Skeleton-and-Trackball Interactive Rotation Specification for 3D Scenes

Xiaorui Zhai[1], Xingyu Chen[1,2], Lingyun Yu[3], and Alexandru Telea[4(✉)]

[1] Bernoulli Institute, University of Groningen, Groningen, The Netherlands
{x.zhai,xingyu.chen}@rug.nl

[2] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China

[3] Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China
Lingyun.Yu@xjtlu.edu.cn

[4] Department of Information and Computing Science, Utrecht University, Utrecht, The Netherlands
a.c.telea@uu.nl

Abstract. We present a new technique for specifying rotations of 3D shapes around axes inferred from the local shape structure, in support of 3D exploration and manipulation tasks. We compute such axes by extracting approximations of the 3D curve skeleton of such shapes using the skeletons of their 2D image silhouettes and depth information present in the Z buffer. Our method allows specifying rotations around parts of arbitrary 3D shapes with a single click, works in real time for large scenes, can be easily added to any OpenGL-based scene viewer, and is simple to implement. We compare our method with classical trackball rotation, both in isolation and in combination, in a controlled user study. Our results show that, when combined with trackball, skeleton-based rotation reduces task completion times and increases user satisfaction, while not introducing additional costs, being thus an interesting addition to the palette of 3D manipulation tools.

Keywords: Skeletonization · 3D interaction · Image-based techniques

1 Introduction

Interactive exploration and navigation of 3D scenes is essential in many applications such as CAD/CAM modeling, computer games, and data visualization [26]. 3D rotations are an important interaction tool, as they allow examining scenes from various viewpoints. Two main 3D rotation types exist – rotation around a *center* and rotation around an *axis*. Rotation around a center can be easily specified via classical mouse-and-keyboard [51] or touch interfaces [49] by well-known metaphors such as the virtual trackball [21]. Axis rotation is easy to specify if the axis matches one of the world-coordinate axes. Rotations around arbitrary axes are much harder to specify, as this requires a total of 7 degrees of freedom (6 for the axis and one for the rotation angle around the axis).

K. Bouatouch et al. (Eds.): VISIGRAPP 2020, CCIS 1474, pp. 26–52, 2022.
https://doi.org/10.1007/978-3-030-94893-1_2

For certain tasks, users do not need to rotate around *any* 3D axis. Consider examining a (complex) 3D shape such as a statue: We can argue that a natural way to display this shape is with the statue's head upwards; and a good way to explore the shape from all viewpoints is to rotate it around its vertical symmetry axis while keeping its upwards orientation fixed.

Several methods support this exploration scenario by aligning the shape's main symmetry axis with one of the world coordinate axes and then using a simple-to-specify rotation around this world axis [12]. This 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 first align the shape with a world axis.

To address the above, Zhai *et al.* [50] recently proposed a novel interaction mechanism based on local symmetry axes: The user points at a region of interest (part) of the viewed 3D shape, from which a local symmetry axis is computed. Next, one can rotate the shape around this axis with an interactively specified angle. This method allows an easy selection of parts and automatic computation of their approximate 3D symmetry axes, both done using the shape silhouette's 2D skeleton. The method handles any 3D scene, *e.g.*, polygon mesh or polygon soup, point-based or splat-based rendering, or combination thereof, without preprocessing; and works at interactive rates for scenes of hundreds of thousands of primitives.

Zhai *et al.* mention that their skeleton-based rotation is not to be seen as a replacement, but a *complement*, of classical trackball rotation. Yet, what this precisely means, *i.e.*, how the two rotation mechanisms perform when used in practice, either separately or jointly, is an open question. Also, they mention a formal evaluation of the effectiveness of skeleton-based rotation as an important open research question. In this paper, we extend the work of Zhai *et al.* in the above directions with the following contributions:

– We provide a more detailed technical explanation of the skeleton-based rotation, covering aspects left open by the original paper [50];
– We present the design and execution of a controlled user study aimed at gauging the added value of skeleton-based rotation when used against, but also combined with, trackball rotation;
– We analyze the results of our study to show that, when used together with trackball rotation, skeleton-based rotation brings in added value, therefore being a good complement, and not replacement, of trackball rotation.

The structure of this paper is as follows. Section 2 presents related work on interactive rotation specification and skeleton computation. Section 3 details the skeleton-based rotation presented in [50]. Section 4 presents a formative evaluation aimed at finding out how the skeleton-based rotation is received by users. Section 5 presents an in-depth quantitative and qualitative user study that studies the hypotheses outlined by the formative study. Section 6 discusses the skeleton-based rotation and our findings regarding its best ways of use. Section 7 concludes the paper.

2 Related Work

Rotation Specification: 3D rotations can be specified by many techniques. The trackball metaphor [8] is one of the oldest and likely most popular techniques. 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. While simple to implement and use, trackball rotation does not allow precise control of the actual axis around which one rotates, as this axis constantly changes while the user moves the pointer [2,51]. Several usability studies of trackball and alternative 3D rotation mechanisms explain these limitations in detail [17,23,27,34]. Several refinements of the original trackball [8] were proposed to address these [24,38]. In particular, Henriksen *et al.* [21] formally analyze the trackball's principle and its limitations and also propose improvements which address some, but not all, limitations. At the other extreme, world-coordinate-axis rotations allow rotating a 3D scene around the *x*, *y*, or *z* axes [26,51]. The rotation axis and rotation angle are chosen by simple click-and-drag gestures in the viewport. This works best when the scene is already *pre-aligned* with a world axis, so that rotating around that axis yields meaningful viewpoints.

Pre-alignment of 3D models is a common preprocessing stage in visualization [7]. Principal Component Analysis (PCA) does this by computing a 3D shape's eigenvectors $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, ordered by their eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$, so that the coordinate system $\{\mathbf{e}_i\}$ is right-handed. Next, the shape is aligned with the viewing coordinate system (x, y, z) by a simple 3D rotation around the shape's barycenter [28,42]. Yet, pre-alignment is not effective when the scene does not have a clear main axis (λ_1 close to λ_2) or when the major eigenvector does not match the rotation axis desired by the user.

3D rotations can be specified by classical (mouse-and-keyboard) [51] but also touch interfaces. Yu *et al.* [49] present a direct-touch exploration technique for 3D scenes called Frame Interaction with 3D space (FI3D). Guo *et al.* [18] extend FI3D with constrained rotation, trackball rotation, and rotation around a user-defined center. [48] used trackball interaction to control rotation around two world axes by mapping it to single-touch interaction. Hancock *et al.* [19,20] use two or three touch input to manipulate 3D shapes on touch tables and, in this context, highlighted the challenge of specifying 3D rotations. All above works stress the need for *simple* rotation-specification mechanisms using a minimal number of touch points and/or keyboard controls.

Medial Descriptors: Medial descriptors, also known as skeletons, are used for decades to capture the symmetry structure of shapes [5,39]. 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* [32] of skeletal point $\mathbf{x}$ and DT_Ω is the distance transform [10,37] of skeletal point $\mathbf{x}$, defined as

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

These feature points define the so-called *feature transform* [22,41]

$$FT_{\Omega}(\mathbf{x} \in \Omega) = \underset{\mathbf{y} \in \partial\Omega}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{y}\|, \tag{3}$$

which gives, for each point $\mathbf{x}$ in a shape Ω, its set of feature points on $\partial\Omega$, or contact points with $\partial\Omega$ of the maximally inscribed disk in Ω centered at $\mathbf{x}$.

Many methods compute skeletons of 2D shapes, described as either polyline contours [33] or binary images [10,15,16,46]. State-of-the-art methods *regularize* the skeleton by removing its so-called spurious branches caused by small noise perturbations of the boundary $\partial\Omega$, which bring no added value, but only complicate further usage of the skeleton. Regularization typically defines a so-called *importance* $\rho(\mathbf{x}) \in \mathbb{R}^{+}|\mathbf{x} \in S_{\Omega}$ which is low on noise branches and high elsewhere on S_{Ω}. Several authors [10,15,16,33,46] set ρ to the length of the shortest path along $\partial\Omega$ between the two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$. Upper thresholding ρ by a sufficiently high value removes noise branches. Importance regularization can be efficiently implemented on the GPU [14] using fast distance transform computation [6]. Overall, 2D skeletonization can be seen, from a practical perspective, as a solved problem.

In 3D, two skeleton types exist [41]: *Surface skeletons*, defined by Eq. 1 for $\Omega \subset \mathbb{R}^3$, consist of complex intersecting manifolds with boundary, and hence are hard to compute and utilize [41]. *Curve skeletons* are curve-sets in $\mathbb{R}^3$ that locally capture the tubular symmetry of shapes [9]. They are structurally much simpler than surface skeletons and enable many applications such as shape segmentation [36] and animation [4]. Yet, they still cannot be computed in real time, and require a well-cured definition of Ω as a watertight, non-self-intersecting, fine mesh [40] or a high-resolution voxel volume [15,35].

Kustra *et al.* [29] and Livesu *et al.* [31] address the above challenges of 3D curve-skeleton computation by using an *image based* approach. They compute an approximate 3D curve skeleton from 2D skeletons extracted from multiple 2D views of a shape. While far simpler and also more robust than true 3D skeleton extraction, such methods need hundreds of views and cannot be run at interactive rates. Our proposal also uses an image-space skeleton computation, but uses different, simpler, heuristics than [29,31] to estimate 3D depth, and a single view, thereby achieving the speed required for interactivity.

