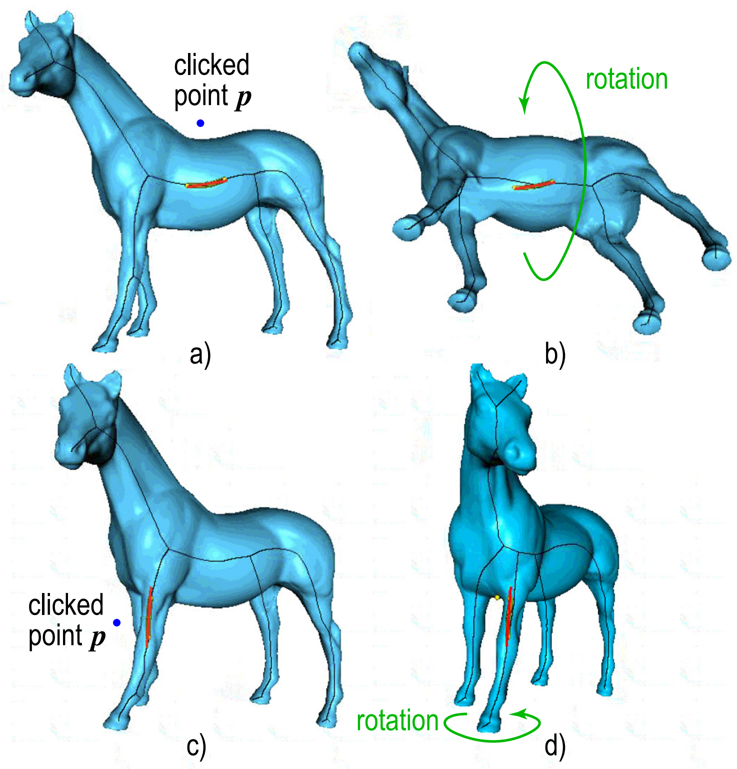

Fig. 3. Depth estimation of rotation axis for (a,b) non-overlapping part and (c,d) overlapping parts. In both cases, the rotation axis (red) is nicely centered in the shape.

We now have a set $N_{3D} = \{(\mathbf{q} \in N(\mathbf{s}_p), \mathbf{q}_z)\}$ of 3D points that approximate the 3D curve skeleton of our shape close to the pointer location $\mathbf{p}$. We set the 3D rotation axis $\mathbf{a}$ to the line passing through the average point of N_{3D} and oriented along the largest eigenvector of N_{3D}'s covariance matrix (Fig. 1e, red dotted line).

3.2 Controlling the rotation

We offer three interactive mechanisms to control the rotation (step (f) in Fig. 1), as follows.
Indication: As the user moves the pointer $\mathbf{p}$, we continuously update the display of $\mathbf{a}$. This shows along which axis the scene would rotate if the user initiated the rotation from $\mathbf{p}$. If $\mathbf{a}$ is found suitable, one can start rotating by a click following one of the two modes listed next; else one can move the pointer $\mathbf{p}$ to find a more suitable axis;
Single click: In this mode, we compute a rotation speed σ equal to the distance $\|\mathbf{p}-\mathbf{s}_p\|$ and a rotation direction δ (clockwise or anticlockwise) given by the sign of the cross-product $(\mathbf{s}_p - \mathbf{p}) \times \mathbf{n}$, where $\mathbf{n}$ is the viewplane normal. We next continuously rotate (spin) the shape around a with the speed σ in direction δ;
Click and drag: Let $\mathbf{d}$ be the drag vector created by the user as she moves the pointer $\mathbf{p}$ from the current to the next place in the viewport with the control, *e.g.* mouse button, pressed. We rotate the scene around $\mathbf{a}$ with an angle equal to $\mathbf{d} \cdot (\mathbf{n} \times \mathbf{a})$ (Fig. 1e).

We stop rotation when the user release the control (mouse button, touchpad, or touch screen). In single-click mode, clicking closer to the shape rotates slowly, allowing to examine the shape in detail. Clicking farther rotates quicker to *e.g.* explore the shape from the opposite side. The rotation direction is given by the *side* of the skeleton where we click: To change from clockwise to counterclockwise rotation in the example in Fig. 1, we only need to click below, rather than above, the horse's rump. In click-and-drag mode, the rotation speed and direction is given by the drag vector $\mathbf{d}$: Values $\mathbf{d}$ orthogonal to the rotation axis $\mathbf{a}$ create corresponding rotations clockwise or anticlockwise around $\mathbf{a}$; values $\mathbf{d}$ along $\mathbf{a}$ yield no rotation. This matches the intuition that, to rotate along an axis, we need to move the pointer *across* that axis.

