Universiteit Hasselt

Masterproef voorgedragen tot het behalen van de graad van master in de informatica

Region-Based Selective Remeshing for User-Guided Generative 3D Model Editing

Auteur:

Vervaecke Xander

promotor:

Prof. Dr. Nick Michiels

Begeleider:

Joren Michels

Academiejaar 2025-2026

Abstract

This thesis grew out of a personal frustration. While trying to build a custom 3D model for a side project, I hit the wall every non-expert hits: tools like Blender take months to learn, and I never got comfortable with them, while AI generators such as Meshy returned a model in seconds but offered no real way to change it. When the result was close but not quite right, the only option was to start over, which is not how I wanted to work.

This thesis explores whether generative AI can instead be used to edit an existing 3D model in controlled steps, without losing the parts of the model the user wanted to keep. The proposed system asks the user to sketch on a chosen view of the model and to describe the intended change in a short prompt. From this input, a few candidate images are generated so the user can pick the one that best matches their idea. That image is then turned into a 3D version of the edit, which the system merges into the original mesh. The merging step, called ASMR-3D, is the technical heart of the thesis: it figures out which region was actually changed, rebuilds only that region with selective remeshing, and keeps every other vertex exactly where it was, while blending the two regions.

The system was tested in a user study with fourteen participants, none of whom had ever used a 3D editing tool before. The merged meshes preserved the original geometry roughly twenty-seven times more faithfully than a full regeneration, and the User Experience Questionnaire placed the system in the *Excellent* band on five of its six scales. All participants said they would use it again. The main weaknesses that remain are the retexturing step and the interface for adjusting the edited region.

Acknowledgments

This master's thesis would not have been possible without the support of a number of people, whom I would like to thank here.

My deepest thanks go to my promoter, Prof. dr. Nick Michiels, and my counselor, Joren Michels. Their guidance, valuable insights, and feedback have been of immense value throughout the entire project. I would also like to thank them explicitly for being so approachable: I could always come to them with my questions and would receive a quick response every time. Above all, I am very grateful for the freedom they gave me to shape this topic in my own way and to develop my own implementation of it. That space allowed me to feel a real sense of ownership over this work, which kept my motivation high throughout the year.

I would also like to thank Prof. Dr. Gustavo Rovelo Ruiz for his help with setting up the user study. His experience with this type of evaluation helped me build a thoughtful and meaningful study.

Finally, I would like to thank all participants of the user study. They took the time to go through the entire system and to give honest, detailed feedback. Without their input, the evaluation of this master's thesis would not have been possible.

To everyone who contributed to this work in one way or another: thank you very much.

Summary

Motivation

The starting point of this thesis was a personal frustration. While working on a side project, I wanted to create a custom 3D model and quickly ran into the same wall that every non-expert hits. Professional tools such as Blender or Maya require months of training and a serious artistic background. I tried to learn Blender myself and never got comfortable with it: the interface, the modeling concepts, and the sheer number of operations involved made even simple objects feel out of reach. To avoid that learning curve, I turned to modern AI tools such as Meshy, which can generate a 3D model from a short text prompt or a single image. These tools were much easier to use, but they had a different problem: I felt I had almost no control over what came out. When the result was close to what I wanted but not quite right, there was no real way to tweak it. The only option was to change the prompt and hope the next regeneration would be better. This is the opposite of how designers actually work, which is in small, iterative steps where each version builds on the previous one.

This thesis attacks that gap from a non-expert point of view. The central question is whether generative AI can be used to edit an existing 3D model in controlled steps, while keeping every part of the model that the user did not want to change exactly as it was. The work splits this question into two concrete challenges. The first is giving the user more direct control over what the edit should look like, beyond what a single text prompt can express. The second is preserving the original geometry of the model, so that the parts of the model that should not change are not affected by the edit.

Proposed pipeline

The proposed system is built as a pipeline that the user goes through once per edit. The pipeline can be used repeatedly to apply several edits in a row to the same model. It is divided into two parts.

The first part of the pipeline gathers user input and produces a new 3D model that reflects the requested edit. It starts with the user uploading an existing 3D model, rotating it to choose a viewpoint, and drawing a sketch in magenta on top of that view. Three brushes are available: a draw brush to mark what should be added, a hide brush to mark what should be removed, and an erase brush to erase the sketch. Once the sketch is done, the user writes a short text prompt that describes the intended change. A language model (Gemini) checks the prompt, asks one clarifying question if needed, and rewrites it into a clearer, more precise version. The user can also attach an optional reference image to communicate visual style or shape.

These inputs are then sent to an image-editing model that produces four edited candidate images in parallel. This intermediate visual feedback is important: it lets the user see roughly what the final result will look like before any 3D processing happens. The user picks the candidate that fits best and can optionally refine it with another sketch and prompt. The chosen image is then expanded into a four-view multiview (front, left, right, and back) that preserves the

look of the edit from every angle. Finally, this multiview is passed to the Meshy service, which produces a new 3D model that reflects the edit. At this point, the user could already stop and keep this generated model, but its geometry does not match the original mesh exactly: it has different vertex positions, slightly different proportions, and a different topology. The second part of the pipeline fixes this.

ASMR-3D: Alignment, Selection, Mesh Reconstruction

The second part of the pipeline, called ASMR-3D, is the core technical contribution of this thesis. It takes the original mesh and the generated mesh and merges them into a single output, keeping the original geometry intact everywhere except in the region the user wanted to change. ASMR-3D consists of three main steps.

The first step is alignment. The original mesh and the generated mesh use different coordinate systems, so they have to be brought into the same frame before they can be merged. Two alignment methods are exposed to the user. The first method works directly on the 3D geometry: it samples both meshes as point clouds, computes local descriptors, finds a rough match using RANSAC, refines this with a closed-form similarity transform, and finishes with point-to-plane ICP. The second method works on the multiview images instead: it extracts silhouettes from both renders, calibrates a pixels-per-world ruler from the original mesh, and uses bounding-box differences to estimate the rotation and translation that bring the edit into place. The two methods cover different failure modes, so the user is shown both results and picks the one that looks best.

The second step decides which part of the mesh should be edited. Rather than asking the user to paint the region by hand, an automatic edit mask is computed through a voting system that combines three independent cues. The first cue is geometric: vertices on the edited mesh that lie far from the original surface are marked as different. The second cue is image-based: the system compares the original render with the chosen edited image and projects the difference, together with the user's sketch, back onto the mesh through the calibrated camera. The third cue is multiview: the same comparison is repeated for each of the four multiview panels, and each panel casts a vote for the vertices it considers changed. The geometric cue is worth one vote, the image-difference cue one vote, and the multiview cue two votes (if a vertex is hit in at least two of the four views). Vertices with at least three votes are kept. Two post-processing steps are then used to expand the selection along the mesh surface and remove small, unrelated clusters. The remaining vertices are grouped into a small set of editable spheres using k-means and Ritter's bounding sphere algorithm. The user can move, resize, add, or remove these spheres in case the automatic mask was not perfect. A separate hide mask, drawn on a single view, can be used to remove parts of the original mesh.

The third step actually merges the two meshes. Cutting both meshes along the boundary and welding them together does not work because the two boundaries have different shapes and vertex densities. Instead, the system builds a hybrid initial mesh: vertices inside the edit region are taken from the generated mesh, and vertices outside are taken from the original. A custom hole-filling algorithm bridges the small gap between the two regions with a zigzag stitch that connects vertices on opposite sides. This initial mesh is then improved using selective continuous remeshing, inspired by the work of Palfinger [Palfinger, 2022]. The system renders a Poisson-blended reference mesh (inspired by CraftMesh [Jincheng et al., 2026]) from several camera angles, and gradually deforms the working mesh so that its rendered normals and silhouettes match those of the reference. Two mechanisms ensure that the original vertices stay exactly where they were: an anchor loss with a very high weight, and explicit gradient masking that zeroes out the gradient on every vertex outside an expanded edit mask. A thin transition zone is allowed to relax slightly around the boundary, which is what makes the seam practically invisible. A user-controlled smoothing factor per region and a remeshing schedule that matches the target edge length to the surrounding mesh density complete the step.

After the geometry is merged, the model is retextured. In the current implementation, the front and back views of the edited multiview are sent back to Meshy together with the merged mesh, which returns a new textured version. This step is the only place where the system does not yet fully respect the geometry-preservation rule: the texture of the entire model is regenerated rather than only the edited region.

Evaluation

The system was evaluated through a within-subjects user study with fourteen participants. Each participant performed two editing tasks. The first task was guided: every participant was asked to replace the head of the same character mesh with a goblin head, using a reference image. The second task was free: participants chose any 3D model they liked from the Meshy catalog and proposed any edit they wanted. Two pilot sessions were used to refine the protocol before the real sessions started.

The evaluation collected three kinds of data. Quantitative metrics were computed automatically for each stage of the pipeline, covering image fidelity, image diversity, generated-mesh similarity, alignment quality, edit-mask precision and recall, topological cleanliness of the merged mesh, and preservation of the original vertices. Subjective ratings were collected through a per-stage Likert questionnaire, the standard User Experience Questionnaire (UEQ), and free-text remarks.

Results

The results support the goals set out at the start of the thesis. On the input side, every participant accepted the prompt rewritten by the language model over the one they had typed themselves, and Likert ratings for the sketching and prompting interface were uniformly positive, with a clear learning effect between the first and the second task. The image-editing model produced candidates that preserve the original render outside the sketched region with very small pixel differences.

