Lifting Freehand Concept Sketches into 3D Implementation details

YULIA GRYADITSKAYA, Université Côte d'Azur, Inria, University of Surrey, CVSSP
FELIX HÄHNLEIN, Université Côte d'Azur, Inria
CHENXI LIU, University of British Columbia
ALLA SHEFFER, University of British Columbia
ADRIEN BOUSSEAU, Université Côte d'Azur, Inria

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; **Shape modeling**; *Non-photorealistic rendering*; • **Applied computing** → *Computer-aided design.*

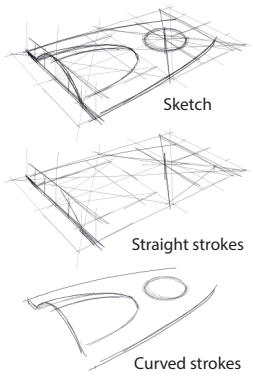
Additional Key Words and Phrases: product design, sketching, line drawing, sketch-based modeling, 3D reconstruction

ACM Reference Format:
Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D Implementation details . *ACM Trans. Graph.* 39, 6, Article 1 (December 2020), 3 pages. https://doi.org/10.1145/3414685.3417851

1 2D ANALYSIS

1.1 Straight line detection.

Our method reconstructs straight scaffold lines as a preliminary step for reconstructing surface curves. We distinguish straight from curved strokes by performing a PCA analysis on the point samples that compose each stroke, and consider that a stroke is curved if the ratio between the second and first eigen values is above a threshold, set to 0.001 in our implementation. While some of the straight strokes correspond to surface features rather than to construction lines, this distinction is not relevant for our algorithm, which reconstructs polyhedral shapes no matter their semantic meaning. Finally, we classify curved strokes as ellipses if they form closed regions.



1.2 Vanishing point detection

Our method targets design drawings of man-made objects, for which scaffolds are dominated by three sets of orthogonal lines aligned with the coordinate axes. We identify these three sets by running the vanishing point detection algorithm of Rother [2002], which considers multiple candidate vanishing points as triplets of points to which most lines converge. We use the variant of Hedau et al. [2009] to compute the probability that a stroke converges towards a vanishing point.

For each triplet of candidate vanishing points the principal point, and focal length are estimated. The triplet that satisfies constraints on orthogonality, principal point, and focal length is then selected. We use the constraints suggested by Rother [2002], except that we converted the constraint on focal length f to a constraint on field-of-view. We impose that the field-of-view lies in the interval $[10°, 120°]$.

1.3 Camera calibration

In addition to providing a strong prior on mutual orthogonality between strokes, the detected vanishing points allow us to calibrate a perspective camera [Guillou et al. 2000; Orghidan et al. 2012], which gives us the 3D direction of the axes associated to these vanishing points. Estimating a perspective projection matrix requires first to find the principal point.

Principal point computation. Since the principal point computation is a key element of computing a camera projection matrix, we summarize below three possible configurations of vanishing points [Rother 2002], which affect the computation of a principal point. If there are three finite vanishing points, the principal point is the orthocenter of the triangle formed by these points [Cipolla et al. 1999]. In case of two finite vanishing points, the principal point lies on the line connecting two finite vanishing points. Since in sketches the principal point can be located arbitrarily with respect to the image area [Gryaditskaya et al. 2019], we select the principal point that lies on the line connecting two finite vanishing points and which minimizes the distance to a third vanishing point. We found this more robust for sketches than minimizing the distance to the image plane center. In case of one finite vanishing point, the projection is near orthogonal and the principal point coincides with the finite vanishing point. In this case, it is impossible to estimate the focal length value and an orthogonal projection should be used instead. The focal length is estimated from a given triplet of vanishing points and a principal point [Guillou et al. 2000].

Projection matrix. Principal point, focal length and a triplet of vanishing points define the projection matrix uniquely up to a scale and a translation. For our application, scale and scene origin can be chosen arbitrarily. We assign the *up* vector of the 3D object

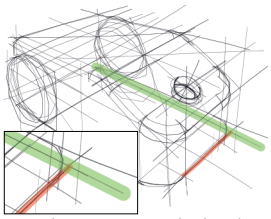
Authors' addresses: Yulia Gryaditskaya, Université Côte d'Azur, Inria, University of Surrey, CVSSP, yulia.gryaditskaya@gmail.com; Felix Hähnlein, Université Côte d'Azur, Inria; Chenxi Liu, University of British Columbia; Alla Sheffer, University of British Columbia; Adrien Bousseau, Université Côte d'Azur, Inria.