The skeleton-based construction of the rotation axis is key to the effectiveness of our approach: If the shape exhibits some elongated structure in the current view (*e.g.* rump or legs in Fig. 1c), this structure will yield a skeleton branch. Clicking closer to this structure than to other structures in the same view – *e.g.*, clicking closer to the rump than to the horse's legs or neck – selects the respective skeleton branch to rotate around. This way, the 3D rotation uses the 'natural' structure of the viewed shape. We argue that this makes sense in an exploratory scenario, since, during rotation, the shape parts we rotate around stay *fixed* in the view, as if one 'turns around' them.

The entire method requires a *single click* and, optionally, a pointer drag motion to execute. This makes our method simpler than other 3D rotation methods that rotate around freely specifiable 3D axes, and also directly applicable to contexts where no second button or modifier keys are available, *e.g.*, touch screens. Moreover, our method does not require any complex (and/or slow) 3D curve-skeleton computation: We compute only 2D (silhouette) skeletons, which are fast and robust to extract [24, 8]. We can handle any 3D input geometry, *e.g.*, meshes, polygon soups, point clouds, or mixes thereof, as long as such primitives render in the Z buffer (see Sec. 4 for examples hereof).

3.3 Improvements of Basic Method

We next present three improvements of the basic local-axis rotation mechanism described above.

Zoom level: A first issue regards computing the scene's 2D silhouette Ω (Sec. 3.1A). For this to work correctly, the entire scene must be visible in the current viewport. If this is not the case, the silhouette boundary $\partial\Omega$ will contain parts of the viewport borders. Fig. 4a shows this for a zoomed-in view of the horse model, with the above-mentioned border parts marked purple.

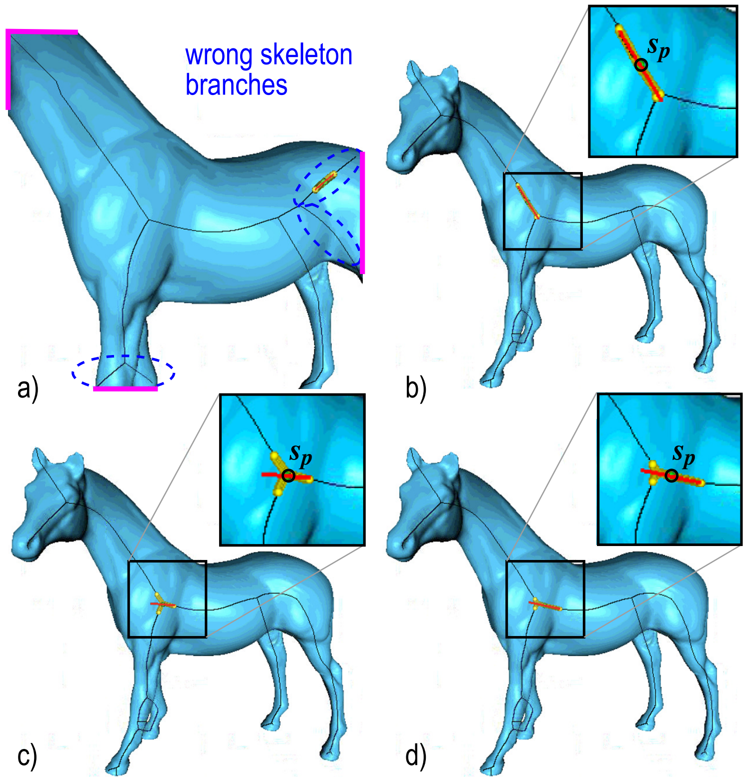

Fig. 4. Two problems of estimating rotation axes from skeletons. (a) Zoomed-in scene. (b-d) Anchor points close to a skeleton junction. See Sec. 3.3.