On the geometric side, both alignment methods produced good results, with the ICP-based method chosen in eleven out of fourteen free trials and the image-based method preferred in the remaining three, confirming that both methods are useful. The automatic edit mask consistently landed on the right part of the model: when participants did adjust it, they almost always extended the selection a little rather than removing what the algorithm had picked, which suggests the mask is a reliable starting point that the user can fine-tune rather than rebuild. The merged mesh preserved the original vertices outside the edit region with an anchor surface error of about 0.0017 on a unit bounding box, which is roughly twenty-seven times more accurate than what a full regeneration with Meshy produces. The merged mesh also matched the average edge length of the original input, which is what makes the transition practically visually invisible.

On the user side, the User Experience Questionnaire reached the *Excellent* band on five of the six scales and the *Good* band on the sixth (Perspicuity, just below the Excellent threshold). All fourteen participants said they would use the system again and recommend it, with ratings of four or five out of five. Free-text remarks were dominated by positive comments about creative freedom and ease of use, with one recurring complaint about the sphere-based interface for editing the automatic mask.

Limitations and future work

The current system is best understood as a high-level prototype. The most important limitation is the retexturing step, which regenerates the entire texture rather than only the edited region. A second limitation is that the hole-filling step occasionally introduces backward-facing

triangles, which appear as non-manifold edges in some merged meshes. A third limitation is that the sphere-based interface for editing the automatic mask was the part of the system that participants liked least, even though the modifications they actually made were small. The pipeline also depends on external generative services (Gemini and Meshy) and inherits their limitations.

The most promising directions for future work follow from these limitations. The retexturing step could be redesigned to blend a freshly generated texture into the original one only along the edit boundary, using Poisson blending in texture space. The hole-filling step could be replaced by starting the optimization from a watertight initial mesh. The edit-mask interface could be replaced by a more direct interaction, such as painting the region on a 2D view or on the model itself. Higher-fidelity multiview editors and inverse-rendering-based 3D generators would also make every downstream stage more reliable.

Conclusion

This thesis shows that generative AI can be used to edit existing 3D models in a controlled, step-by-step way without losing the original geometry. The proposed pipeline combines multimodal input with a region-based selective remeshing method that explicitly preserves the vertices the user did not want to change. The user study supports the central claims: the system is usable by people with no 3D-modeling background, it preserves the original geometry to a degree that a lot of comparable tools can not reach, and it received uniformly positive ratings from the participants who used it. Although several stages still have clear room for improvement, the work demonstrates that fine-grained user control and exact preservation of the original model can be treated as first-class requirements rather than as afterthoughts in generative 3D editing.

Samenvatting

Motivatie

Het startpunt van deze thesis was een persoonlijke frustratie. Tijdens het werken aan een project wilde ik een eigen 3D-model maken en liep ik al snel tegen dezelfde muur als elke niet-expert. Professionele tools zoals Blender of Maya vergen vaak maanden aan training en een serieuze artistieke achtergrond. Ik heb zelf geprobeerd om Blender te leren en raakte er nooit echt vertrouwd mee: de interface, de modelleerconcepten en het enorme aantal bewerkingen maakten dat zelfs eenvoudige objecten onbereikbaar aanvoelden. Om die leercurve te vermijden, schakelde ik over op moderne AI-tools zoals Meshy, die een 3D-model kunnen genereren op basis van een korte tekstprompt of één enkele afbeelding. Deze tools waren veel makkelijker in gebruik, maar hadden een ander probleem: ik had het gevoel dat ik bijna geen controle had over wat eruit kwam. Wanneer het resultaat dicht in de buurt kwam van wat ik wilde, maar net niet helemaal juist was, was er geen echte manier om het aan te passen. De enige optie was om de prompt aan te passen en te hopen dat de volgende generatie beter zou zijn. Dit is het tegenovergestelde van hoe ontwerpers in de praktijk werken, namelijk in kleine, iteratieve stappen waarbij elke versie voortbouwt op de vorige.

Deze thesis pakt dit probleem aan vanuit het standpunt van een niet-expert. De centrale vraag is of generatieve AI gebruikt kan worden om een bestaand 3D-model in gecontroleerde stappen te bewerken, terwijl elk deel van het model dat de gebruiker niet wou veranderen exact zo blijft. Het werk splitst deze vraag op in twee concrete uitdagingen. De eerste is om de gebruiker meer directe controle te geven over hoe de bewerking eruit moet zien, verder dan wat een enkele tekstprompt kan uitdrukken. De tweede is om de oorspronkelijke geometrie van het model te behouden, zodat de delen van het model die niet mogen veranderen ook niet beïnvloed worden door de bewerking.

Voorgestelde pipeline

Het voorgestelde systeem is opgebouwd als een pipeline die de gebruiker per bewerking één keer doorloopt. De pipeline kan herhaaldelijk gebruikt worden om verschillende bewerkingen na elkaar toe te passen op hetzelfde model. Ze is opgedeeld in twee delen.

Het eerste deel van de pipeline verzamelt de input van de gebruiker en produceert een nieuw 3D-model dat de gevraagde bewerking weergeeft. Het start met de gebruiker die een bestaand 3D-model uploadt; het roteert om een "view" te kiezen en een schets in magenta tekent bovenop dat aanzicht. Er zijn drie penselen beschikbaar: een tekenpenseel om aan te geven wat toegevoegd moet worden, een verbergpenseel om aan te geven wat verwijderd moet worden, en een gumpenseel om enkel de schetsen te verbergen. Zodra de schets klaar is, schrijft de gebruiker een korte tekstprompt die de verandering beschrijft. Een taalmodel (Gemini) controleert de prompt, stelt indien nodig één verduidelijkende vraag en herschrijft hem tot een duidelijkere, preciezere versie. De gebruiker kan ook optioneel een referentiebeeld toevoegen om meer context te geven.

Deze inputs worden vervolgens naar een generatief AI-model gestuurd dat vier bewerkte kandidaatafbeeldingen produceert. Deze tussentijdse visuele feedback is belangrijk: ze laat de gebruiker grofweg zien hoe het eindresultaat eruit zal zien voor er enige 3D-verwerking gebeurt. De gebruiker kiest de kandidaat die het best past en kan deze optioneel verder verfijnen met een nieuwe schets en prompt. Het gekozen beeld wordt vervolgens uitgebreid naar een multiview met vier aanzichten (voor, links, rechts en achter), die de uitstraling van de bewerking vanuit elke hoek behoudt. Tot slot wordt deze multiview doorgegeven aan de Meshy-service, die een nieuw 3D-model produceert dat de bewerking weergeeft. Op dit punt zou de gebruiker al kunnen stoppen en dit gegenereerde model behouden, maar de geometrie ervan komt niet exact overeen met de originele mesh: ze heeft andere vertexposities, andere proporties en een andere topologie. Het tweede deel van de pipeline lost dit op.

ASMR-3D: Alignment, Selection, Mesh Reconstruction

Het tweede deel van de pipeline, genaamd ASMR-3D, is de centrale technische bijdrage van deze thesis. Het neemt de originele mesh en de gegenereerde mesh en verbindt ze tot één enkele output, waarbij de originele geometrie intact blijft, behalve in de regio die de gebruiker wou veranderen. ASMR-3D bestaat uit drie hoofdstappen.

De eerste stap is alignment. De originele mesh en de gegenereerde mesh gebruiken verschillende coördinatenstelsels, dus moeten ze in hetzelfde referentiekader gebracht worden voor ze kunnen worden samengevoegd. Twee alignmentmethoden worden aan de gebruiker aangeboden. De eerste methode werkt rechtstreeks op de 3D-geometrie: ze samplet beide meshes als puntenwolken, berekent lokale descriptoren, vindt een ruwe overeenkomst met behulp van RANSAC en eindigt met point-to-plane ICP. De tweede methode werkt in plaats daarvan op de multiview-beelden: ze extraheert silhouetten uit beide renders, kalibreert een pixels-per-wereld-liniaal vanuit de originele mesh en gebruikt verschillen in bounding boxes om de rotatie en translatie te schatten die de bewerking op zijn plaats brengt. De twee methoden dekken verschillende gebreken af, dus krijgt de gebruiker beide resultaten te zien en kiest hij degene die er het best uitziet.

De tweede stap beslist welk deel van de mesh bewerkt moet worden. In plaats van de gebruiker te vragen om de regio met de hand in te tekenen, wordt een automatisch editmask berekend via een stemsysteem dat drie onafhankelijke aanwijzingen combineert. De eerste aanwijzing is geometrisch: vertices op de bewerkte mesh die ver van het originele oppervlak liggen, worden als verschillend gemarkeerd. De tweede aanwijzing is beeldgebaseerd: het systeem vergelijkt de originele render met het gekozen bewerkte beeld en projecteert het verschil, samen met de schets van de gebruiker, terug op de mesh via de gekalibreerde camera. De derde aanwijzing is multiview: dezelfde vergelijking wordt herhaald voor elk van de vier multiview-panelen, en elk paneel brengt een stem uit voor de vertices die het als veranderd beschouwt. De geometrische aanwijzing is één stem waard, de beeldverschil-aanwijzing één stem en de multiview-aanwijzing twee stemmen (als een vertex in minstens twee van de vier aanzichten geraakt wordt). Vertices met minstens drie stemmen worden behouden. Twee nabewerkingsstappen worden vervolgens gebruikt om de selectie uit te breiden langs het mesh-oppervlak en kleine, niet-gerelateerde clusters te verwijderen. De overblijvende vertices worden gegroepeerd in een kleine verzameling bewerkbare bollen via k-means en Ritter's bounding sphere-algoritme. De gebruiker kan deze bollen verplaatsen, vergroten, toevoegen of verwijderen voor het geval dat het automatische masker niet perfect was. Een aparte hide mask, getekend op één enkel aanzicht, kan gebruikt worden om delen van de originele mesh te verwijderen.