3 Proposed Method

We construct a 3D rotation in five steps (Fig. 1). We start by loading the scene of interest – any arbitrary collection of 3D primitives, with no constraints on topology or sampling resolution – into the viewer (a). Next, the user can employ any mechanisms offered by the viewer, *e.g.* trackball rotation, zoom, or pan, to choose a *viewpoint of interest*, from which the 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 – for the sake of illustration – we want to rotate to examine the horse from different angles.

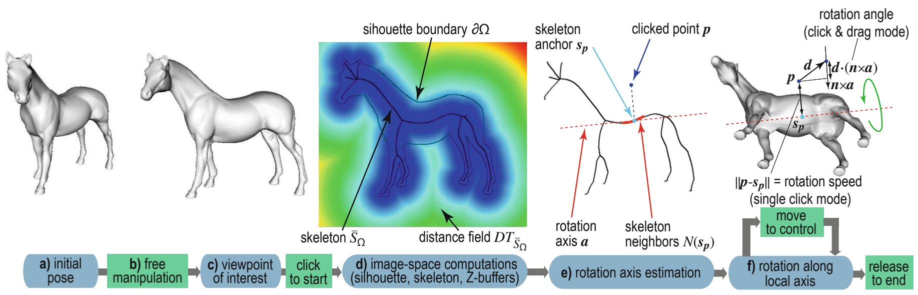

sihouette boundary ∂Ω
skeleton anchor s_p
clicked point p
rotation angle (click & drag mode)
d
$d\cdot(n\times a)$
p
$n\times a$
s_p
$\|p-s_p\|$ = rotation speed (single click mode)
skeleton $\bar{S}_\Omega$
distance field $DT_{\bar{S}_\Omega}$
rotation axis a
skeleton neighbors $N(s_p)$
move to control
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 rotation pipeline with tool states (blue) and user actions (green). Image taken from [50]. (Color figure online)

3.1 Rotation Axis Computation

From the above-mentioned initial viewpoint, we compute the rotation axis by performing three image-space operations, denoted as A, B, and C next.

A. Silhouette Extraction: This is the first operation in Fig. 1, step (d). We render the shape with Z buffering on and using the *GL_LESS* OpenGL depth-test. Let Ω_{near} be the resulting Z buffer. We next find the silhouette Ω of the 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 (Eq. 1) by the method in [46] (Fig. 1, step (d)). To eliminate spurious skeletal branches caused by small-scale noise along $\partial\Omega$, we regularize S_Ω by the salience-based metric in [43]. This regularization works as follows – see also the sketch in Fig. 2c. For every point $\mathbf{x} \in S_\Omega$ of the full skeleton delivered by Eq. 1, we first compute the importance ρ [46], *i.e.*, the shortest path along $\partial\Omega$ between the two feature points of $\mathbf{x}$ (see also Sect. 2). This path is marked red in Fig. 2c. As shown in [15,41,46], and outlined in Sect. 2, 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 which is captured (described) by $\mathbf{x}$.

We next define the *salience* of skeletal point $\mathbf{x}$ as

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

that is, the importance ρ normalized by the skeletal point's distance to boundary. As shown in [43], σ is *overall high* on skeleton branches caused by important (salient) cusps of $\partial\Omega$ and *overall low* on skeleton branches caused by small-scale details (noise cusps) along $\partial\Omega$. Figure 2c shows this for a small cusp on the boundary of a 2D silhouette of a noisy 3D dino shape. As we advance in this image along the black skeleton branch into the shape's rump (going below the grey area in the picture), ρ stays constant, but the distance to boundary DT_Ω increases, causing σ to decrease. Hence, we can regularize S_Ω simply by removing all its pixels having a salience value lower than a fixed threshold σ_0. Following [43], we set $\sigma_0 = 1$. Figure 2 illustrates this regularization by showing the raw skeleton S_Ω and its regularized version

$$\overline{S}_\Omega = \{\mathbf{x} \in S_\Omega | \sigma(\mathbf{x}) \geq \sigma_0\} \tag{5}$$

for the noisy dino shape. Salience regularization (Fig. 2b) removes all spurious branches created by boundary noise, but leaves the main skeleton branches, corresponding to the animal's limbs, rump, and tail, intact. Images (d–g) in the figure show the silhouette Ω, importance ρ, distance transform DT_Ω, and salience σ for a zoom-in area around the shape's head, for better insight. Looking carefully at image (e), we see that ρ has non-zero values also outside the main skeleton branch corresponding to the animal's neck, visible as light-blue pixels. While such details may look insignificant, they are crucial: Thresholding ρ by too low values – the alternative regularization to our proposal – keeps many spurious skeletal branches, see the red inset in Fig. 2a. In contrast, σ is practically zero outside the neck branch (Fig. 2g). So, thresholding σ by $\sigma_0 = 1$ yields a clean skeleton, see the red inset in Fig. 2b. Salience regularization is simple and automatic to use, requiring no free parameters, and hence preferable to ρ regularization – which requires careful setting of the threshold for ρ – or to any other skeleton regularization we are aware of. For further details on salience regularization, we refer to [43] and also its public implementation [44].

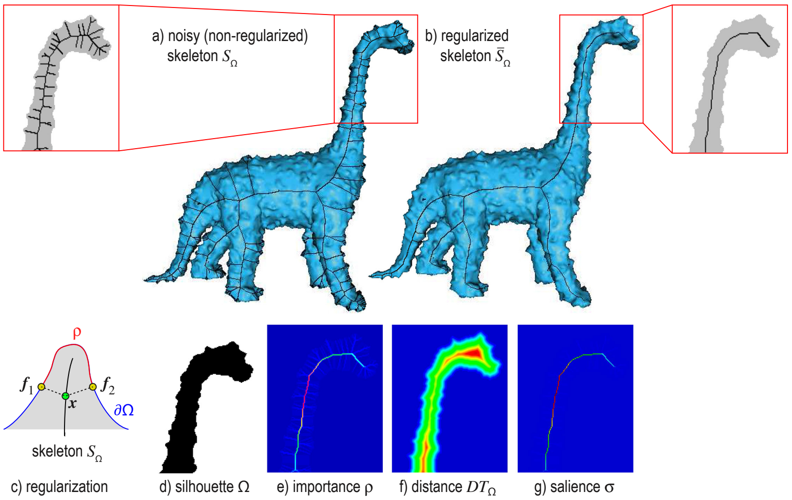


Fig. 2. Raw skeleton S_Ω with (a) noise-induced branches and (b) salience-based regularized skeleton $\overline{S}_\Omega$. c) Principle of salience regularization. (d–g) Details of silhouette, importance, distance transform, and salience values for the noisy dino's head.

C. Rotation Axis Computation: This is step (e) in Fig. 1. Let $\mathbf{p}$ be the pixel under the user-controlled 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 (Eq. 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 fixed 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 set the depth $\mathbf{q}_z$ as the average of $\Omega_{far}(\mathbf{q})$ and $\Omega_{near}(\mathbf{q})$. Here, Ω_{near} is the Z buffer of the scene rendered as described in step A above; and Ω_{far} is the Z buffer of the scene rendered as before, but with front-face culling on, *i.e.*, the depth of the *nearest* backfacing polygons to the view plane.

Figure 3 shows how this works. The user clicks above the horse's rump and drags the pointer upwards (a). Image (b) shows the resulting rotation. As visible in the inset in (a), 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. To better understand this, the image left to Fig. 3a shows the horse rendered transparently, seen from above. The depth values in Ω_{near} and Ω_{far} are shown in green, respectively blue. The skeleton depth values (red) are the average of these. Note that, when the rotation ends, the new silhouette skeleton does *not* match the rotation axis – see inset in (b). This is normal and expected. If the user wants to start a new rotation from (b), then the 2D skeleton from this image will be used to compute a new, matching, rotation axis.

Next, we consider a case of overlapping shape parts (Fig. 3c). The user clicks left to the horse's left-front leg, which overlaps the right-front one, and drags the pointer to the right. Image (d) shows the resulting rotation. The rotation axis (red) is centered inside the left-front leg. In this case, $\Omega_{far}(\mathbf{q})$ contains the Z values of 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. The image left to Fig. 3c clarifies this by showing the horse from above and the respective depth values in Ω_{near} (green) and Ω_{far} (blue).

Separately, we handle non-watertight surfaces as follows: If $\Omega_{far}(\mathbf{q})$ contains the default Z value (one), this means there's no backfacing surface under a given pixel $\mathbf{q}$, so the scene is not watertight at $\mathbf{q}$. We then set $\mathbf{q}_z$ to $\Omega_{near}(\mathbf{q})$.