This leads to branches in the skeleton S_Ω that do not provide meaningful rotation axes. We prevent this to occur by requiring that the entire scene is visible in the viewport before initiating the rotation-axis computation. If this is not the case, we do not allow the skeleton-based rotation to proceed, but map the user's interaction to a traditional trackball-based rotation.

Skeleton junctions: If the user selects $\mathbf{p}$ so that the skeleton anchor $\mathbf{s}_p$ is too close to a skeleton junction, then the neighbor-set $N(\mathbf{s}_p)$ will contain points belonging to more than two branches. Estimating a line from such a point set (Sec. 3.1C) is unreliable, leading to possibly meaningless rotation axes. Figures 4b-d illustrates the problem. The corresponding skeleton points $N(\mathbf{s}_p)$ used to estimate the axis are shown in yellow, and the resulting axes in red. When $\mathbf{s}_p$ is relatively far from the junction (Figs. 4b,d), $N(\mathbf{s}_p)$ contains mainly points from a single skeleton branch, so the estimated rotation axes are reliable. However, when $\mathbf{s}_p$ is very close to the junction (Fig. 4c), $N(\mathbf{s}_p)$ contains points from all three meeting branches, so, as the user moves the pointer $\mathbf{p}$, the estimated axis 'flips' abruptly and can even assume orientations that do not match any skeleton branch.

We measure the *reliability* of the axis $\mathbf{a}$ by the anisotropy ratio $\gamma = \lambda_1/\lambda_3$ of the largest to smallest eigenvalue of N_{3D}'s covariance matrix. Other anisotropy metrics can be used equally well, *e.g.* [7]. High γ values indicate elongated structures N_{3D}, from which we can reliably compute rotation axes. Low values, empirically detected as $\gamma < 5$, indicate problems to find a reliable rotation axis. When this occurs, we prevent executing the axis-based rotation.

Selection distance: A third issue concerns the position of the point $\mathbf{p}$ that initiates the rotation: If one clicks too far from the silhouette Ω, the rotation axis $\mathbf{a}$ may not match what one expects. To address this, we forbid the rotation when the distance d from $\mathbf{p}$ to Ω exceeds a given upper limit d_{max}. That is, if the user clicks too far from any silhouette in the viewport, the rotation mechanism does not start. This signals to the user that, to initiate the rotation, she needs to click closer to a silhouette.

We compute d as $DT_{\overline{\Omega}}(\mathbf{p})$, where $\overline{\Omega}$ is the viewpoint area outside Ω, *i.e.*, all viewport pixels where Ω_{near} equals the default Z buffer value (see Sec. 3.1A).

We studied two methods for estimating d_{max} (see Fig. 5). First, we set d_{max} to a fixed value, in practice 10% of the viewport size. Using a constant d_{max} is however not optimal: We found that, when we want to rotate around *thick* shape parts, such as the horse's rump in Fig. 5b, it is intuitive to select $\mathbf{p}$ even quite far away from the silhouette. This is the situation of point $\mathbf{p}_1$ in Fig. 5b. In contrast, when we want to rotate around *thin* parts, such as the horse's legs, it is not intuitive to initiate the rotation by clicking too far away from these parts. This is the situation of point $\mathbf{p}_2$ in Fig. 5b. Hence, d_{max} depends on the scale of the shape part we want to rotate around; selecting large parts can be done by clicking farther away from them than selecting small parts.

We model this by setting d_{max} to the local shape thickness (see Fig. 5c). We estimate this thickness as follows: Given the clicked point $\mathbf{p}$, we find the closest point to it on the silhouette boundary $\partial\Omega$ as $\mathbf{q} = FT_{\overline{\Omega}}(\mathbf{p})$. The shape thickness at location $\mathbf{q}$ is the distance to the skeleton, *i.e.*, $DT_{\overline{S}_{\Omega}}(\mathbf{q})$. Here, the point $\mathbf{p}_1$ is the farthest clickable point around the location $\mathbf{q}_1$ to the silhouette that allows initiating a rotation around the rump. If we click further from the silhouette than the distance d_{max} from $\mathbf{p}_1$ to $\mathbf{q}_1$, no rotation is done. For the