De derde stap voegt de twee meshes effectief samen. De twee meshes langs de grens doorsnijden en aan elkaar lassen werkt niet, omdat de twee grenzen verschillende vormen en vertexdichtheden hebben. In plaats daarvan bouwt het systeem een hybride initiële mesh: vertices binnen de bewerkingsregio worden uit de gegenereerde mesh genomen, en vertices erbuiten uit de originele. Een eigen ontwikkeld algoritme voor gatvulling overbrugt de kleine kloof tussen de twee regio's met een zigzagstiksel dat vertices aan tegenovergestelde zijden verbindt. Deze initiële

mesh wordt vervolgens verbeterd met behulp van selectieve continuous remeshing, geïnspireerd op het werk van Palfinger [Palfinger, 2022]. Het systeem rendert een Poisson-geblendde referentiemesh (geïnspireerd op CraftMesh [Jincheng et al., 2026]) vanuit verschillende camerahoeken en vervormt de werkende mesh geleidelijk zodat haar gerenderde normalen en silhouetten overeenkomen met die van de referentie. Twee mechanismen zorgen ervoor dat de originele vertices precies blijven waar ze waren: een anchor loss met een zeer hoog gewicht en expliciete gradient masking die de gradiënt op elke vertex buiten een uitgebreid edit mask op nul zet. Een dunne transitiezone mag licht ontspannen rond de grens, en dat is wat de naad praktisch onzichtbaar maakt. Een door de gebruiker instelbare smoothing-factor per regio en een remeshing-schedule dat de doelranglengte afstemt op de omliggende mesh-dichtheid maken de stap compleet.

Nadat de geometrie samengevoegd is, wordt het model van een nieuwe tekstuur voorzien. In de huidige implementatie worden de voor- en achteraanzichten van de bewerkte multiview samen met de samengevoegde mesh teruggestuurd naar Meshy, dat een nieuwe getextureerde versie teruggeeft. Deze stap is de enige plaats waar het systeem de regel van geometriebehoud nog niet volledig respecteert: de textuur van het volledige model wordt opnieuw gegenereerd in plaats van enkel die van de bewerkte regio.

Evaluatie

Het systeem werd geëvalueerd via een within-subjects-gebruikersstudie met veertien deelnemers. Elke deelnemer voerde twee bewerkingstaken uit. De eerste taak was gestuurd: elke deelnemer kreeg de opdracht om het hoofd van dezelfde personage-mesh te vervangen door een goblinhoofd, met behulp van een referentieafbeelding. De tweede taak was vrij: deelnemers kozen om het even welk 3D-model dat ze leuk vonden uit de Meshy-catalogus en stelden om het even welke bewerking voor die ze wilden. Twee pilootsessies werden gebruikt om het protocol te verfijnen voordat de echte sessies begonnen.

De evaluatie verzamelde drie soorten data. Kwantitatieve metrieken werden automatisch berekend voor elke fase van de pipeline en dekten beeldfideliteit, beelddiversiteit, gelijkenis van de gegenereerde mesh, alignmentkwaliteit, precisie en recall van het edit mask, topologische zuiverheid van de samengevoegde mesh en het behoud van de originele vertices. Subjectieve beoordelingen werden verzameld via een Likert-vragenlijst per fase, de standaard User Experience Questionnaire (UEQ) en vrije tekstvelden.

Resultaten

De resultaten ondersteunen de doelen die aan het begin van de thesis werden vooropgesteld. Aan de invoerkant accepteerde elke deelnemer de door het taalmodel herschreven prompt boven de prompt die ze zelf hadden getypt, en de Likert-scores voor de schets- en prompt-interface waren over de hele lijn positief, met een duidelijk leereffect tussen de eerste en de tweede taak. Het beeldbewerkingsmodel produceerde kandidaten die de originele render buiten de geschetste regio behouden met zeer kleine pixelverschillen.

Aan de geometrische kant leverden beide alignmentmethoden goede resultaten op, waarbij de op ICP gebaseerde methode in elf van de veertien vrije trials werd gekozen en de op beelden gebaseerde methode in de overige drie de voorkeur kreeg, wat bevestigt dat beide methoden nuttig zijn. Het automatische editmask landde consistent op het juiste deel van het model: wanneer deelnemers het wel aanpasten, breidden ze de selectie bijna altijd lichtjes uit in plaats van te verwijderen wat het algoritme had gekozen, wat suggereert dat het masker een betrouwbaar startpunt is dat de gebruiker kan bijschaven in plaats van opnieuw op te bouwen. De samengevoegde mesh behield de originele vertices buiten de bewerkingsregio met een anchor surface error van ongeveer 0.0017 op een unit bounding box, wat ruwweg zevenentwintig keer nauwkeuriger is dan wat een volledige regeneratie met Meshy oplevert. De samengevoegde

mesh had ook ongeveer dezelfde gemiddelde randlengte als de originele invoer, en dat is wat de overgang visueel praktisch onzichtbaar maakt.

Aan de gebruikerskant bereikte de User Experience Questionnaire de *Excellent*-band op vijf van de zes schalen en de *Good*-band op de zesde (Perspicuity, net onder de Excellent-drempel). Alle veertien deelnemers gaven aan dat ze het systeem opnieuw zouden gebruiken en aanbevelen, met beoordelingen van vier of vijf op vijf. De vrije opmerkingen werden gedomineerd door positieve commentaren over creatieve vrijheid en gebruiksgemak, met één terugkerende klacht over de op bollen gebaseerde interface om het automatische masker te bewerken.

Beperkingen en toekomstig werk

Het huidige systeem kan het best begrepen worden als een hoogniveauprototype. De belangrijkste beperking is de hertexturingstap, die de volledige textuur opnieuw genereert in plaats van enkel de bewerkte regio. Een tweede beperking is dat de gatvullingsstap af en toe achterwaartsgerichte driehoeken introduceert, die in sommige samengevoegde meshes verschijnen als non-manifold edges. Een derde beperking is dat de op bollen gebaseerde interface voor het bewerken van het automatische masker het deel van het systeem was dat deelnemers het minst leuk vonden, hoewel de aanpassingen die ze daadwerkelijk maakten klein waren. De pipeline is ook afhankelijk van externe generatieve diensten (Gemini en Meshy) en erft hun beperkingen.

De meest beloftevolle richtingen voor toekomstig werk volgen uit deze beperkingen. De hertexturingstap zou herontworpen kunnen worden om een nieuw gegenereerde textuur enkel langs de bewerkingsgrens in de originele textuur te blenden, met behulp van Poisson-blending in de textuurruimte. De gatvullingsstap zou vervangen kunnen worden door de optimalisatie te starten vanaf een watertight initiële mesh. De edit mask interface zou vervangen kunnen worden door een meer directe interactie, bijvoorbeeld door de regio rechtstreeks op een 2D-aanzicht of op het model zelf te schilderen. Multiview-editors met hogere fideliteit en op inverse rendering gebaseerde 3D-generatoren zouden ook elke daaropvolgende fase betrouwbaarder maken.

Conclusie

Deze thesis toont aan dat generatieve AI gebruikt kan worden om bestaande 3D-modellen op een gecontroleerde, stapsgewijze manier te bewerken zonder de originele geometrie te verliezen. De voorgestelde pipeline combineert multimodale invoer met een region-based selectieve remeshingmethode die expliciet de vertices behoudt die de gebruiker niet wou veranderen. De gebruikersstudie ondersteunt de centrale claims: het systeem is bruikbaar voor mensen zonder 3D-modelleerachtergrond, het behoudt de originele geometrie in een mate die veel vergelijkbare tools niet kunnen bereiken en het kreeg over de hele lijn positieve beoordelingen van de deelnemers die het gebruikten. Hoewel verschillende fasen nog duidelijk ruimte voor verbetering hebben, demonstreert het werk dat meer precieze gebruikerscontrole en exact behoud van het originele model behandeld kunnen worden als eerste-klasvereisten in plaats van als extras in generatieve 3D-bewerking.

Contents

Chapter 1

Introduction

When working on a personal project, I (the author of this thesis) wanted to create a custom 3D model, but found tools like Blender far too complex to learn. Instead, I turned to an automated 3D model generator named Meshy, only to find that while it was easier to use, it offered little control over the result. Whenever the output was not quite right, there was no way to tweak it: the entire model had to be regenerated from scratch. This frustration sparked the central question of this thesis: Can we do better?

3D models play a vital role in a wide range of applications, including gaming, robotics simulation, virtual and augmented reality, film, and interactive entertainment [Jiang, 2024]. Despite this ever-growing demand, creating custom 3D assets for specific objects remains remarkably difficult. Manual modeling has been described as **notoriously difficult** and inherently requires expertise [Cheng et al., 2023]. Tools such as *Blender* or *Maya* can assist designers, but using them effectively still demands familiarity with complex professional software, considerable artistic skill, and substantial time investment, which makes the process hardly accessible to layman users [Öcal et al., 2024]. Producing even a single high-quality asset is a labor-intensive and time-consuming process in which skilled designers must invest considerable effort, making 3D content creation one of the most resource-heavy parts of any modern visual pipeline [Jiang, 2024].