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 propose three interactive mechanisms to control the rotation (Fig. 1), step (f)):

- **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 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;

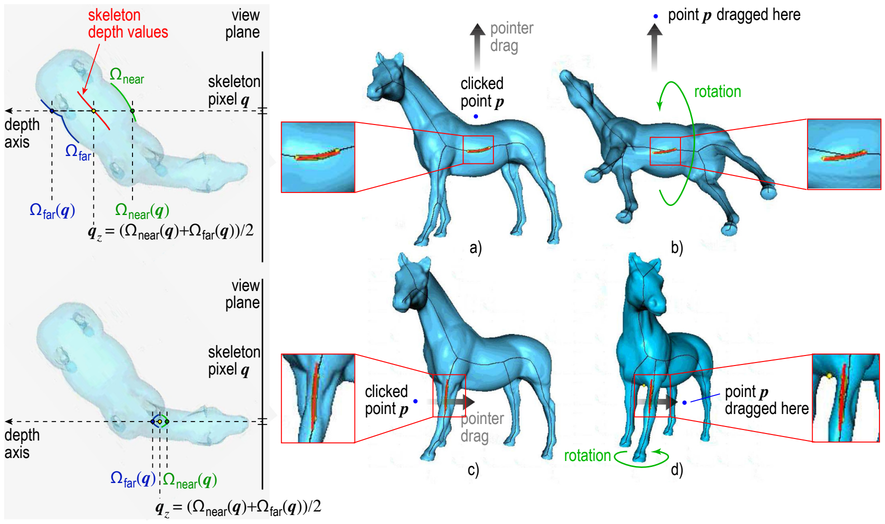


Fig. 3. Depth estimation of rotation axis for (a, b) non-overlapping part and (c, d) overlapping parts. In both cases, the rotation axis (red) is nicely centered in the shape. See Sect. 3.1. (Color figure online)

– **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). 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 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 yields 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 for freely specifiable 3D axes, and also applicable to contexts where no second button or modifier keys are available, *e.g.*, touch screens.

3.3 Improvements of Basic Method

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

Zoom Level: A first issue regards computing the scene's 2D silhouette Ω (Sect. 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. Figure 4a shows this for a zoomed-in view of the horse model, with the above-mentioned border parts marked purple. 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 standard trackball-based rotation.

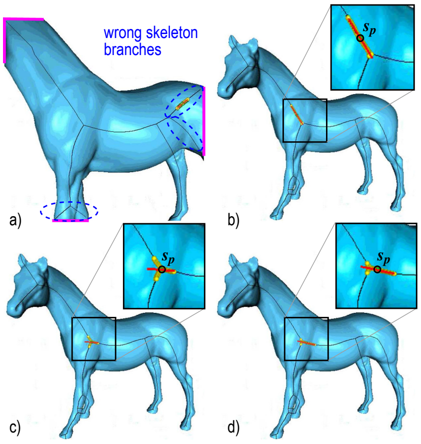


Fig. 4. Two problems of estimating rotation axes from skeletons. (a) Zoomed-in scene. Anchor points close to (c), respectively farther from (b, d) a skeleton junction. See Sect. 3.3. Image taken from [50]. (Color figure online)

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 (Sect. 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 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. When $\mathbf{s}_p$ is very close to a junction (Fig. 4c), $N(\mathbf{s}_p)$ contains points from all three meeting skeletal 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 [13]. 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 starts 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 Sect. 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, *e.g.* the horse's rump in Fig. 5b, it is intuitive to select $\mathbf{p}$ even quite far away from the silhouette. This is the case 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.

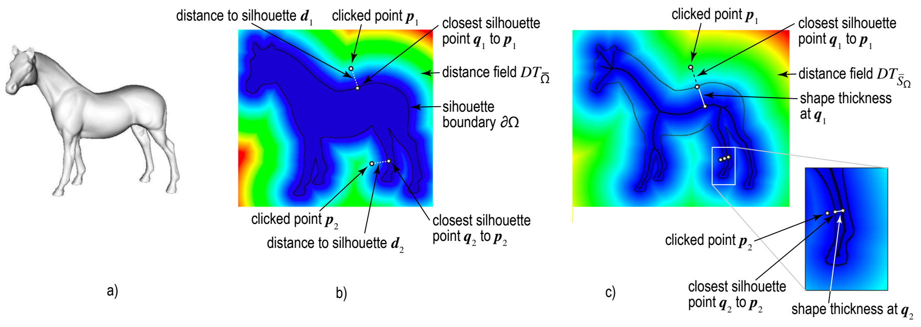

distance to silhouette d_1
clicked point p_1
closest silhouette point q_1 to p_1
distance field $DT_{\overline{\Omega}}$
sihouette boundary $\partial\Omega$
clicked point p_2
distance to silhouette d_2
closest silhouette point q_2 to p_2
clicked point p_1
closest silhouette point q_1 to p_1
distance 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 $\mathbf{p}_2$. Image taken from [50].

We model this by setting d_{max} to the local *shape thickness* (Fig. 5c). We estimate thickness as follows: We find the closest point on the silhouette boundary $\partial\Omega$ to the clicked point $\mathbf{p}$ as $\mathbf{q} = FT_{\overline{\Omega}}(\mathbf{p})$. The shape thickness at $\mathbf{q}$ is the distance to the skeleton, *i.e.*, $DT_{\overline{S}_\Omega}(\mathbf{q})$. This is the 2D equivalent of the more general 3D-shape-thickness estimation proposed in [45]. In Fig. 5c, the point $\mathbf{p}_1$ is the farthest clickable point around $\mathbf{q}_1$ to the silhouette that allows starting 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 $\mathbf{q}_2$ must, however, be much closer to the silhouette (Fig. 5c), since here the local shape thickness (distance d_{max} from $\mathbf{p}_2$ to $\mathbf{q}_2$) is smaller.

4 Formative Evaluation

To evaluate our method, we conducted first a *formative* evaluation. In this evaluation, only the authors of this work and a few other researchers, familiar with 3D interactive data visualization, were involved. This evaluation aimed at (a) verifying how the skeleton-based rotation practically works on a number of different 3D shapes; and (b) eliciting preliminary observations from the subjects to construct next a more in-depth evaluation study. We next present the results of this first evaluation phase. Section 5 details the second-phase evaluation designed using these findings.

Figure 6 shows our 3D skeleton-based rotation applied to two 3D mesh models. For extra insights, we recommend watching the demonstration videos [47]. First, we consider a 3D mesh model of a human hand 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 wrist, and start rotation. Images (g–j) show the resulting rotation.

Figure 6(k-ad) show a more complex ship shape. This mesh contains multiple self-intersecting and/or disconnected parts, some very thin (sails, mast, ropes) [30]. Computing a 3D skeleton for this shape is extremely hard or even impossible, as Eq. 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 s.

Figure 7 shows a different dataset type – a 3D point cloud that models a collision simulation between the Milky Way and the nearby Andromeda Galaxy [11,25]. 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, but rather 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

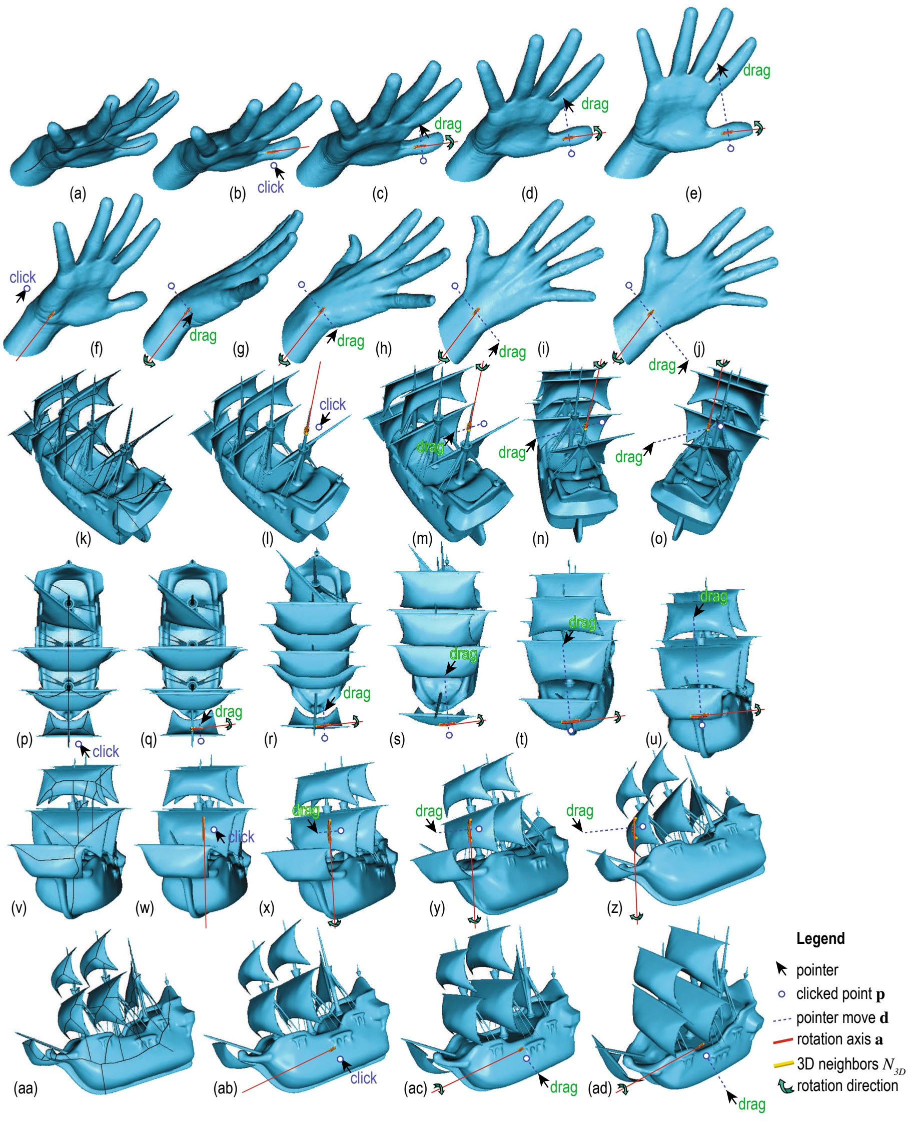


Fig. 6. Examples of two rotations (a–e), (f–j) for the hand shape and four rotations (k–o), (p–u), (v–z), (aa–ad) for the ship model. Image taken from [50].