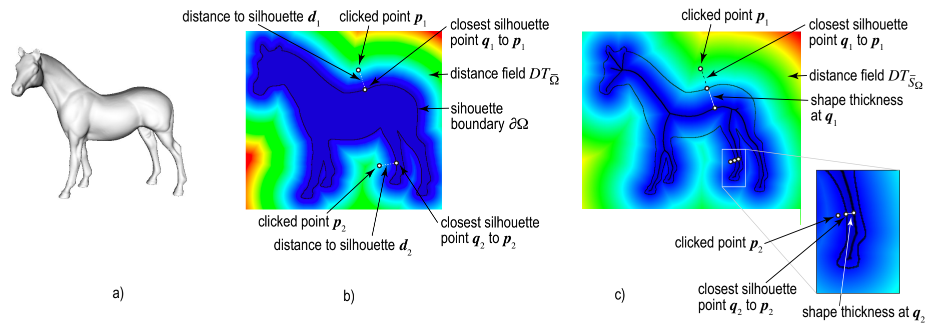
distance to silhouette d_1
clicked point p_1
closest silhouette point q_1 to p_1
distance field $DT_{\overline{\Omega}}$
sihouette boundary $\partial\Omega$
clicked point p_2
distance to silhouette d_2
closest silhouette point q_2 to p_2
clicked point p_1
closest silhouette point q_1 to p_1
distance field $DT_{\overline{S}_\Omega}$
shape thickness at q_1
clicked point p_2
closest silhouette point q_2 to p_2
shape thickness at q_2
a)
b)
c)

Fig. 5. Improvements of axis-based rotation method. (a) A view of the shape to be rotated. (b) Fixed maximum-distance setting for two clicked points $\mathbf{p}_1$ and $\mathbf{p}_2$. (c) Thickness-based maximum-distance setting for two clicked points $\mathbf{p}_1$ and $\mathbf{p}_2$.

leg part, the farthest clickable point around the location $\mathbf{q}_2$ has, however, to be much closer to the silhouette (see Fig. 5c), since here the local shape thickness, *i.e.*, the distance d_{max} from $\mathbf{p}_2$ to $\mathbf{q}_2$, is much smaller.

We model this by setting d_{max} to the local shape thickness (see Fig. 5c). We estimate this thickness as follows: Given the clicked point $\mathbf{p}$, we find the closest point to it on the silhouette boundary $\partial\Omega$ as $\mathbf{q} = FT_{\overline{\Omega}}(\mathbf{p})$. The shape thickness at location $\mathbf{q}$ is the distance to the skeleton, *i.e.*, $DT_{\overline{S}_\Omega}(\mathbf{q})$. Here, the point $\mathbf{p}_1$ is the farthest clickable point around the location $\mathbf{q}_1$ to the silhouette that allows initiating a rotation around the rump. If we click further from the silhouette than the distance d_{max} from $\mathbf{p}_1$ to $\mathbf{q}_1$, no rotation is done. For the leg part, the farthest clickable point around the location $\mathbf{q}_2$ has, however, to be much closer to the silhouette (see Fig. 5c), since here the local shape thickness, *i.e.*, the distance d_{max} from $\mathbf{p}_2$ to $\mathbf{q}_2$, is much smaller.

4 Results

Fig. 6 shows our 3D skeleton-based rotation applied to two 3D mesh models. For extra insights, we recommend also watching the demonstration videos [25]. First, we consider a 3D mesh model of a human hand (100K faces), which is not watertight (open at wrist). We start from a poor viewpoint from which we cannot easily examine the shape (a). We click close to the thumb (b) and drag to rotate around it (b-e), yielding a better viewpoint (e). Next, we want to rotate around the shape to see the other face, but keeping the shape roughly in place. Using a trackball or world-coordinate axis rotation cannot easily achieve this. We click on a point close to the shape-part we want to keep *fixed* during rotation (f), near the the wrist, and start rotation. Images (g-j) show the resulting rotation.