Nowhere is this bottleneck more visible than in the entertainment industry, and in particular in video game development. Visual assets typically consume the largest portion of a game's production budget: 2D and 3D art alone can range from roughly $10,000 for small indie projects to well over $500,000 for cinematic asset pipelines, while AAA titles routinely spend tens of millions of dollars on modeling, texturing, rigging, and animation alone [Pixune Studios, 2026]. This high cost is driven directly by the same factors highlighted above: 3D artists require specialized training, the workflow is slow and iterative, and each asset must be hand-crafted to fit a specific artistic direction. As a result, 3D content production has become one of the dominant cost drivers not only in games, but also in film, VR/AR experiences, and immersive entertainment more broadly.

Over the years, several intermediate approaches have tried to relieve this burden. Studios buy ready-made models from online asset stores or capture real objects through 3D scanning and photogrammetry, but both options are restricted to what already exists. A more flexible alternative is *procedural content generation*, where content is not modeled manually but produced by an algorithm following a well-defined procedure, with artists steering the result through parameters [Hendrikx et al., 2013]. Familiar examples include the endless terrain of *Minecraft*[1], the trees grown by tools such as *SpeedTree*[2], or the planets of *No Man's Sky*[3]. These methods work well for repetitive content like landscapes or foliage, but each generator has to be hand-engineered for one specific category and offers limited control over the visual style of an individual object.

Generative 3D modeling and, in particular, diffusion-based methods push this idea much further. Intuitively, such a model is trained on huge collections of 3D shapes until it learns what plausible objects look like; at generation time, it starts from pure random noise and gradually *denoises* it, step by step, into a coherent shape that matches a text prompt or a reference image. This makes it possible to produce a custom 3D model of almost any object directly from a short description, drastically reducing the time, cost, and expertise required to obtain usable 3D content [Jiang, 2024, Öcal et al., 2024]. Diffusion-based generative models facilitate the **automatic creation of complex 3D structures using minimal conditioning, such as text or image prompts**, thereby introducing new possibilities in model design and creative workflows. [Poole et al., 2022, Lin et al., 2023, Cheng et al., 2023, Jincheng et al., 2026]

Systems such as *DreamFusion* [Poole et al., 2022], *Magic3D* [Lin et al., 2023], and *SDFusion* [Cheng et al., 2023] show that combining diffusion models with implicit 3D representations can produce geometrically consistent and semantically rich 3D assets. Despite these advancements, current 3D model generators still lack user control and editing capabilities. Most existing approaches focus on **one-shot model generation** instead of supporting continuous, user-guided refinement.

A fundamental problem of this one-shot generation paradigm is the limited alignment between user intention and the final output. Because these models rely on sparse conditioning signals, typically text or a single reference image, the generated result is often an approximate interpretation rather than an exact realization of the desired user design. Small changes in the prompts can yield significantly different outputs, revealing the lack of control and predictability in the generation process. [Brack et al., 2023] As a result, users have little to no ability to iteratively change particular aspects of a model once it has been generated. Instead, modifying even minor details often requires restarting the generation process from scratch, as current systems lack efficient mechanisms for localized or incremental editing, leading to unintended edits in non-target areas. [Park et al., 2025] This limitation contrasts traditional design workflows, where iterative adjustments and reuse of existing assets are essential to obtain precise and user-intended results [Liang et al., 2025].

In this thesis, we attempt to address these problems by dividing them into **two major chal-**

[1] www.minecraft.net
[2] https://unity.com/products/speedtree
[3] https://www.nomanssky.com/

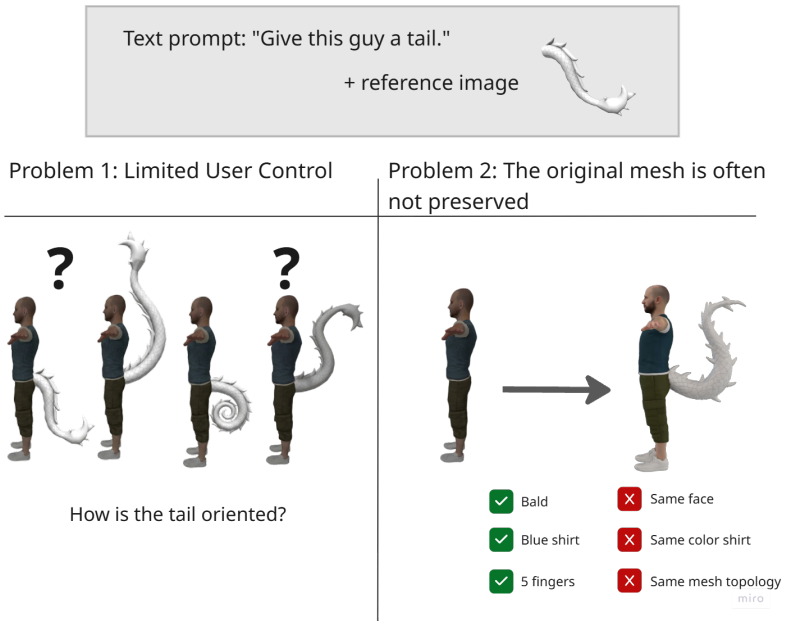

Figure 1.1: Visualization of the biggest challenges in current generative 3D model editing.

lenges. These are shown in Figure 1.1.

The initial challenge exists in the limitations of ***user control***. Current systems mainly depend on text prompts and/or reference images, giving users only indirect control over the design and style of the generated models. While natural language provides an interface for conceptual description, it falls short of accurately defining spatial structure, proportions, or local geometry. Recent studies have sought to address this by including additional conditioning methods, such as sketches or geometric constraints. For example, *SketchDream* [Liu et al., 2024] and *3D Shape Reconstruction from Free-Hand Sketches* [Wang et al., 2022] incorporate sketch-based guidance to generate models from 2D user input. Similarly, *ShapeCrafter* [Fu et al., 2023] enables iterative text-based refinement. These techniques greatly enhance creative direction but are primarily intended for creating entirely new 3D models rather than modifying or repurposing existing ones.

A second challenge is the absence of efficient mechanisms for **incorporating existing models** into the generative process. Most diffusion-based frameworks start from noise or a latent prior, making it difficult to integrate or partially preserve previously designed 3D assets. [Poole et al., 2022] In professional design practice, however, reusability and iterative edits are fundamental. Users often prefer to adapt existing geometry rather than regenerate it entirely. [Park et al., 2025] Some recent works have begun addressing this issue. *Instruct-NeRF2NeRF* [Haque et al., 2023] conditions NeRF optimization on textual editing instructions to modify existing scenes, while *CraftMesh* [Jincheng et al., 2026] includes a framework that enables high fidelity mesh manipulation via Poisson seamless fusion. However, these methods generally provide limited creative flexibility, as they are typically constrained to image and text prompts as conditioning signals and do not preserve the exact geometry of the original model during editing (since they often work with SDF).

This thesis proposes a fresh approach to *user-guided generative 3D editing* that tackles these challenges by combining user inputs of different types: text prompts, sketches, and reference images. The system supports interactive refinement of existing 3D models, providing real-time previews to guide user-driven modifications. Through this approach, users can iteratively adapt a 3D model until they are satisfied with the result.

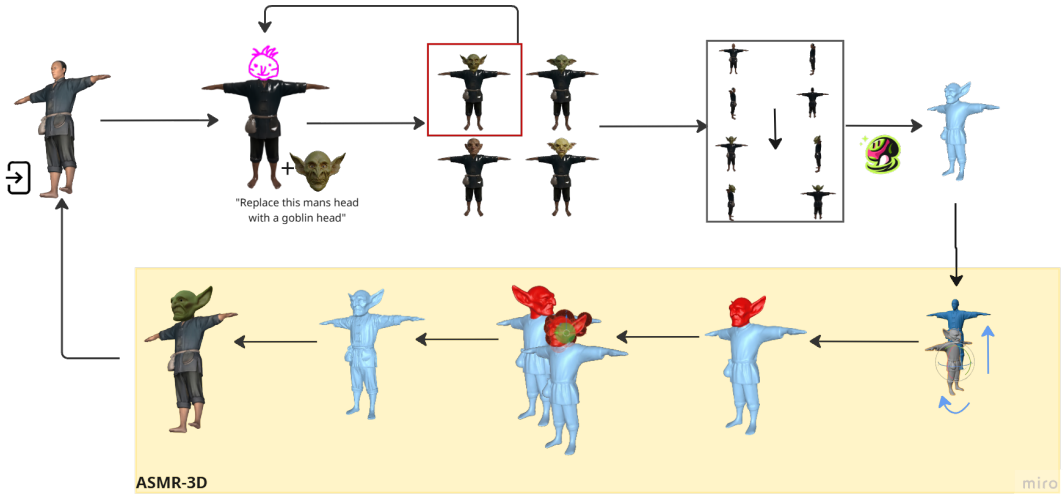

Figure 1.2: A visual representation of the developed pipeline.

The developed system showcases a method that merges the ease of use of 3D model generators with the accuracy of direct manipulation, organized as a flexible framework intended for multiple 3D modeling domains rather than concentrating on particular tasks such as furniture, characters, or architectural components. This is accomplished through an interactive pipeline, visualized in 1.2

At its core lies a region-based selective remeshing method developed and used in this research. Inspired by the remeshing approach used in CMD [Li et al., 2025, Palfinger, 2022] and the mesh fusion used in CraftMesh [Jincheng et al., 2026], the proposed technique merges the outputs of the generated models with the current geometry via localized reconstruction. This allows users to keep or change certain parts of a model, offering greater control over the generation processes while keeping the original model intact. We call this method ASMR-3D.