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 view-point 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

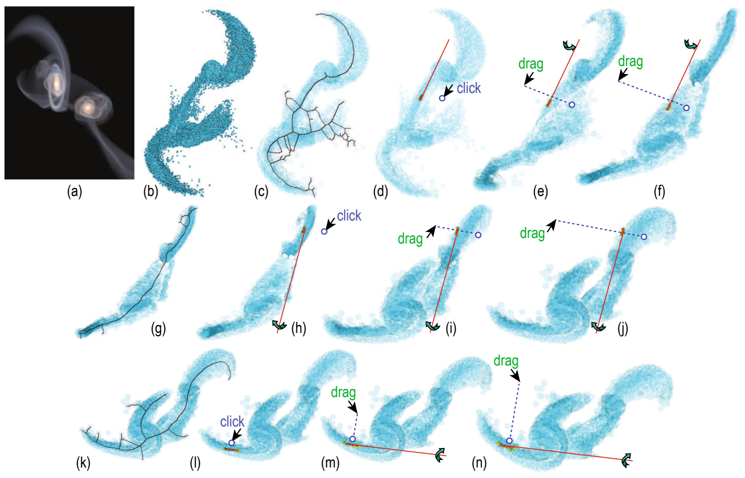


Fig. 7. Exploration of astronomical point cloud dataset. (a) Volume-rendered overview [25]. Rotations around three 3D axes (b–f), (g–j), (k–n). Image taken from [50].

structure. As for the earlier examples, executing these three rotations took roughly 15 s. 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.

We gathered several insights during our formative evaluation by free-form discussions with the participants – that is, without following a strict evaluation protocol based on tasks and quantitative responses. We summarize below the most important ones:

– Skeleton rotation works quite well for relatively *small* changes of viewpoint; more involved changes require decomposing the desired rotation into a set of small-size changes and careful selection of their respective rotation axes;
– Skeleton rotations seems to be most effective for *precise* rotations, in contrast to typical trackball usage, which works well for larger, but less precise, viewpoint changes;
– All participants stated that they *believe* that skeletons allow them to perform certain types of rotation easier than if they had used the trackball for the same tasks. However, they all mentioned that they do not feel that skeletons can *replace* a trackball. Rather, they believe that a free combination of both to be most effective. Since they could only use the skeleton rotation (in our evaluation), they do not know whether (or when) this tool works better than a trackball;
– All participants agreed that *measuring* the added-value of skeleton rotation is very important for its adoption.

5 Detailed Evaluation: User Study

The formative evaluation (Sect. 4) outlined that there is perceived added-value in the skeleton rotation tool, but this value needs to be actually measured before users would consider adopting the tool – either standalone or in combination with trackball. To deepen our understanding of how skeleton-based rotation works, and to answer the above questions, we designed and conducted a more extensive user evaluation. We next describe the design, execution, and analysis of the results of this evaluation.

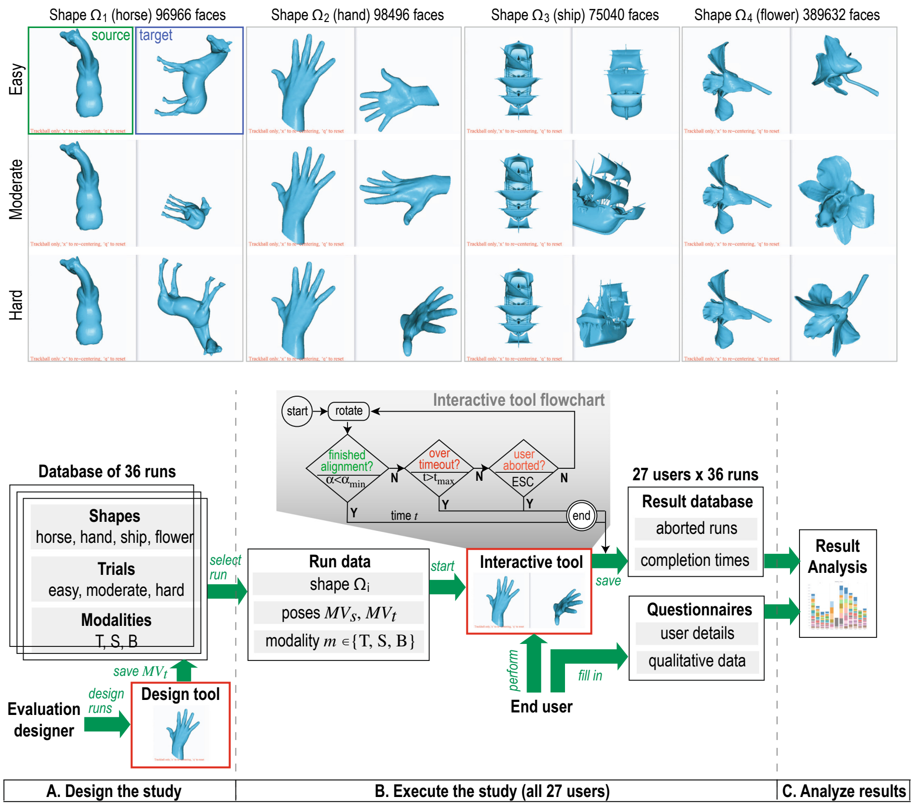


Fig. 8. Top: user evaluation showing the 12 trials for one modality (Sect. 5.1). Each trial consists of a source window in which the user interacts to align the shape to match the target window. Bottom: execution of end-to-end user evaluation. The use of our interactive tool in both design and evaluation modes is shown in red (Sect. 5.2). (Color figure online)

5.1 Evaluation Design

Tool: To assess how the skeleton rotation modality compares with the trackball modality, we designed an experiment supported by an interactive *tool*. The tool has two windows: The *target* window shows a 3D shape viewed from a viewpoint (pose) that is preselected by the evaluation designer. No interaction is allowed in this window. The *source* window shows the same shape, which can be freely manipulated by the user via the skeleton (S), the trackball (T), or both tools (B), activated via the left, respectively right, mouse buttons. Both windows have the same resolution (512^2 pixels), use the same lighting and rendering parameters, and have a fixed position on the computer screen, to simplify usage during the experiment that invokes multiple runs of the tool. Besides rotation, the tool also allows panning and zooming. We also added an option to automatically zoom out to show the full extent of a shape. This eliminates the issues described in Sect. 3.3, *i.e.*, manipulations that move part of the shape outside the window. When in S mode, the tool shows the silhouette skeleton (black), nearest skeleton points (yellow), and estimated rotation axis (red) as explained earlier in Sect. 3.1 and shown *e.g.* in Fig. 3. The user can interactively tune the simplification level of the skeleton via the '+' and '$-$' keys, to show more or fewer branches from which to select a suitable rotation axis (*cf* Fig. 2).

Figure 8 (central inset) shows a flowchart of the tool's operation, which we detail next. Participants are asked to use the tool with each modality in turn (S, T, B) to align the source with the target. The tool continuously computes, after each motion of the mouse pointer, the value

$$\alpha = \arccos\left(\frac{Tr(MV_s \cdot MV_t^T) - 1}{2}\right), \tag{6}$$

where MV_s and MV_t are the 3×3 OpenGL rotation matrices (ignoring, thus, translation and scaling) corresponding to the pose of the shape in the source and the target, respectively; Tr is the matrix trace operator; and T denotes matrix transposition. The value $\alpha \in [0, 180]$ is the smallest rotation (around any axis) needed to obtain the target pose from the source pose [3]. Note that Eq. 6 is sensitive to mirroring, which is desired, since rotations *cannot* cause mirroring. Alignment is considered *completed* when $\alpha < \alpha_{min}$; in practice, we set $\alpha_{min} = 15°$. Also, note that Eq. 6 only checks for *rotation*, and not scaling or panning, differences. This makes sense, since the tested modalities S, T, B control rotation only; scaling (zooming) and panning, though allowed to help users to inspect shapes, are not part of our evaluation, and perform identically with S, T, and B. During manipulation, the tool continuously displays the current value of α. This shows users how far away they are from the target rotation MV_t, thus, from completing a task. This feedback is useful when visual comparison of the source and target poses is hard to do.