Fig. 6(k-ad) show a more complex ship object (380K polygons). This mesh contains multiple self-intersecting and/or disconnected parts, some very thin (sails, mast, ropes) [15]. Computing a 3D skeleton for this shape is extremely

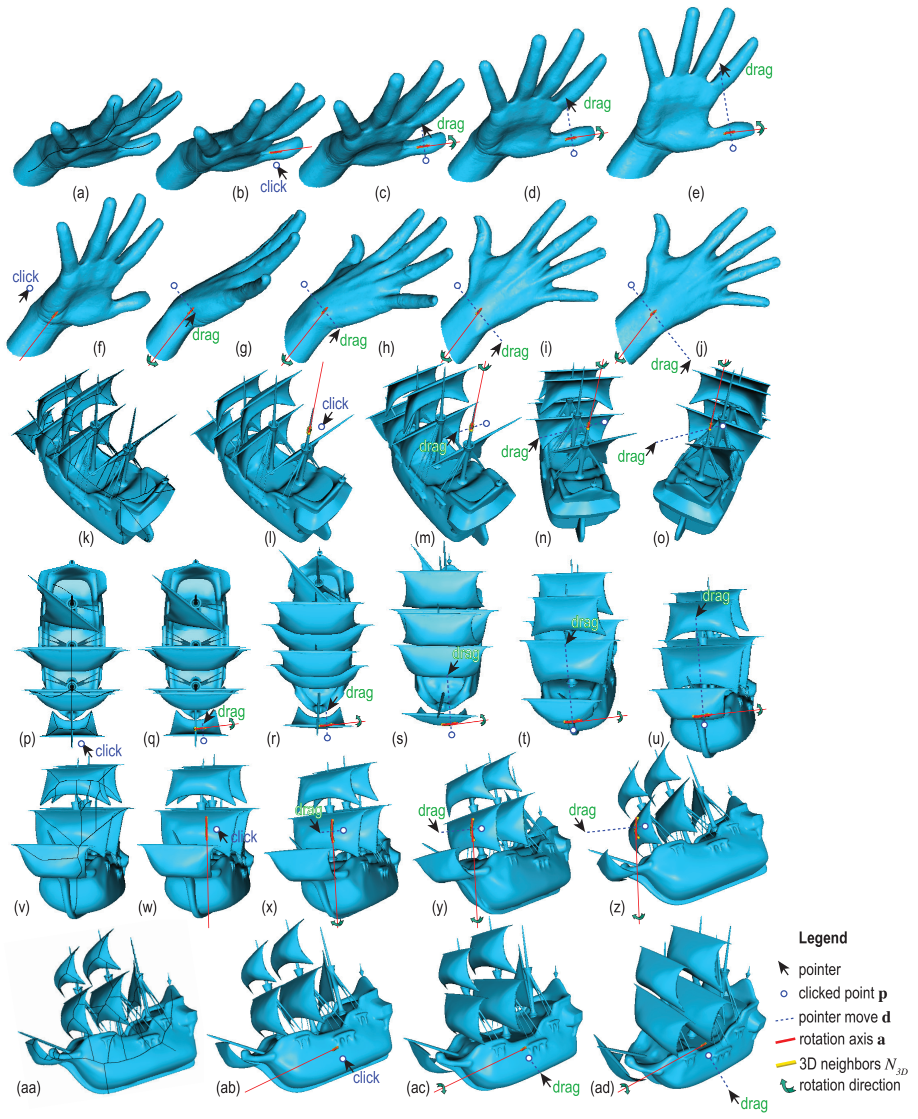

Fig. 6. Examples of two rotations (a-e), (f-j) for the hand shape and four rotations (k-o), (p-u), (v-z), (aa-ad) for the ship.

hard or even impossible, as Eqn. 1 requires a watertight, non-self-intersecting, connected shape boundary $\partial\Omega$. Our method does not suffer from this, since we compute the skeleton of the *2D silhouette* of the shape. We start again from a poor viewing angle (k). Next, we click close to the back mast to rotate around it, showing the ship from various angles (l-o). Images (p-u) show a different rotation, this time around an axis found by clicking close to the front sail, which allows us to see the ship from front. Note how the 2D skeleton has changed after this rotation – compare images (p) with (v). This allows us to select a new rotation axis by clicking on the main sail, to see the ship's stern from below (w-z). Finally, we click on the ship's rump (aa) to rotate the ship and make it vertical (ab-ad). The entire process of three rotations took around 20 seconds.