0730-0301/2020/12-ART1 $15.00
https://doi.org/10.1145/3414685.3417851

coordinate system to the vanishing point with the highest absolute vertical coordinate. The camera translation vector is then computed as done by Orghidan et al. [2012].

1.4 Intersection detection and grouping

We pre-compute all intersections between strokes. Because our input drawings are drawn freehand, strokes sometimes stop before their intended end, leaving small gaps between lines that should intersect. We observed that the degree of inaccuracy often varies over the drawing, as fine details are drawn with more care than rough structures.

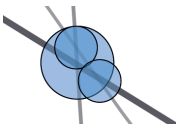

Strokes accuracy neigborhoods

We account for such variation by expressing the neighborhood size of a stroke r_{stroke} as a function of the speed v_{stroke} at which the stroke has been drawn:

$$r_{stroke} = \tau_{\min} + \frac{(\tau_{\max} - \tau_{\min})}{(v_{\max} - v_{\min})} v_{stroke}, \quad (1)$$

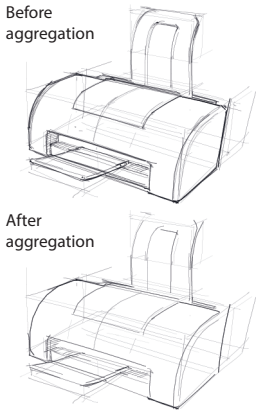
where $\tau_{\min} = 2\bar{w}_{strokes}, \tau_{\max} = 6.5\bar{w}_{strokes}$ are two thresholds, whose values are derived based on the maximum width of the strokes $\bar{w}_{strokes}$ in the current sketch, and $v_{\min}, v_{\max}$ are the minimum and the maximum speed of the strokes over all the sketches in the OpenSketch dataset [Gryaditskaya et al. 2019]. We detect additional intersections by extending each straight stroke by the size of its neighborhood.

Another characteristic of freehand design drawings is that multiple coincident lines often do not intersect exactly at the same point, but rather result in a multitude of nearby intersections. We deal with this uncertainty by grouping nearby intersections, such that any lines incident with the group are considered to intersect. However, we found hard clustering algorithms difficult to apply to this problem, because groups of intersections often overlap in noisy drawings. Instead, we adopt a soft clustering strategy where for each intersection, we pair it with all other intersections along the two intersecting strokes if the intersections in the pair are encompassed by *neighborhood* of any of the intersections in this pair. We select the *neighborhood* size of an intersection between two strokes as the neighborhood size of the stroke that was drawn last. The motivation is that the accuracy of the intersection is defined by the accuracy of the last drawn stroke: let's say if the newly added detail stroke intersects some rough scaffold, the accuracy of the intersection should match the accuracy of the detail stroke. An intersection is then considered to belong to the group of another one if any of the two falls within the neighborhood of the other.


Intersections pair grouping

1.5 Stroke aggregation

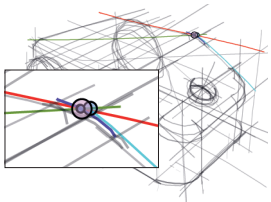
Designers often employ many overdrawn strokes to achieve the line or curve they envision, which greatly increases the complexity of our sketches. We reduce this complexity by aggregating nearby strokes based on their spatial and angular proximity. For straight strokes, we consider that two strokes are in the same cluster if any of the two falls within the neighborhood of the other one for more than 75% of its length, and if they form an angle smaller than 5°.



In addition, we also require that these strokes are drawn in sequence, *i.e.*, less than five strokes apart. We then fit a single line segment per cluster, and keep the time stamp of the earliest stroke as the time stamp of the cluster. These clustering parameters are intentionally conservative as we prefer to keep a few spurious overdrawn strokes rather than to merge strokes that should not be merged, for instance because they might represent different parts of the scaffold. For curved strokes, we perform a more aggressive clustering using the first steps of *StrokeAggregator* [Liu et al. 2018] (conservative setting of their public binary).