Shapes: We use the alignment tool to evaluate the performance of the S, T, and B modalities on $N = 4$ *shapes* Ω_i, $1 \leq i \leq N$, shown in Fig. 8(top). Shapes were selected so as to be familiar, have a structure that exposes potential local-rotation axes, and have geometric complexity ranging from simple (horse, hand) to complex (ship). The flower shape is of lower complexity than the ship; however, its manifold structure makes it particularly hard to understand and manipulate, since it looks quite similar from many viewpoints. All shapes use identical material properties and no opacity or textures, to favor uniform evaluation. We excluded the more complicated point-cloud shape (Fig. 7) used during formative evaluation (Sect. 4) since no more than five of our recruited subjects had the technical background needed to understand what such data means in the first place.

Task Difficulties: For each shape, we use three target poses MV_t to capture three levels of difficulty of the alignment task:

– *Easy:* Alignment can be done by typically one or two manipulations, such as a rotation around one of the x or y window axes, or a rotation around a clearly-visible symmetry axis of the shape). For example, the blue-framed target in Fig. 8 can be obtained from the green-framed pose (left to it) by a single counterclockwise rotation of the horse with 90° around the y axis or, alternatively, the rump's skeleton;
– *Hard:* Alignment requires multiple rotations around many different rotation axes; it is not easy to see, from the source and target, which would be these axes;
– *Intermediate:* Alignment difficulty is gauged as between the above two extremes.

We call next the combination of shape Ω_i and start-and-end pose (MV_s, MV_t) a *trial*. Using multiple-difficulty trials aims to model tasks of different complexity. Trial difficulty was assessed by one of the authors (who also designed the actual poses MV_t) and agreed upon by the others by independent testing. We verified that all three modalities could accomplish all trials within a time t lower than a predefined timeout $t_{max} = 120$ s.

Figure 8(top) shows the source (left window in each window-pair) and target (right window in same pair) windows for the 12 trials spanning the 4 shapes using the T modality. Source windows show the currently-enabled modality in red text, to remind users how they can interact. We see, for instance, that the *easy* trial would require, in S mode, a simple 90° rotation around the y axis (in T mode) for the ship model, or around the main skeleton branch passing through the horse's rump for the horse model. In contrast, the *hard* task requires several incremental rotations for all modalities. The 12 trials use *identical* initial poses MV_s and target poses MV_t. That is, the user is asked to perform, for each shape, the same alignment $MV_s \rightarrow MV_t$ using all three modalities, thus ensuring that only the target pose (endpoint of manipulation) and, of course, the used modality, affect the measured execution time.

In total, we thus execute 12 trials $\times$ 3 modalities $=$ 36 *runs*. For each run, we record the time needed for the user to complete it. If the user fails to perform the alignment within the allowed timeout, the run is considered *failed* and the user moves automatically to the next run. Users can at any time (a) abort a run by pressing 'ESC' to move to the next run; this helped impatient users who did not grasp how to perform a given alignment task and did not want to wait until the timeout; (b) abort the entire evaluation, if something goes entirely wrong; and (c) reset the viewpoint to the initial one (MV_s), to 'undo' all manipulations performed so far if these are deemed unproductive.

Pose Design: The different target poses MV_t were designed in advance by us by using the S and T tools – intermixed – to freely change the shape's pose until obtaining the desired target poses, and stored, as explained, as 3×3 OpenGL rotation matrices.

5.2 Evaluation Execution

Subjects: Twenty-seven persons took part in the evaluation. they self-report ages of 9 to 64 years (median: 24, average: 26.9); and gender being male (16) and female (11), see Fig. 9b. To gauge their experience with 3D manipulation, we asked them to report how many times a year they used 3D games and/or 3D design software. Both categories are reported in Fig. 9b as '3D software usage'. Results show a median of 30 times, with

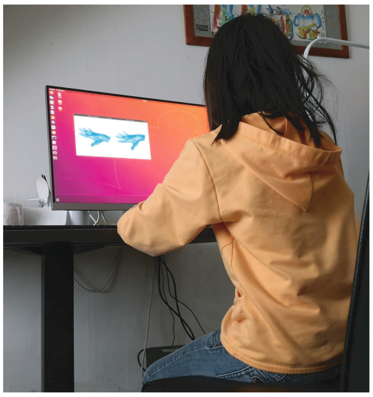
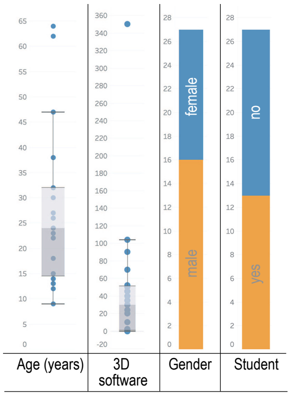



Fig. 9. a) Setup employed during the user evaluation. b) Self-reported characteristics of the experiment participants. See Sect. 5.2.

the minimum being zero (never) and the maximum being basically every day. From these data we conclude that most participants should have a good practical mastery of 3D manipulation. Fromf the 27 participants, 13 were students in fields as diverse as Computer Science, social science, medicine, economy and society, and mathematics; the other 14 were primary or secondary school pupils (6) or employed in various liberal professions (8). All participants reported no color blindness issues. All except one were right-handed. They all reside in the Netherlands or Belgium. Communication during the training and experiment was done in the native language of each participant by a (near-)native speaker. For participants with limited English proficiency, all English material (tutorial, questionnaires) was transcribed by the trainer.

Workflow: Participants followed the evaluation workflow showed in Fig. 8(bottom). First, we created the information needed to execute the 36 runs (Fig. 8(bottom, A)), as explained in Sect. 5.1. Next, participants were given access, prior to the actual experiment, to an web tutorial which describes both S and T tools in general, and also allows users to practice with these tools by running the actual application to execute some simple alignment tasks. No statistics were collected from this intake phase. After intake, users asked if they felt interested in, and able to follow, the tutorial. This intake acted as a simple filter to separate users with interest in the evaluation (and potential ability to do it) from the rest, so as to minimize subsequent effort. Seven persons dropped from the process due to lack of general computer skills (1 user), one too young (6 years), one too old (82 years), and four due to technical problems related to remote-deployment of the tool. These persons are not included in any of the statistics further on, nor in Fig. 9b.

Next, a trainer (role filled by different co-authors) took part in a *controlled session* where they explained to either individual participants or, when social distancing rules due to the Corona pandemy were not applicable, to groups of participants how the tool works and also illustrated it live. The aim of this phase was to refine the knowledge disseminated by the web tutorial and confirm that participants understood well the evaluation process and tooling. Participants and trainers used Linux-based PCs (16 to 32

GB RAM) with recent NVidia cards, wide screens, and a classical two-button mouse. To maximize focus on the experiment, no application was run on screen during the evaluation besides the two-window tool described in Sect. 5.1. Training took both the in-person form (with trainer and user(s) physically together), and via TeamViewer or Skype screen sharing, when social distancing rules mandated separation. Training took between 20 and 40 min per user, and was done until users told that they were confident to use the tool to manipulate both a simple model and a complex one via all three modalities (S, T, B). During this phase, we also verified that the tool runs at real-time framerates on the users' computers so as to eliminate confusing effects due to interaction lag; and that the users did not experience any difficulty in using the keyboard shortcuts outlined in Sect. 5.1.

After training, and confirmation by participants that they understand the evaluation tool and tasks to be done, participants started executing the 36 runs (Fig. 8(bottom, B). They could pause between runs as desired but not change the orders of the runs. Figure 9a shows the setup used during the evaluation by one of the actual participants; notice the two-window interaction tool on the screen. At the end, the results of all 36 runs – that is, either completion time or run failure (either by timeout or user abortion) – were saved in a database with no mention of the user identity. Next, users completed a questionnaire covering both personal and self-assessment data and answers to questions concerning the usability of the tool. Both types of results (timing data and questionnaires) were further analyzed (Fig. 8(bottom, C)), as described in Sect. 5.3.

5.3 Analysis of Results

We next present both a quantitative analysis of the timing results and an analysis of the qualitative data collected via questionnaires.

Analysis of Timing Results. A most relevant question is: How did performance (measured in completion time and/or number of aborted runs) depend on the interaction *modality* and *shape*? Figure 10a shows the average completion time, for the *successful* runs, aggregated (over all users) per modality and next per shape. User identities are categorically color-coded for ease of reading the figure. Median and interquartile ranges for each modality are shown by black lines, respectively gray bands. We see that the S modality is significantly slower than T and S. However, the B modality is faster than T, both as median and interquartile range, and also for each specific shape. This is an interesting observation, as it suggests that, in B mode, users did gain time by using S only for some specific manipulations for which T was hard to use. A likely explanation for this is that the B modality was always used last during the trials. Hence, when in B mode, users could discover the situations when S outperformed T, and switch to S in those cases to gain time. We will analyze thys hypothesis further below.