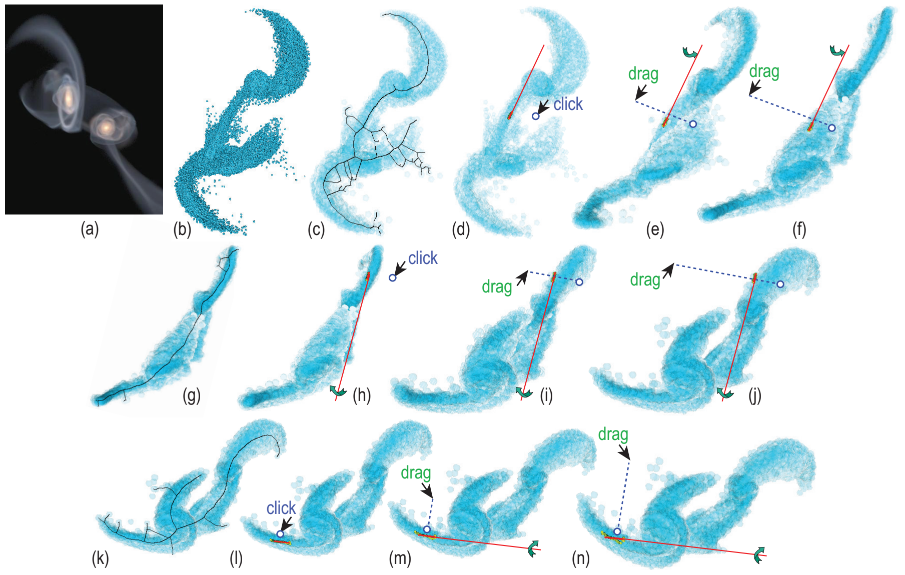

Fig. 7. Exploration of astronomical point cloud dataset. (a) Volume-rendered overview [11]. Rotations around three 3D axes (b-f), (g-j), (k-n).

Fig. 7 shows a different dataset type – a 3D point cloud that models a collision simulation between the Milky Way and the nearby Andromeda Galaxy [5, 11]. Its 160K points describe positions of the stars and dark matter in the simulation. Image (a) uses volume rendering to show the complex structure of the cloud, for illustration purposes – we do not use this rendering in our method. Rather, we render the cloud in our pipeline using 3D spherical splats (b). Image (c) shows the cloud, rendered with half-transparent splats, so that opacity reflects local point density. Since we render a 3D sphere around each point, this results in a front and back buffer Ω_{near} and Ω_{far}, just as when rendering a 3D polygonal model. From these, we can compute the 2D skeleton of the cloud's silhouette, as shown in the figure. Images (d-f) show a rotation around the central tubular structure of the cloud, which reveals that the could is relatively flat when seen from the last viewpoint (f). Image (g) shows the new 2D skeleton corresponding to the viewpoint after this rotation. We next click close to the upper high-density structure (f) and rotate around it. Images (h-j) reveal a spiral-like structure present in the lower part of the cloud, which was not visible earlier. To explore this structure better, we next click on its local symmetry axis (l) and rotate around it. Images (l-n) reveal now better this structure. As for the earlier examples, executing these three rotations took roughly 15 seconds. Scientists involved with studied this dataset for roughly a decade appreciated positively the ease of use of the skeleton-based rotation as compared to standard trackball and multi-touch gestures.

5 Discussion

We next outline our method's advantages and limitations:
Ease of use: We can rotate around 3D axes locally aligned with the scene's features with a single click and optionally pointer drag motion. This makes our method usable to contexts where no second button, modifier keys, or multi-touch input is available. Finding the axis works with even inexact click locations as we use a *set* of closest 2D-skeleton points for that ($N(\mathbf{s}_p)$, Sec. 3).
Genericity: We handle 3D meshes, polygon soups, and point clouds; our only requirement is that these generate fragments with a depth value. This contrasts using 3D curve skeletons for interaction, which heavily constrain the input scene quality, and cannot be computed in real time, as already mentioned.
Novelty: To our knowledge, this is the first time when 2D image-based skeletons have been used to perform interactive manipulations of 3D shapes. Compared to view-based reconstructions of 3D curve skeletons from their 2D silhouettes [14], our method requires a single viewpoint to compute an approximate 3D curve skeleton.
Simplicity and speed: Our method consists of basic OpenGL 1.1 operations (primitive rendering and Z-buffer reading) plus the 2D image-based skeletonization method in [8]. This skeletonization method delivers us the skeleton S_Ω, its regularization $\overline{S}_\Omega$, and the feature transform $FT_{\overline{S}_\Omega}$. This method is efficiently implemented in NVidia's CUDA and C++, so it handles scenes of hundreds of thousands of polygons rendered onto 1000^2 pixel viewports in a few milliseconds on a consumer-grade GPU, *e.g.* GTX 660. Its computational complexity is linear in the number of silhouette pixels, *i.e.*, $O(|\Omega|)$. This is due to the fact that the underlying distance transform used has the same linear complexity. For details on this, we refer to the original algorithm [3].