Alongside system design and implementation, this thesis features a usability study to evaluate the perceived usability of the proposed system for user-directed generative editing processes, as well as user satisfaction with the end result. These assessments aim to provide an understanding of how this user-driven, generative interface can improve the creative process.

Overall, the usability study indicates that participants responded positively to the proposed pipeline, suggesting that the system provides a viable and promising approach to user-guided generative 3D editing, although several aspects remain open for further refinement and improvement in future work.

Research questions:

Main RQ: How can generative AI be used to simplify user-guided editing of existing 3D models while preserving geometric fidelity and improving user control?

This is addressed through the following sub-questions:

RQ1: How can region-based selective remeshing be used to integrate newly generated geometry with existing 3D models?

RQ2: How can multimodal inputs (text, sketches, and reference images) improve fine-grained user control in generative 3D editing?

RQ3: To what extent can the proposed method preserve the original geometric structure during localized edits?

RQ4: How do users rate the usability and effectiveness of the proposed system in supporting iterative 3D model editing, as measured by the User Experience Questionnaire (UEQ) and a custom satisfaction questionnaire?

Chapter 2

Related work

2.1 Traditional and procedural 3D modeling techniques

Before diffusion-based generative models entered the picture, the computer graphics community spent several decades developing rule-based techniques for producing 3D content algorithmically. These traditional procedural approaches remain widely used in industry today and form the conceptual backdrop against which modern generative methods should be understood. Rather than learning a distribution over shapes from data, procedural methods describe geometry through explicit rules, grammars, or mathematical functions that an artist or technical designer authors and parameterizes [Hendrikx et al., 2013].

Foundational examples are *Lindenmayer systems*, or L-systems, originally introduced by the biologist Aristid Lindenmayer in 1968 to model the development of multicellular organisms [Lindenmayer, 1968]. An L-system is a parallel string-rewriting grammar: starting from an initial axiom, every symbol is simultaneously replaced according to a fixed set of production rules, and the resulting string is interpreted geometrically, typically through a turtle graphics scheme that translates symbols into commands such as "draw a segment" or "rotate by an angle". By repeating this rewriting step, surprisingly complex branching structures emerge from very compact rule sets, which makes L-systems particularly well-suited for modeling plants, trees, and other self-similar natural phenomena. Prusinkiewicz and Lindenmayer formalized and popularized this approach for computer graphics in *The Algorithmic Beauty of Plants*, introducing stochastic, context-sensitive, and parametric extensions that remain the basis of most modern vegetation tools, including commercial systems such as *SpeedTree* [Prusinkiewicz and Lindenmayer, 1990].

Procedural techniques have been generalized far beyond plants. For man-made structures, shape grammars play a similar role: a hierarchical mass model is iteratively refined by context-sensitive rules that split, scale, and decorate volumes into facades, windows, and roofs. The most influential example is the CGA shape grammar of Müller et al., which demonstrated that entire urban environments, including a virtual reconstruction of Pompeii, can be generated from compact rule sets with a high level of geometric detail [Müller et al., 2023]. Closely related are noise-based methods for terrain and texture, fractal subdivision schemes, and parametric modeling pipelines used in tools such as Houdini [1] and Blender's [2] Geometry Nodes.

Compared to the generative methods discussed in the next section, these procedural approaches offer strong guarantees: results are deterministic (or controllably stochastic), interpretable, and trivially editable by adjusting parameters or rules. Their main weakness is exactly the flip side of this strength: each generator must be manually engineered for a specific category of content,

[1] https://www.sidefx.com/
[2] https://www.blender.org/

and producing a one-off, semantically rich object from a free-form description is essentially out of scope.

2.2 Direct mesh manipulation and geometry-based editing

Before turning to generative methods, it is useful to look at how 3D models have traditionally been edited inside graphics software once they are already created. In tools such as *Blender*, *Maya*, or *ZBrush*, the user does not regenerate the object from scratch; they grab, push, smooth, or sculpt existing vertices, edges, and faces. To make this process less tedious than moving each vertex one by one, the computer graphics community has developed mathematical *deformation* techniques that propagate a small amount of user input across a large region of the mesh while preserving the surface's geometric detail.

A well-known example is the **Laplacian Surface Editing** framework by Sorkine et al. [Sorkine et al., 2004]. Instead of representing a vertex by its absolute position in space, the method stores each vertex relative to the average position of its direct neighbors, which they call a *Laplacian coordinate*. Because this coordinate captures *how a vertex sits inside its local neighborhood*, it acts as a compact description of the surface's intrinsic detail. The user then selects a Region of Interest (ROI), marks a small set of vertices as a *handle*, and translates or rotates that handle. The system reconstructs the surface by solving a sparse linear system that keeps every Laplacian coordinate as close as possible to its original value while respecting the new handle position. The visual result is that bumps, wrinkles, and other small features deform together with the global shape, giving the impression that the object behaves like a piece of clay or rubber when its handle is dragged. Around the ROI, the user can also fix a thin ring of vertices that stay in place during the deformation; these act as a boundary buffer and ensure that the edited region blends smoothly into the rest of the mesh, which is left completely untouched. Beyond simple deformation, the same representation supports *coating transfer*, in which the fine surface detail of one model is lifted from its underlying smooth shape and re-applied to another model, and *mesh transplanting*, where an entire patch from one object is sewn onto another with a gradual transition between the two surfaces.

A few years later, Sorkine and Alexa generalized this idea with the **As-Rigid-As-Possible (ARAP)** surface modeling method [Sorkine and Alexa, 2007]. ARAP looks at each small neighborhood of the mesh and asks that, during the deformation, it stays as close as possible to a pure rigid motion (rotation plus translation), without stretching or shearing. The deformation is then computed by iteratively alternating between estimating the best rotation per vertex and solving a linear system for the new positions. ARAP has become one of the most widely used handle-based deformation techniques and is shipped in many open-source and commercial 3D editors. Other operators in this family include cage-based deformation, free-form deformation (FFD), and skeleton/rig-based posing; each of them exposes a small number of intuitive controls that drive a much larger mesh.

These techniques are very powerful when the goal is to *reshape* an existing model: bending an arm, stretching a chair leg, or moving the tip of an ear. However, they have a number of important limitations for the use case studied in this thesis. First, they are purely geometric: they can move existing geometry around, but they cannot create new semantic content. There is no way to "ARAP" a hat onto a character or a wheel onto a vehicle if the underlying mesh does not already contain those parts. Second, edits that change the *topology* of the surface (adding holes, removing limbs, fusing two parts) are not natural in this framework and usually require separate cut, fill, or Boolean operations. Third, even though the math is hidden behind a handle, the interaction model still expects the user to understand concepts such as ROI, anchors, and handles. These tools live inside professional 3D software, which generally requires artistic skill and familiarity with complex interfaces, making it hard to access for layman users [Öcal et al., 2024]. Finally, the quality of the result depends strongly on the local mesh resolution and

triangulation: badly tessellated regions need to be remeshed before the deformation, adding yet another technical step.

For this thesis, direct mesh manipulation is relevant in two ways. It provides the conceptual baseline against which the proposed system should be compared: a designer who is comfortable with ARAP-style tools does not really need a generative method to bend an existing chair leg. At the same time, the limitations above explain exactly why a generative approach is needed for the type of edits we target: introducing new parts, replacing regions with different semantic content, or following a user prompt such as "make this a fantasy chair". These are operations that geometric deformation tools simply cannot perform on their own. This positions our work as *complementary* to direct manipulation rather than as a replacement.

2.3 Generative 3D-modeling

The emergence of generative artificial intelligence has fundamentally changed the way 3D content is created. Instead of relying entirely on manual modeling in specialized software, recent methods can automatically generate 3D objects based on relatively limited input, such as text descriptions, reference images, or sketches. These developments have been strongly influenced by diffusion-based models, originally developed for 2D image generation and subsequently extended to 3D applications [Poole et al., 2022, Lin et al., 2023, Cheng et al., 2023].

A first major line of research involves Score Distillation Sampling (SDS)-based methods, in which a pre-trained 2D diffusion model is used as a prior to optimize a 3D representation. DreamFusion [Poole et al., 2022] was an influential breakthrough in this area: a NeRF is optimized so that rendered images from different camera viewpoints match the samples from a text-to-image diffusion model.

Neural Radiance Fields (NeRF) is a method that encodes a 3D scene as a neural network. It learns from a small number of images and can then create new views of the scene from camera angles it has not seen before. A NeRF takes a 3D point and a viewing direction as input, and it predicts two things: how much light is at that point and how dense the scene is there. Then it uses these predictions, along with camera rays, to render a new image. [Gao et al., 2026]

Thanks to SDS, this can be achieved without specific 3D training data, which is particularly relevant given the limited availability of large-scale 3D datasets. Later methods, such as Magic3D [Lin et al., 2023], refined this idea with a two-step approach where a coarse structure is first formed, followed by a finer mesh. This made generation more efficient and enabled higher resolutions than with previous SDS approaches [Lin et al., 2023].

In addition to NeRF-based representations, implicit representations such as **signed distance fields (SDFs)** have become more and more important. SDFusion [Cheng et al., 2023] demonstrates that continuous distance functions can be used to generate detailed structures. At the core of the method is an encoder-decoder that compresses 3D shapes into a compact latent representation, upon which a diffusion model is learned, enabling multimodal inputs such as images, text, and partial shapes.