Figure 10b shows the number of *failed* runs per modality, shape, and user. These are largest for the S modality. This tells again that S cannot be used *alone* as a general-purpose manipulation tool. If we combine this insight with the total times per shape (Fig. 10a), we see that the perceived difficulty of the task varies significantly over both shapes and modalities: T and S behave quite similarly, with *horse* and *ship* being easier

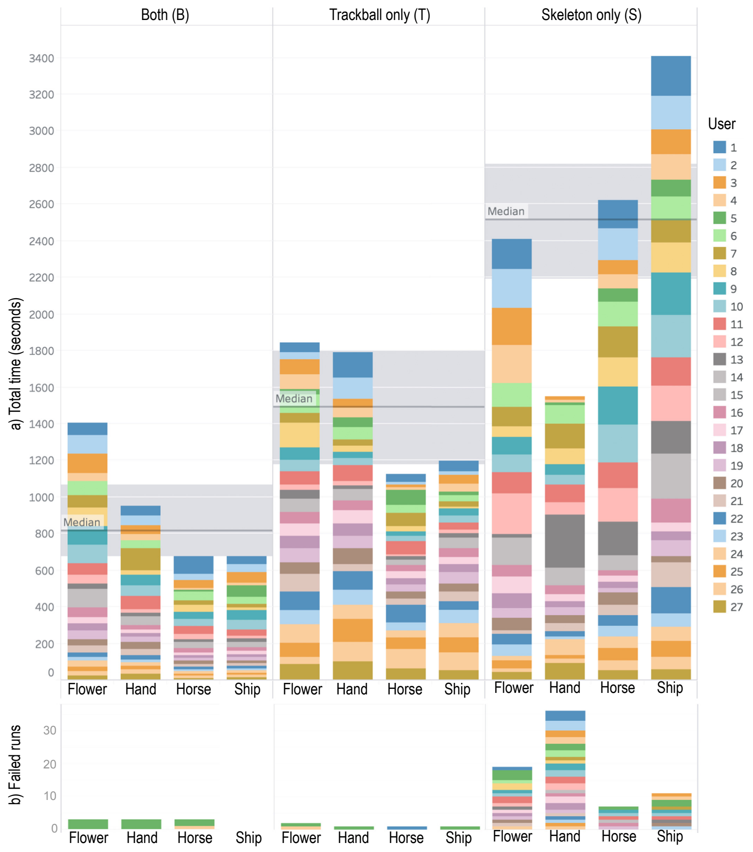


Fig. 10. Completion time (a) and number of failed runs (b) per modality and shape, all users. See Sect. 5.2.

to handle and *flower* being the hardest. In contrast, *hand* seems to be the hardest to handle by the S modality, as it has most aborted runs. Upon a closer analysis, we found that the pose used by the 'hard' trial for *hand* (see the respective image in Fig. 8(top)) is quite easy to achieve with T (and thus also B), but quite difficult to obtain using S, since it implies, at several points, performing a rotation around an axis orthogonal to the hand's palm, for which no skeleton line exists in the silhouette. The second-hardest shape for S is *ship*. Analyzing the users' detailed feedback showed us that *ship*'s complex geometry produces a wealth of potential rotation axes with quite different angles, which makes the users' choice (of the optimal rotation axis) hard. This happens far less for the other simpler-structure shapes. Separately, Fig. 10b shows that the number of aborted runs in B mode is far lower than that in S mode, being practically the same as for T mode. This, and the fact that B mode is fastest, reinforces our hypothesis that users employ the S tool in B mode only for very *specific* manipulations and revert to T for all other operations. Hence, S works best as a complement, not a replacement, of T.

Figure 11a introduces additional information in the analysis by showing how the average times vary over the three task *difficulty* levels (easy, moderate, hard, see Sect. 5.1). For all shapes and modalities, the task labeled easy by us is, indeed, completed the fastest. The other two difficulty levels are, however, not significantly different in execution times. We also see that effort (time) is distributed relatively uniformly over all difficulty levels for all shapes and modalities. This indicates that there is no 'outlier' task or shape in our experiment that would strongly bias our evaluation's insights.

Finally, we examine the data from a *user-centric* perspective. Figure 11b shows the total time per user, split per modality, with the fastest users at the right and the slowest at the left. We see a quite large spread in performance, the fastest user being roughly 2.5 times faster than the slowest one. We see that the T modality does not explain the big speed difference – the red bars' sizes do not correlate with the total time. In contrast, the blue bars show an increase when scanning the chart right-to-left, at the 8^{th} leftmost bar – meaning that the 8 slowest users needed clearly more time to use the B modality as opposed to the remaining 19 users. Scanning the graph right-to-left along its orange bars shows a *strongly* increasing bar-size. That is, the main factor differentiating slow from fast users is their skill in using the *S tool*. We hypothesized that this skill has to do with the users' familiarity with 3D manipulation tools. To examine this, we show a scatterplot of the average time per user (all trials, all shapes) *vs* the user's self-reported number of days per year that one uses 3D computer games or 3D creation software (Fig. 11c). All points in the plot reside in the lower range of the y axis, *i.e.*, all users report under 100 days/year of 3D tool usage, except user 12 who indicated 3D gaming daily. The computed correlation line shown in the figure ($R^2 = 0.0022$, $p = 0.813$) indicates a negligible inverse correlation of average time with 3D software usage. Hence, our hypothesis is not confirmed. The question what determines the variability in users' average completion times is still open.

Questionnaire Results. As mentioned at the beginning of Sect. 5.2, users completed a questionnaire following the experiment. They were asked to answer 13 questions concerning their experience with each of the three modalities (T, S, B) using a 7-point Likert scale S (1 = strongly disagree, 2 = disagree, 3 = disagree somewhat, 4 = no opinion, 5 = agree somewhat, 6 = agree, 7 = strongly agree). An extra question (Q14) asked which of the three modalities users prefer *overall*. Figure 12(bottom) shows these 14 questions. Here, 'tool' refers to the modality being evaluated. Following earlier studies that highlight that user *satisfaction* is not the same as user *efficiency* or *effectiveness* when using interactive tools [17, 34], we included questions that aim to cover all these aspects. Users could also input free text to comment on their perceived advantages and limitations of all three modalities or any other remarks.

Figure 12(top) shows the aggregated answers for Q1..Q13 for each of the three modalities with box-and-whisker plots (box shows the interquartile range; whiskers show data within 1.5 times this range). We see that the S modality ranks, overall, worse than the T modality, except for accuracy (Q5). Accuracy (Q5) can be explained by the fact that users need to control a *single* degree of freedom with S – the rotation angle – but two degrees of freedom with T. In other words, once a suitable rotation axes is chosen, S allows one to precisely specify the rotation angle around this axis. We also see

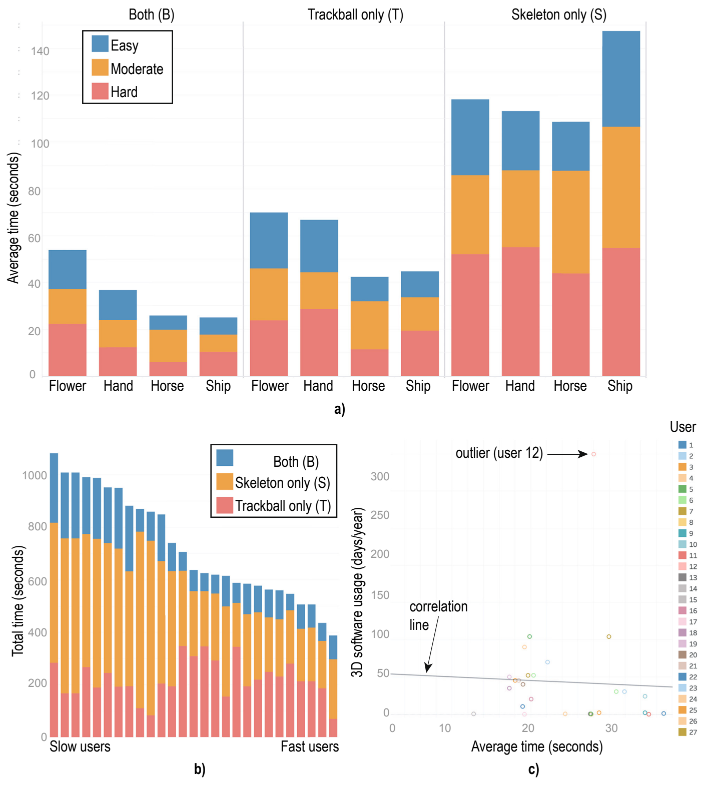
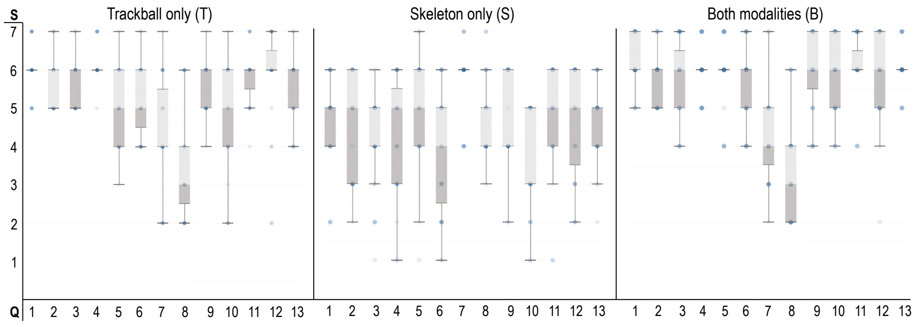


Fig. 11. a) Average completion time per difficulty levels, modality, and shape. b) Total time for all users, from slowest to fastest, split per modality. c) Correlation of average time (all runs) with users' frequency of 3D software usage. See Sect. 5.2.

that S helps completing the task less often than T (Q10), which matches the failure rates shown in Fig. 10b. However, the B modality ranks in nearly all aspects better than both T and S. This supports our hypothesis that S best *complements*, rather than replaces, T. An interesting finding are the scores for Q8 and Q6, which show that B was perceived as less tiring to use, and needing fewer steps to accomplish the task respectively, than both T and S. This matches the results in Fig. 10a that show that B is faster than both T and S – thus, arguably less tiring to use. For Q14, 22 of the 27 users stated that they prefer B overall, while the remaining 5 users preferred T, with none mentioning S as the highest-preference tool. As for the previous findings, this strongly supports our hypothesis that the S and T modalities work best when combined.