Implementing the two improvements presented in Sec. 3 is also computationally efficient: The skeleton's distance transform $DT_{\overline{S}_\Omega}$ is already computed during the rotation axis estimation (Sec. 3.1C). The distance $DT_{\overline{\Omega}}$ and feature transforms $FT_{\overline{\Omega}}$ require one extra skeletonization pass of the background image $\overline{\Omega}$. All in all, our interaction method delivers frame rates over 100 frames-persecond on the aforementioned consumer-grade GPU. For replication purposes, the full code of the method is provided online [25].

Limitations: Since 3D rotation axes are computed from 2D silhouette skeletons, rotations are not, strictly speaking, invertible: Rotating from a viewpoint $\mathbf{v}_1$ with an angle α around a 3D local axis $\mathbf{a}_1$ computed from the silhouette Ω_1 leads to a viewpoint $\mathbf{v}_2$ in which, from the corresponding silhouette Ω_2, a different axis $\mathbf{a}_2 \neq \mathbf{a}_1$ can be computed. This is however a problem only if the user *releases* the pointer (mouse) button to end the rotation; if the button is not released, the computation of a new axis $\mathbf{a}_2$ is not started, so moving the pointer can reverse the first rotation. Another limitation regards the measured effectiveness of our rotation mechanism. While our tests show that one can easily rotate a scene around its parts, it is still unclear which specific *tasks* are best supported by this rotation, and by how much so, as compared to other rotation mechanisms such as trackball. We plan to measure these aspects next by organizing several controlled user experiments in which we select a specific task to be completed with the aid of rotation and quantitatively compare (evaluate) the effectiveness of our rotation mechanism as compared to other established mechanisms such as trackball.

6 Conclusion

We proposed a novel method for specifying interactive rotations of 3D scenes around local axes using image skeletons. We compute local 3D rotation axes out of the 2D image silhouette of the rendered scene, using heuristics that combine the silhouette's image skeleton and depth information from the rendering's Z buffer. Specifying such local rotation axes is simple and intuitive, requiring a single click and drag gesture, as the axes are automatically computed using the closest scene fragments rendered from the current viewpoint. Our method is simple to implement, using readily-available distance and feature transforms provided by modern 2D skeletonization algorithms; can handle 3D scene consisting of arbitrarily complex polygon meshes (not necessarily watertight, connected, and/or of good quality) but also 3D point clouds; can be integrated in any 3D viewing system that allows access to the rendered Z buffer; and works at interactive frame-rates even for scenes of hundreds of thousands of primitives. We demonstrate our method on several polygonal and point-cloud 3D scenes of varying complexity.

Several extension directions are possible as follows. More cues can be used to infer more accurate 3D curve skeletons from image data, such as shading and depth gradients. Separately, we plan to execute a detailed user study to measure the effectiveness and efficiency of the proposed skeleton-based 3D rotation for specific exploration tasks of spatial datasets such as 3D meshes, point clouds, and volume-rendered data.