Another important trend in the literature is the expansion into alternative generative paradigms. Some methods combine generative adversarial networks (GAN's) with geometric representations to allow efficient 3D generation, while other works focus on multiview consistency or native 3D generation. These trends demonstrate that the field is evolving from indirect, 2D-driven 3D synthesis toward more structurally aware generation processes, in which geometry, texture, and control are modeled more explicitly [Wang et al., 2023].

This section provides the technical foundation for RQ1 and RQ2: SDS-based methods, NeRF, and SDF representations determine which generative building blocks are available for 3d model generation. The paradigms discussed also help contextualize the choice of multimodal input (RQ2) within a broader research landscape.

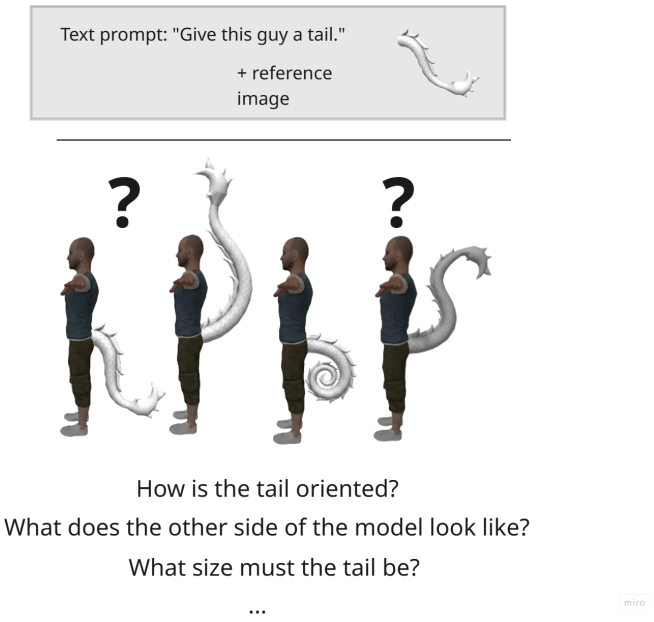

Figure 2.1: Visualization of the lack of user control in generative 3d modeling tools.

2.4 Limitations of generative 3D modeling tools

Despite rapid progress, generative 3D models continue to have major limitations. One of the most distinct limitations is the limited control over the generation process. Many systems still operate under a *one-shot generation* paradigm, in which a complete model is generated in a single step from a single prompt or reference. In practice, this often yields results that only partially match the user's intent. Furthermore, small changes in text can lead to large, sometimes unpredictable, changes in the results, showing a lack of reliability and reproducibility [Brack et al., 2023].

In addition, many existing methods do not adequately correspond to the way designers work in practice. Incremental refinement is central to traditional 3D workflows: a model is adjusted, checked, and reworked step by step. However, many generative systems barely allow for such local interaction, meaning that even small changes require a complete regeneration [Park et al., 2025].

A typical problem for single-image 3D generation is that it is very hard to predict the hidden parts of the object. The results can look random, unclear, or may not match what the user intended them to look like. The model must guess the unseen parts because there is not enough information given. [Chen et al., 2026]

One could argue that these problems can be solved by writing a more detailed prompt that describes every aspect of the object upfront. While better prompts do help to some degree, this approach quickly runs into limits in practice. Natural language is often unclear and not well-suited for describing exact shapes, proportions, or small stylistic choices, and even very detailed prompts tend to lose fine details during generation [Brack et al., 2023]. Small but important details, such as the exact curve of an edge, the position of a small decoration, or the alignment of a specific part, are often missed or misinterpreted by the model, no matter how carefully they are described. This shows that the issue is not just about writing a better prompt, but a deeper limitation of one-shot, prompt-based generation that cannot be fully fixed by prompting alone.

This problem is visualized in Figure 2.1.

An additional problem is structural inconsistency. When objects consist of multiple parts, errors

often arise in symmetry, alignment, or relative proportions. Without explicit mechanisms for local consistency, different parts of a model can become inconsistent with one another, leading to artifacts or unstable geometry. Finally, the scarcity of large-scale and diverse 3D datasets is still a key factor limiting the generalizability of generative models [Poole et al., 2022]. This is visualized on the right side of Figure1.1.

This section directly addresses all four research questions. The lack of control (RQ2), the one-shot regeneration issue (RQ1, RQ3), and the poor integration with iterative workflows (RQ4) are precisely the shortcomings that the proposed method aims to address. These limitations thus form the foundation of the thesis's problem statement.

2.5 Solutions and improvements in generative control

To tackle these limitations, recent literature is increasingly focusing on controllable and multi-modal generation. Instead of using text alone as input, new methods combine multiple forms of user guidance, such as sketches, reference images, or supplementary textual descriptions. Sketch-based systems, such as *SketchDream* [Liu et al., 2024], use freehand drawings to guide the overall shape or silhouette of an object. This reduces the ambiguity inherent in text alone and gives the user more direct control over the geometric intent of the result.

A related direction is personalization. Methods such as *DreamBooth3D* [Raj et al., 2023] make it possible to customize a model to a specific object or a particular identity based on a few reference images. This is helpful for applications where consistency between multiple outputs is important, such as avatars, games, or personalized digital content.

In addition to multimodal input, iterative generation is another key area for improvement. Systems such as *ShapeCrafter* [Fu et al., 2023] make it possible to improve a model step by step through successive prompts or instructions. This is consistent with the incremental nature of design practices, in which users typically first establish a general direction and then gradually refine it. Such systems indicate that generative 3D modeling does not necessarily have to be a static, one-time step, but can also be an interactive process.

Recent studies are also exploring new methods to further improve the quality of geometric predictions. *Tactile DreamFusion* [Gao et al., 2024], for example, uses tactile information to generate more accurate geometric details.

Together, these developments indicate that the field is evolving toward more controlled and user-centered 3D generation, in which input from multiple modalities is leveraged to reduce ambiguity and achieve more predictable results.

This is highly relevant to RQ2: these works demonstrate that multimodal input (sketches, reference images, structured text) reduces the ambiguity of purely textual prompts. Iterative systems such as *ShapeCrafter* also align with RQ4, which focuses on user-centered, step-by-step interaction. However, these methods do not yet provide a solution for local mesh integration (RQ1, RQ3), underscoring the necessity of the proposed work.

2.6 3D model editing, region-based editing and remeshing

In addition to full-scale generation, the editing and reuse of existing 3D models is an important area of research. In professional workflows, objects are rarely completely rebuilt from scratch and only a limited portion is modified while the rest of the geometry is preserved. This requires a process in which changes are applied locally, and the remaining parts of the model remain unaffected. In recent literature, this is approached as a form of generative editing, in which new content is generated for a selected region and then integrated into the original model [Erkoç et al., 2024, Jincheng et al., 2026].

A key challenge here is that the processed mesh must not only be semantically correct but also geometrically and visually consistent with the original. The unedited regions must remain identical, where the transition zone between old and new must be completely seamless [Erkoç et al., 2024]. In practice, a local modification often affects normal directions, topology, or texture continuity. Therefore, techniques from seamless editing are employed to soften these boundaries and improve visual integration.

In the context of local editing, *SDFusion [Cheng et al., 2023]* introduces a shape infilling capability that reconstructs missing or masked regions of a 3D shape while preserving the surrounding geometry. Instead of regenerating an entire object, the method operates on partially observed signed distance fields, where a subset of the volume is masked out. During inference, the diffusion process is conditioned on the known geometry, enabling the model to synthesize only the missing regions while respecting global structure and local continuity. This formulation ensures that the generated content aligns with the existing shape, producing coherent completions that naturally blend with the unedited parts.

This is the most directly relevant literature for RQ1 and RQ3. The principle that only selected regions are modified while the rest of the geometry remains identical is at the core of our method. SDFusion's shape infilling provides a conceptual template, but operates on SDFs rather than direct meshes, which is a key difference for the chosen approach.

2.6.1 Local editing using semantic regions

To enable this local control, region-based editing is often used, in which the model is divided into semantic regions or components. Using modern segmentation networks such as *P3-SAM* and *SAMPart3D* [Ma et al., 2025, Yang et al., 2024], meaningful parts (e.g., arms or legs) can be automatically identified. Methods such as 3D Highlighter [Decatur et al., 2022] also use textual cues to delineate editing regions, allowing users to modify a single segment in a targeted manner.

A powerful example of this is *PrEditor3D* [Erkoç et al., 2024], in which the object is projected onto a few images, local edits are performed via text prompts and masks, and a new 3D model is then reconstructed. Next, a 3D segmentation determines exactly which parts have actually been modified, after which a merging algorithm seamlessly integrates the edited segments. In this way, only the intended regions remain changed, and the rest of the model is preserved exactly.

While these segmentation-based approaches give clean and well-defined editing boundaries, they also come with important drawbacks. Because the edit region is determined by a segmentation network, the user can only edit parts that the network is able to recognize and label. This works well for common categories such as arms, legs, or wheels, but it becomes much harder when the user wants to modify a region that does not match any predefined semantic part, or when the desired edit spans multiple parts at once [Erkoç et al., 2024]. Therefore, we chose to take a different approach.

2.6.2 Local edits using reference images

CraftMesh [Jincheng et al., 2026] is an innovative mesh-editing system that utilizes generative models in 2D and 3D. Instead of regenerating the entire model, *CraftMesh* breaks the task down into three phases. First, it edits a 2D image of the object using a text- or image-driven model, after which region-specific 3D meshes are generated for the modified areas. These new mesh segments are then seamlessly added to the original model via Poisson-based fusion, preserving the unmodified geometry and adjusting only the boundary zone around the edit.