From the free text that captures the user's comments on the perceived advantages and limitations of all three modalities, we could distil several salient points. For space constraints, we list only a few below:

– **Trackball (T):** Several users praised T for being "easy to use". However, users also complained about trackball being imprecise for performing fine adjustments;



Q1	The tool met my needs for performing the alignment task
Q2	The tool worked as expected (after following the training)
Q3	The tool helped me be more effective than the two other tools
Q4	The tool was easy to use
Q5	The tool was accurate
Q6	The tool requires the fewest steps (compared to the other two) to accomplish my goals
Q7	I felt that I have to think carefully to get a good result with this tool
Q8	The tool was tiring to use
Q9	Both occasional and regular users would like the tool
Q10	I can use the tool successfully every time
Q11	I learned to use the tool quickly
Q12	I easily remember how to use the tool
Q13	I am satisfied with the tool
Q14	Which tool (T, S, or B) do you overall prefer?

Fig. 12. Results of 13-point user questionnaire for the three modalities. Questions are shown below the charts. See Sect. 5.3.

- **Skeleton (S):** This modality was mentioned as better than the other two by only a few users, and specifically for the horse, hand, and flower models, because of their clear and simple skeletons, which allow one to intuitively rotate the shape around its parts ("easy to turn the hand around a finger"; "easy to turn the horse around a leg"; "S helps to turn the flower around its stem"). However, several users mentioned advantages of S when used in combination with T. These are discussed below;
- **Both (B):** Overall, this modality received the most positive comments. It was deemed the "most accurate"; and "feeling quick to use when we have two methods [to choose from]". Specifically, users noted that B is "good for doing final adjustments/fine tuning the alignment" and that "S helps T to getting the desired result easily" and "I started with T and used S for final touches". One user also commented: "I work as a graphic designer with a lot of 3D tools; I see how S helps me by providing a lot of control when rotating, and I would love to have this tool along my other manipulation tools in my software [...] but I would not use it standalone".

Summarizing the above, we see that our initial hypothesis that the S modality helps (complements) T for precision tasks is largely supported by user experience.

6 Discussion

6.1 Technical Aspects

The skeleton-based rotation method presented in Sect. 3 has the following main features:

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. Also, the skeleton tool can be directly combined (used alongside) any other interaction tool, such as trackball, with no constraints.

Reversibility: 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 back will reverse the rotation.

Scalability: Our method uses OpenGL 1.1 (primitive rendering and Z-buffer reading) plus the 2D image-based skeletonization method in [46] used to compute the skeleton S_Ω, its regularization $\overline{S}_\Omega$, and the feature transform $FT_{\overline{S}_\Omega}$. We implemented skeletonization in NVidia's CUDA and C++ to handle scenes of hundreds of thousands of polygons rendered at 1000^2 pixel resolution in a few milliseconds on a consumer-grade GPU, *e.g.* GTX 660. The skeletonization 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 [6]. The separate code of this skeletonization method is available at [1]. Implementing the two improvements presented in Sect. 3 is also computationally efficient: The skeleton's distance transform $DT_{\overline{S}_\Omega}$ is already computed during the rotation axis estimation (Sect. 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 method delivers interaction at over 100 frames-per-second on the aforementioned consumer-grade GPU. The full code of our skeleton-and-trackball manipulation tool (Sect. 5.1) is provided online [47].

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 similar view-based reconstructions of 3D curve skeletons from their 2D silhouettes [29,31], our method requires a *single* viewpoint to compute an approximate 3D curve skeleton and is two to three orders of magnitude faster.

6.2 Usability and Applicability

The evaluation described in Sect. 5 confirmed the insights elicited from the earlier formative study (Sect. 4), *i.e.* that skeleton rotation is best for precise, small-scale, final alignment touches; and that skeleton rotation best works as a *complement*, and not replacement, of trackball rotation. The latter point was supported by all types of data from our evaluation – task timing, scores assigned by users to evaluation questions, and free-form text feedback. The same data shows that users rank the combined modality (B) as better than both S and T modalities taken separately. The user scores also show that, overall, the combined modality is easy to learn and use (Fig. 12, Q2-4-8-11-12). Put together, all above support our claim of added value for the skeleton-based rotation technique.

Besides the above results, the user study also unveiled several questions which we cannot fully answer:

User Performance: There is a large variability of user performance, measured as task success rates and completion times (see Fig. 11b and related text). We cannot explain this variability by differences in the experiment setup, previous user familiarity with 3D manipulation, amount of training with the evaluated tool, or other measured factors. This variability may be due to user characteristics which the self-reported variables (Fig. 9b and related text) do not capture; to the high heterogeneity of the user population; but also due to dependent variables which we did not measure, *e.g.*, how often did users use the skeleton simplification level (Sect. 5.1) to produce suitable skeletons for generating rotation axes. Repeating the experiment with a more homogeneous population and more measured variables would help answering this question.

Applicability: An important limitation of our study is that, for the B modality, we did not measure how (much) the task was completed using each *separate* modality, *i.e.*, S and T. The formative study (Sect. 4), textual user feedback for the controlled experiment, and our observation of the users during the experiment jointly show that, in most cases involving moderate or hard tasks, trackball was *first* used to obtain a viewpoint roughly close to the target one, which was next fine-tuned using skeleton. This is fully in line with our initial design ideas (see Fig. 1 and related text) and also with earlier findings on what trackball best works for [21,27,34]. However, understanding more precisely which are the rotation types that skeleton best supports would greatly help to improve the combined modality by *e.g.* suggesting this modality to the user when it appears fit, and/or conversely, blocking this modality when it does not match the task at hand.

Study Limitations: Besides the above-mentioned aspects, our study (Sect. 5) has further limitations: It uses only four shapes that cannot capture the rich distributions of 3D shapes that need manipulation. Also, it only covers the task of rotating from an initial pose to a given final pose. Yet, manipulation is also used for free exploration and/or design actions which do not require reaching a predefined pose. It is unclear how to *quantitatively* measure the added value of interaction tools in such contexts, beyond qualitative user-satisfaction questionnaires [21]. Also, we cannot exclude learning effects between the trials that address the same task with different modalities. Finally, what is the exact added-value of all the rotation-specification improvements (Sect. 3.3) was not currently measured. Exploring all these directions is left to future work.

7 Conclusion

We proposed a method for specifying interactive rotations of 3D scenes around local rotation axes using image skeletons. We compute such axes from the skeleton of the 2D silhouette of the rendered scene, enhanced with depth information from the rendered Z buffer. Specifying such rotation axes requires a single click-and-drag gesture and no additional parameter settings. Our method is simple to implement, using distance and feature transforms provided by modern 2D skeletonization algorithms; handles 3D scenes consisting of arbitrarily complex polygon meshes (not necessarily watertight, connected, and/or of good quality) or 3D point clouds; can be integrated in any 3D viewing system that allows access to the Z buffer; and works at interactive frame-rates even for scenes of hundreds of thousands of primitives.

We measured the added value of the proposed rotation technique by a formative study (to elicit main concerns from users) followed by a controlled user study. Results showed that, when combined with trackball rotation, our method leads to better results (in terms of task completion times) and higher user satisfaction than trackball rotation alone. Also, our method is easy to learn and does not carry a significant learning or execution cost for the users, thereby not increasing the costs of using standard trackball rotation.

Several future work directions are possible. More cues can be used to infer more accurate 3D curve skeletons from image data, such as shading and depth gradients, leading to more precise rotation axes. Such data-driven cues could be also used to better control the rotation, and also suggest to the user which of the two modalities (skeleton-based or trackball rotation) are best for a given context. Separately, we aim to deploy our joint skeleton-and-trackball rotation tool on touch displays (single or multiple input) and evaluate its effectiveness in supporting domain experts to perform 3D exploration for specific applications, such as the astronomical data exploration outlined in Sect. 4.