1.6 Intersection filtering

The scaffolds we are interested in, represent 3D polyhedra, whose corners have three or more coincident lines. In contrast, accidental occlusions most often occur between two isolated lines. We deduced from this observation a simple heuristic to filter out many of the accidental occlusions between scaffold lines.

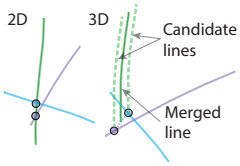


For each group of intersections, we measure the orientation of all incident straight strokes and mark the intersections with three or more different orientations as likely intersections, resulting in a set of intersections $\mathbb{I}^*$: We consider that two orientations differ if they form an angle of more than 5° for straight strokes and 35° for curved strokes. We then for each stroke s_j check that there is at least one intersection i, marked as likely ($i \subset \mathbb{I}^*$), within 25% of the interval between the two most extreme intersections at this stroke from the respective extreme endpoints. If there are no such intersections we mark all the intersections, within the considered intervals, with the strokes that have non-coinciding directions as likely $\mathbb{I}^*_{s_j}$. We process each stroke independently, so that the newly added sets of intersections do not influence the selection of additional intersections for the remaining strokes. Thus the total set of intersections which we keep is $\mathbb{I}^* \cup \left(\bigcup_{i=1}^{N} \mathbb{I}^*_{s_j} \right)$, where N is the total number of strokes in the sketch. This allows us to reduce the problem dimensionality from several thousands to a few hundred unknowns, on average.

2 CANDIDATES LINES FOR STRAIGHT STROKES

Let us first consider the general case where we have successfully reconstructed all strokes until the current one. The new stroke produces N intersections with the reconstructed strokes, from which we generate the $\frac{N(N-1)}{2}$ candidate 3D lines passing through all possible pairs of intersections. In the case of an axis-aligned stroke,

we also create N additional candidates, each going through one of the intersections and parallel to the associated coordinate axis.



In practice, the above procedure produces many similar candidates when multiple pairs of intersections align in 3D. We merge these redundant lines by averaging their end points. We detect candidates lines that are nearly identical by testing if they run near the same intersections. We consider that a 3D intersection is near a line if its distance to the line is less than 10% of the length of that line. We measure this distance both in 3D space and along the viewing ray going through the intersection.

In the presence of an ambiguous stroke, we generate candidate lines by considering all consistent combinations of versions of preceding strokes on which it depends. If we count the total number of candidate lines, formed by each pair of intersections, then we need to construct the pairs formed by the intersections with assigned strokes, the pairs formed by the intersections with assigned strokes and each version of each stroke with multiple candidates, and, finally, all the pairs between each version of each stroke with multiple candidates, resulting in the following total number of candidate lines:

$$N_{cl} = \frac{K(K-1)}{2} + \sum_{j=1}^{L} KM_j + \sum_{j=1}^{L} M_j \left(\sum_{i=1, i\neq j}^{L} M_i \right), \tag{2}$$

where K is the number of intersecting assigned strokes, L is the number of intersecting strokes with multiple candidates, M_j is the number of candidates for the $j^{th} \in [1, L]$ stroke.

REFERENCES

Roberto Cipolla, Tom Drummond, and Duncan P Robertson. 1999. Camera Calibration from Vanishing Points in Image of Architectural Scenes.. In *BMVC*.

Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Fredo Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* (2019).

Erwan Guillou, Daniel Meneveaux, Eric Maisel, and Kadi Bouatouch. 2000. Using vanishing points for camera calibration and coarse 3D reconstruction from a single image. *The Visual Computer* 16, 7 (2000).

Varsha Hedau, Derek Hoiem, and David Forsyth. 2009. Recovering the spatial layout of cluttered rooms. In *Proc. ICCV*. IEEE.

Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: consolidating raw sketches into artist-intended curve drawings. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 97.

Radu Orghidan, Joaquim Salvi, Mihaela Gordan, and Bogdan Orza. 2012. Camera calibration using two or three vanishing points. In *FedCSIS*.

Carsten Rother. 2002. A new approach to vanishing point detection in architectural environments. *Image and Vision Computing* 20, 9-10 (2002).