In the second step, *CraftMesh* generates processed region meshes as a middle result. The processed reference image is combined with the original model using a 3D generator (such as CraftsMan3D) to produce the processed reference mesh, which follows the global shape but contains less detail. At the same time, only the modified part of the image is used to create the

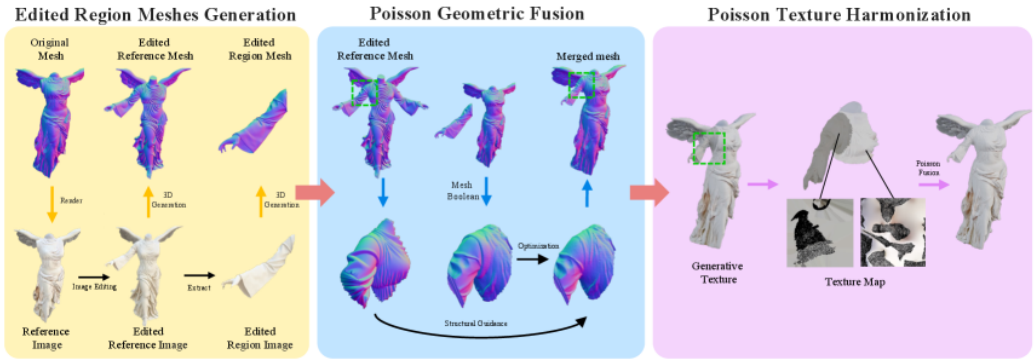

Figure 2.2: Overview of CraftMesh pipeline with region-specific 3D-Editing and Poisson Fusion, reproduced from [Jincheng et al., 2026].

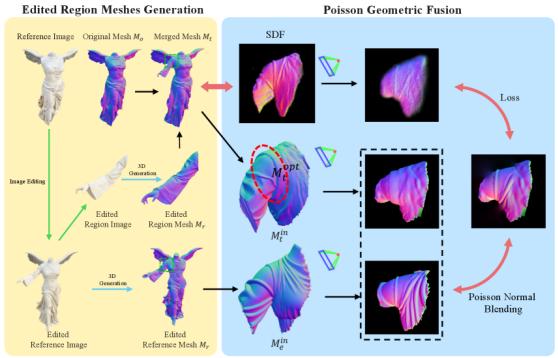

Figure 2.3: Generation of region-specific mesh in CraftMesh, reproduced from [Jincheng et al., 2026].

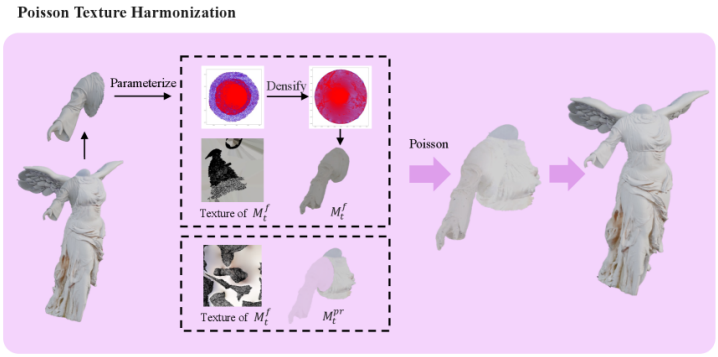

Figure 2.4: Poisson Geometric Fusion pipeline in CraftMesh, reproduced from [Jincheng et al., 2026].

edited region mesh, which contains highly detailed local geometry. This combined approach makes sure that the final result remains both realistic in terms of overall shape (via the reference mesh) and rich in detail within the edited area.

In the third step, **Poisson Geometric Fusion** takes place. First, a Boolean operation (union or difference) is applied between the original mesh and the processed region mesh, resulting in a sparse overlap with boundaries that do not yet align. CraftMesh then improves only this transition area using a hybrid **SDF/mesh representation**, new vertices are added locally, or existing ones are merged to keep the surface smooth. The key idea is the blending of the normal vector fields of both the edited reference mesh and the region mesh, which are calculated and combined via a Poisson fusion. The result is a "blended" normal that preserves fine details within the edited area and makes a smooth transition to the outside of it. The SDF is then optimized to adjust the generated mesh so that it matches this combined normal. This way, the new geometry perfectly connects with the original. The unedited parts remain unchanged, and only the boundary zone around the edited area is revised for a perfect fit.

After geometric processing, CraftMesh also ensures the texture blends well with the rest of the model. The newly added parts often have their own generated texture that does not automatically match the color of the original surface. CraftMesh therefore first parameters the texture of the new part and creates a dense 2D mesh of the pixels. It applies a Poisson solution to this 2D mesh, so that colors at the edge of the edited area blend smoothly into the existing texture. As a result, the color transitions blend naturally into one another, making it seem like the object has always been a single entity.

Thanks to this combined approach of 2D/3D generation, smart segmentation, and Poisson blending, the final result remains geometrically coherent and visually consistent. CraftMesh demonstrates that local mesh editing is possible without interrupting the entire model. The rest of the object is preserved, while the edited part is realistically integrated, keeping both global structure and detail.

This can be used for our implementation; the downside is that it does not keep the geometry of the model because it has to transfer it into SDF space first. This counteracts RQ1 and RQ3.

2.6.3 Controllable Multiview Diffusion (CMD)

Controllable Multiview Diffusion (CMD) [Li et al., 2025] is a 3D editing pipeline that lets users modify one or more rendered views of an existing mesh and propagate those edits into the full 3D model. In CMD, you first render color and normal maps from the original mesh, edit a target view in 2D, and then use a multiview diffusion model (an extension of ControlNet [Zhang et al., 2023]) to generate updated color and normal maps for all views. Finally, CMD updates the

mesh by incrementally changing its vertices using a method called continuous remeshing. Small moves are made so the mesh's projections match the new normals, while splitting or collapsing triangles as needed. A Laplacian smoothing term is applied during this reconstruction to keep the surface smooth and avoid artifacts. The figure below, Figure2.5, illustrates the progressive part-by-part generation pipeline of CMD.

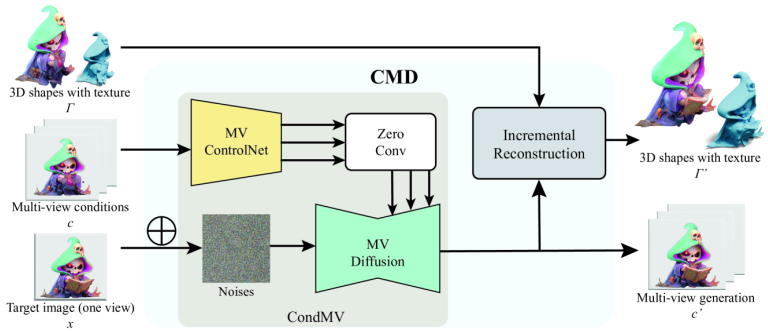

Figure 2.5: Progressive 3D-generation pipeline of CMD. [Li et al., 2025]

CMD pipeline steps:

- Render and Edit: Render one or more views (color and normal) of the input mesh and apply user edits to a chosen view.
- Multiview Diffusion: Feed the edited view and original renderings into a conditional multiview diffusion model (a multiview ControlNet system called CondMV) to generate new consistent color and normal maps for all views.
- Incremental Reconstruction: Starting from the original mesh, iteratively adjust vertices so that rendered normals match the generated normals. In each iteration, adapt the mesh topology locally, split edges/triangles where more detail is needed, and collapse small ones where it isn't. Apply Laplacian smoothing to keep the surface smooth.
- Texture Baking: After geometry is updated, bake the new color maps onto the mesh to produce the final textured model.

This paper uses Palfinger's continuous remeshing for inverse rendering (2022) [Palfinger, 2022]. Starting out from multiview normal maps and an initial mesh (e.g., a sphere), it iteratively reconstructs a consistent surface via continuous remeshing.

2.7 Continuous remeshing

This section will give more information about how continuous remeshing works, since it forms the basis of the selective remeshing implementation described later. We start with some background information so that the concepts can be fully understood.

Rendering vs Inverse Rendering

Forward rendering (usually just called rendering) means going from scene parameters to images. Given the mesh, camera, lights, and materials, a renderer computes how light reacts with the surfaces and returns the final image.

Inverse rendering solves the opposite problem: we are given one or more images and aim to recover unknown scene parameters such as shape, materials, or lighting. Instead of asking "what image do I get from this scene?", we ask "what scene would produce this image?". This is usually formulated as an optimization problem, where we search for parameters π (geometry, materials, etc.) that minimize some loss between the rendered images $\mathcal{R}(\pi)$ and the observed images.

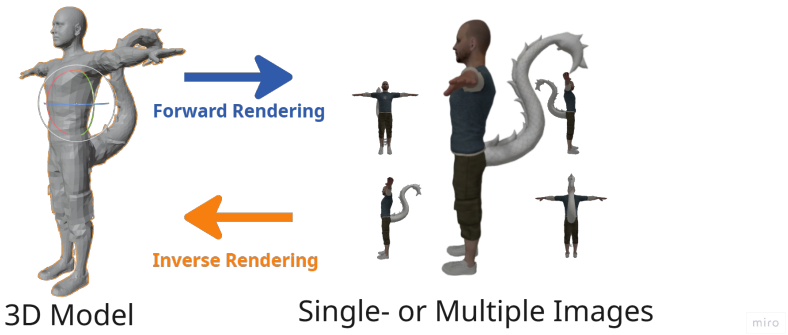

Figure 2.6: Visualization of Forward and Inverse Rendering.