References

1. Telea, A.: Real-time 2D skeletonization using CUDA (2019). http://www.staff.science.uu.nl/~telea001/Shapes/CUDASkel
2. Bade, R., Ritter, F., Preim, B.: Usability comparison of mouse-based interaction techniques for predictable 3D rotation. In: Butz, A., Fisher, B., Krüger, A., Olivier, P. (eds.) SG 2005. LNCS, vol. 3638, pp. 138–150. Springer, Heidelberg (2005). https://doi.org/10.1007/11536482_12
3. Belousov, B.: Difference between two rotation matrices (2016). http://www.boris-belousov.net/2016/12/01/quat-dist
4. Bian, S., Zheng, A., Chaudhry, E., You, L., Zhang, J.J.: Automatic generation of dynamic skin deformation for animated characters. Symmetry **10**(4), 89 (2018)
5. Blum, H.: A transformation for extracting new descriptors of shape. In: Models for the Perception of Speech and Visual Form, pp. 362–381. MIT Press (1967)
6. Cao, T.T., Tang, K., Mohamed, A., Tan, T.S.: Parallel banding algorithm to compute exact distance transform with the GPU. In: Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 83–90 (2010)
7. Chaouch, M., Verroust-Blondet, A.: Alignment of 3D models. Graph. Models **71**(2), 63–76 (2009)

8. Chen, M., Mountford, S., Sellen, A.: A study in interactive 3D rotation using 2D control devices. Comput. Graph Forum **22**(4), 121–129 (1998)
9. Cornea, N., Silver, D., Min, P.: Curve-skeleton properties, applications, and algorithms. IEEE TVCG **13**(3), 597–615 (2007)
10. Costa, L., Cesar, R.: Shape Analysis and Classification. CRC Press, Boca Raton (2000)
11. Dubinski, J.: When galaxies collide. Astron. Now **15**(8), 56–58 (2001)
12. Duffin, K.L., Barrett, W.A.: Spiders: a new user interface for rotation and visualization of N-dimensional point sets. In: Proceedings of IEEE Visualization, pp. 205–211 (1994)
13. Emory, M., Iaccarino, G.: Visualizing turbulence anisotropy in the spatial domain with componentality contours. Cent. Turbul. Res. Ann. Res. Briefs 123–138 (2014). https://web.stanford.edu/group/ctr/ResBriefs/2014/14_emory.pdf
14. Ersoy, O., Hurter, C., Paulovich, F., Cantareiro, G., Telea, A.: Skeleton-based edge bundling for graph visualization. IEEE TVCG **17**(2), 2364–2373 (2011)
15. Falcao, A., Feng, C., Kustra, J., Telea, A.: Multiscale 2D medial axes and 3D surface skeletons by the image foresting transform. In: Saha, P., Borgefors, G., di Baja, G.S. (eds.) Skeletonization - Theory, Methods, and Applications. Elsevier (2017). Chap. 2
16. Falcão, A., Stolfi, J., Lotufo, R.: The image foresting transform: theory, algorithms, and applications. IEEE TPAMI **26**(1), 19–29 (2004)
17. Frokjaer, E., Hertzum, M., Hornbaek, K.: Measuring usability: are effectiveness, efficiency, and satisfaction really correlated? In: Proceedings of CHI, pp. 345–352 (2000)
18. Guo, J., Wang, Y., Du, P., Yu, L.: A novel multi-touch approach for 3D object free manipulation. In: Chang, J., Zhang, J.J., Magnenat Thalmann, N., Hu, S.-M., Tong, R., Wang, W. (eds.) AniNex 2017. LNCS, vol. 10582, pp. 159–172. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69487-0_12
19. Hancock, M., Carpendale, S., Cockburn, A.: Shallow-depth 3D interaction: design and evaluation of one-, two- and three-touch techniques. In: Proceedings of ACM CHI, pp. 1147–1156 (2007)
20. Hancock, M., ten Cate, T., Carpendale, S., Isenberg, T.: Supporting sandtray therapy on an interactive tabletop. In: Proceedings of ACM CHI, pp. 2133–2142 (2010)
21. Henriksen, K., Sporring, J., Hornbaek, K.: Virtual trackballs revisited. IEEE TVCG **10**(2), 206–216 (2004)
22. 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)
23. Hinckley, K., Tullio, J., Pausch, R., Proffitt, D., Kassell, N.: Usability analysis of 3D rotation techniques. In: Proceedings of UIST, pp. 1–10 (1997)
24. Hultquist, J.: A virtual trackball. In: Graphics Gems, vol. 1, pp. 462–463 (1990)
25. Dubinski, J., et al.: GRAVITAS: portraits of a universe in motion (2006). https://www.cita.utoronto.ca/~dubinski/galaxydynamics/gravitas.html
26. Jackson, B., Lau, T., Schroeder, D., Toussaint, K., Keefe, D.: A lightweight tangible 3D interface for interactive visualization of thin fiber structures. IEEE TVCG **19**(12), 2802–2809 (2013)
27. Jacob, I., Oliver, J.: Evaluation of techniques for specifying 3D rotations with a 2D input device. In: Proceedings of HCI, pp. 63–76 (1995)
28. Kaye, D., Ivrissimtzis, I.: Mesh alignment using grid based PCA. In: Proceedings of CGTA, pp. 174–181 (2015)
29. Kustra, J., Jalba, A., Telea, A.: Probabilistic view-based curve skeleton computation on the GPU. In: Proceedings of VISAPP. SciTePress (2013)
30. Kustra, J., Jalba, A., Telea, A.: Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. Comput. Graph. Forum **33**(4), 73–87 (2014)
31. Livesu, M., Guggeri, F., Scateni, R.: Reconstructing the curve-skeletons of 3D shapes using the visual hull. IEEE TVCG **18**(11), 1891–1901 (2012)

32. Meijster, A., Roerdink, J., Hesselink, W.: A general algorithm for computing distance transforms in linear time. In: Goutsias, J., Vincent, L., Bloomberg, D.S. (eds.) Mathematical Morphology and its Applications to Image and Signal Processing. Computational Imaging and Vision, vol. 18, pp. 331–340. Springer, Boston (2002). https://doi.org/10.1007/0-306-47025-X_36
33. Ogniewicz, R.L., Kubler, O.: Hierarchic Voronoi skeletons. Pattern Recog. **28**, 343–359 (1995)
34. Partala, T.: Controlling a single 3D object: viewpoint metaphors, speed, and subjective satisfaction. In: Proceedings of INTERACT, pp. 536–543 (1999)
35. 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)
36. Rodrigues, R.S.V., Morgado, J.F.M., Gomes, A.J.P.: Part-based mesh segmentation: a survey. Comput. Graph. Forum **37**(6), 235–274 (2018)
37. Rosenfeld, A., Pfaltz, J.: Distance functions in digital pictures. Pattern Recogn. **1**, 33–61 (1968)
38. Shoemake, K.: ARCBALL: a user interface for specifying three-dimensional orientation using a mouse. In: Proceedings of Graphics Interface (1992)
39. Siddiqi, K., Pizer, S.: Medial Representations: Mathematics, Algorithms and Applications. Springer, Heidelberg (2008). https://doi.org/10.1007/978-1-4020-8658-8
40. Sobiecki, A., Yasan, H.C., Jalba, A.C., Telea, A.C.: Qualitative comparison of contraction-based curve skeletonization methods. In: Hendriks, C.L.L., Borgefors, G., Strand, R. (eds.) ISMM 2013. LNCS, vol. 7883, pp. 425–439. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38294-9_36
41. Tagliasacchi, A., Delame, T., Spagnuolo, M., Amenta, N., Telea, A.: 3D skeletons: a state-of-the-art report. Comput. Graph. Forum **35**(2), 573–597 (2016)
42. Tangelder, J.W.H., Veltkamp, R.C.: A survey of content based 3D shape retrieval methods. Multimed. Tools Appl. **39**(3) (2008)
43. Telea, A.: Feature preserving smoothing of shapes using saliency skeletons. In: Linsen, L., Hagen, H., Hamann, B., Hege, H.C. (eds.) Visualization in Medicine and Life Sciences II. Mathematics and Visualization, pp. 136–148. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21608-4_9
44. Telea, A.: Source code for salience skeleton computation (2014). https://webspace.science.uu.nl/~telea001/Shapes/Salience
45. Telea, A., Jalba, A.: Voxel-based assessment of printability of 3D shapes. In: Soille, P., Pesaresi, M., Ouzounis, G.K. (eds.) ISMM 2011. LNCS, vol. 6671, pp. 393–404. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21569-8_34
46. Telea, A., van Wijk, J.J.: An augmented fast marching method for computing skeletons and centerlines. In: Ebert, D., Brunet, P., Navazo, I. (eds.) Proceedings of Eurographics/IEEE VGTC Symposium on Visualization (VisSym), pp. 251–259. The Eurographics Association (2002)
47. The Authors: Source code and videos of interactive skeleton-based axis rotation (2019). http://www.staff.science.uu.nl/~telea001/Shapes/CUDASkelInteract
48. Yu, L., Isenberg, T.: Exploring one- and two-touch interaction for 3D scientific visualization spaces. In: Posters of Interactive Tabletops and Surfaces, November 2009
49. 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)
50. Zhai, X., Chen, X., Yu, L., Telea, A.: Interactive axis-based 3D rotation specification using image skeletons. In: Proceedings of 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP, pp. 169–178. SciTePress (2020)
51. Zhao, Y.J., Shuralyov, D., Stuerzlinger, W.: Comparison of multiple 3D rotation methods. In: Proceedings of IEEE VECIMS, pp. 19–23 (2011)