References

1. Bade, R., Ritter, F., Preim, B.: Usability comparison of mouse-based interaction techniques for predictable 3D rotation. In: Proc. Smart Graphics (SG). pp. 138–150 (2005)
2. Bian, S., Zheng, A., Chaudhry, E., You, L., Zhang, J.J.: Automatic generation of dynamic skin deformation for animated characters. Symmetry **10**(4), 89 (Mar 2018). https://doi.org/https://doi.org/10.3390/sym10040089
3. Cao, T.T., Tang, K., Mohamed, A., Tan, T.S.: Parallel banding algorithm to compute exact distance transform with the GPU. In: Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games. pp. 83–90 (2010)
4. Chaouch, M., Verroust-Blondet, A.: Alignment of 3D models. Graphical Models **71**(2), 63–76 (2009)
5. Dubinski, J.: When galaxies collide. Astronomy Now **15**(8), 56–58 (2001)
6. Duffin, K.L., Barrett, W.A.: Spiders: A new user interface for rotation and visualization of N-dimensional point sets. In: Proc. IEEE Visualization. pp. 205–211 (1994)
7. Emory, M., Iaccarino, G.: Visualizing turbulence anisotropy in the spatial domain with componentality contours. Center for Turbulence Research Annual Research Briefs pp. 123–138 (2014), https://web.stanford.edu/group/ctr/ResBriefs/2014/14$_{e}$*mory.pdf*
8. Ersoy, O., Hurter, C., Paulovich, F., Cantareiro, G., Telea, A.: Skeleton-based edge bundling for graph visualization. IEEE TVCG **17**(2), 2364 – 2373 (2011)
9. Guo, J., Wang, Y., Du, P., Yu, L.: A novel multi-touch approach for 3D object free manipulation. In: Proc. AniNex. pp. 159–172. Springer (2017)
10. Hesselink, W.H., Roerdink, J.B.T.M.: Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. IEEE TPAMI **30**(12), 2204–2217 (2008)
11. J. Dubinski *et al.*: GRAVITAS: Portraits of a universe in motion (2006), https://www.cita.utoronto.ca/ dubinski/galaxydynamics/gravitas.html
12. Jackson, B., Lau, T.Y., Schroeder, D., Toussaint, K.C., Keefe, D.F.: A lightweight tangible 3D interface for interactive visualization of thin fiber structures. IEEE TVCG **19**(12), 2802–2809 (2013)
13. Kaye, D., Ivrissimtzis, I.: Mesh alignment using grid based PCA. In: Proc. CGTA). pp. 174–181 (2015)
14. Kustra, J., Jalba, A., Telea, A.: Probabilistic view-based curve skeleton computation on the GPU. In: Proc. VISAPP. SCITEPRESS (2013)
15. Kustra, J., Jalba, A., Telea, A.: Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. Comp Graph Forum **33**(4), 73–87 (2014)
16. Reniers, D., van Wijk, J.J., Telea, A.: Computing multiscale skeletons of genus 0 objects using a global importance measure. IEEE TVCG **14**(2), 355–368 (2008)
17. Rodrigues, R.S.V., Morgado, J.F.M., Gomes, A.J.P.: Part-based mesh segmentation: A survey. Comp Graph Forum **37**(6), 235–274 (2018)
18. Rosenfeld, A., Pfaltz, J.: Distance functions in digital pictures. Pattern Recognition **1**, 33–61 (1968)
19. Siddiqi, K., Pizer, S.: Medial Representations: Mathematics, Algorithms and Applications. Springer (2008)
20. Sobiecki, A., Yasan, H., Jalba, A., Telea, A.: Qualitative comparison of contraction-based curve skeletonization methods. In: Proc. ISMM. Springer (2013)

21. Tagliasacchi, A., Delame, T., Spagnuolo, M., Amenta, N., Telea, A.: 3D skeletons: A state-of-the-art report. Comp Graph Forum **35**(2), 573–597 (2016)
22. Tangelder, J.W.H., Veltkamp, R.C.: A survey of content based 3D shape retrieval methods. Multimedia Tools and Applications **39**(441) (2008)
23. Telea, A.: Feature preserving smoothing of shapes using saliency skeletons. In: Proc. VMLS. pp. 136–148. Springer (2011)
24. Telea, A., van Wijk, J.J.: An augmented fast marching method for computing skeletons and centerlines. In: Proc. VisSym. pp. 251–259. Springer (2002)
25. The Authors: Source code and videos of interactive skeleton-based axis rotation (2019), http://www.staff.science.uu.nl/ telea001/Shapes/CUDASkelInteract
26. Yu, L., Isenberg, T.: Exploring one- and two-touch interaction for 3D scientific visualization spaces. Posters of Interactive Tabletops and Surfaces (Nov 2009), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.440.7750rep=rep1type=pdf
27. Yu, L., Svetachov, P., Isenberg, P., Everts, M.H., Isenberg, T.: FI3D: Direct-touch interaction for the exploration of 3D scientific visualization spaces. IEEE TVCG **16**(6), 1613–1622 (2010)
28. Zhao, Y.J., Shuralyov, D., Stuerzlinger, W.: Comparison of multiple 3D rotation methods. In: Proc. IEEE VECIMS. pp. 19–23 (2011)