These concepts are visualized in Figure 2.6.

Optimization in Graphics

Many modern inverse rendering methods use gradient-based optimization. [Loper and Black, 2014, Yan et al., 2025] We define a loss function

$$\mathcal{L}(\pi) = \text{difference}\big(\mathcal{R}(\pi), I_{\text{target}}\big)$$

where I_{target} is a target image (or a set of images) and $\mathcal{R}(\pi)$ is the rendered image. We then update the parameters π in the direction that reduces this loss, for example, using gradient descent or Adam [Yan et al., 2025]. In practice, this means we repeatedly render images, measure the error, compute the gradient of the error with respect to the parameters, and take a small step to improve the scene [Loper and Black, 2014].

Differentiable Rendering

Differentiable rendering includes techniques that allow the rendering process to be differentiable. This means that gradients from image space losses can be sent back to 3D scene parameters like geometry, materials, camera position, and lighting. It connects 2D image evidence with 3D optimization. This enables inverse graphics pipelines to learn from rendered images instead of needing detailed 3D supervision. The literature presents several formulations, including methods based on mesh, voxel, point cloud, and implicit representation. Each of these is aimed at providing useful gradients for optimization [Kato et al., 2020].

Differentiable rendering is the key building block that enables methods such as Palfingers continuous remeshing to optimize geometry and appearance from images. [Palfinger, 2022, Li et al., 2025]

2.7.1 Mesh Representation and Remeshing

Mesh Structures

We use triangle meshes as our surface representation. They consist of Vertices, Edges, and Faces, as visualized in Figure 2.7. Each vertex $\mathbf{v}_i \in \mathbb{R}^3$ stores a 3D point, and each face $f_j = (i, k, l)$ is a triangle connecting three vertices. On top of this, we can store per-vertex or per-face attributes such as normals or colors, which are interpolated across triangles during rendering.

A very dense mesh with many small triangles can approximate fine geometric details, but is expensive to render and optimize. A coarse mesh is cheap to process but cannot represent sharp features or small bumps well. We could say that a good vertex count is therefore important for both reconstruction quality and runtime speed.

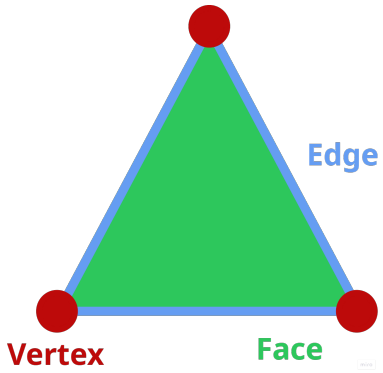

Figure 2.7: Visualization of triangle mesh composition.

Limitations of Fixed Meshes

If we keep the mesh connectivity fixed during optimization, we quickly run into problems. Large gradient steps on the vertices can create mesh defects such as flipped triangles, crumpled regions, and self-intersections. Also, a fixed triangulation cannot easily adapt to regions where we suddenly need more detail, for example, around sharp edges or small geometric features. [Palfinger, 2022]

In practice, fixed meshes often require careful manual adjustment of the initial resolution to find a trade-off between accuracy and performance. Even then, some areas may still be under-resolved while others are over-resolved, which is inefficient and can harm reconstruction quality. These issues motivate methods that can adapt the mesh resolution automatically during optimization. [Palfinger, 2022]

Remeshing Concepts

Remeshing means changing the mesh connectivity and sometimes the vertex positions to improve mesh quality or adapt the mesh to a target function. Typical local operations include edge splitting (adding vertices and triangles) and merging (removing vertices and triangles).

Edge splitting (also called subdivision) is used when the current distribution of, for example, vertices is too coarse. Merging (or coarsening) is used when the mesh is too dense, and many triangles do not contribute much to the approximation. By combining these operations with suitable quality metrics (for example, edge length or triangle shape), remeshing can equalize triangle sizes and remove badly shaped "sliver" triangles.

In the context of inverse rendering, adaptive remeshing allows the mesh to grow more triangles in parts where the image loss is high and remove triangles where they are not needed. This makes optimization stronger and often improves both speed and accuracy, compared to keeping the mesh fixed. [Palfinger, 2022, Li et al., 2025]

2.7.2 Continuous Remeshing Method

This section describes the continuous remeshing component used in CMD for 3D reconstruction. All mathematical formulas and definitions are quoted from the provided sources. [Li et al., 2025] [Palfinger, 2022]

Problem Formulation

The input is an initial mesh $\mathcal{M}_0 = (\mathbf{V}_0, \mathbf{F}_0)$ (e.g., a sphere or the original mesh) and sets of target multiview color images c_i, normal maps n_i and camera parameters. Let $\mathbf{V}$ denote the current vertex positions. Using a differentiable renderer, we compute rendered color maps $\hat{c}_i(\mathbf{V})$ and normal maps $\hat{n}_i(\mathbf{V})$ for each view.

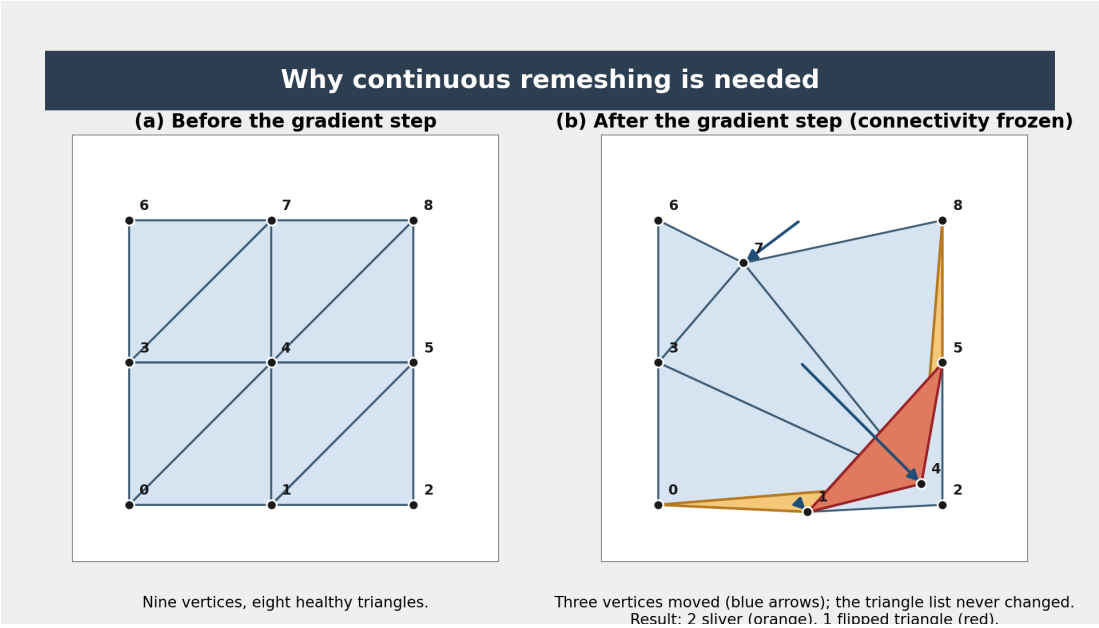

Figure 2.8: The core problem. **(a)** A clean 3×3 grid with nine vertices and eight healthy triangles. **(b)** After one gradient step, three vertices have moved (blue arrows), but the triangle list never changed: two triangles have collapsed into slivers (orange), and one has flipped its winding (red). The repair operations introduced later in this section act on exactly these defects.

CMD minimizes a reconstruction loss

$$\mathcal{L} = \mathcal{L}_2(n', \hat{n}') + \mathcal{L}2(c', \hat{c}') + \mathcal{L}\text{smooth},$$

where $\mathcal{L}2(\cdot, \cdot)$ is the squared L_2 loss between predicted normals n' and target normals $\hat{n}'$ and similarly for colors. The term $\mathcal{L}$smooth is a Laplacian smoothness regularizer.

This loss is optimized (e.g., with Adam) over $\mathbf{V}$, subject to regularization. In CMD, $\mathcal{L}_{\text{smooth}}$ is weighted by a parameter to preserve mesh smoothness.

Continuous Remeshing Idea

The optimization described above only moves vertices; it never changes which vertices are connected to which. The mesh starts with a fixed connectivity, and the optimizer simply pushes the existing vertices around. This is the central problem that continuous remeshing solves.

The problem with vertex updates alone. When a gradient step pulls vertices toward the target shape, some edges get stretched, and others get compressed. Because the connectivity is frozen, neighboring triangles cannot rearrange themselves: long edges turn into long, thin slivers, and in regions of strong curvature, triangles can even fold over and end up pointing the wrong way (so-called *flipped* triangles). Figure 2.8 illustrates this on a 3×3 grid: panel (a) is the clean starting mesh; in panel (b), three vertices have been pushed by a gradient step, but the triangle list is unchanged, which turns two triangles into slivers and one into a flipped triangle. A differentiable renderer cannot produce a clean image from such a mesh, and the loss gradients it returns from those broken triangles point in the wrong directions, which makes the next step worse. Palfinger's solution [Palfinger, 2022] is conceptually simple: after every gradient step, run a short local pass that repairs the connectivity, instead of letting bad triangles accumulate.

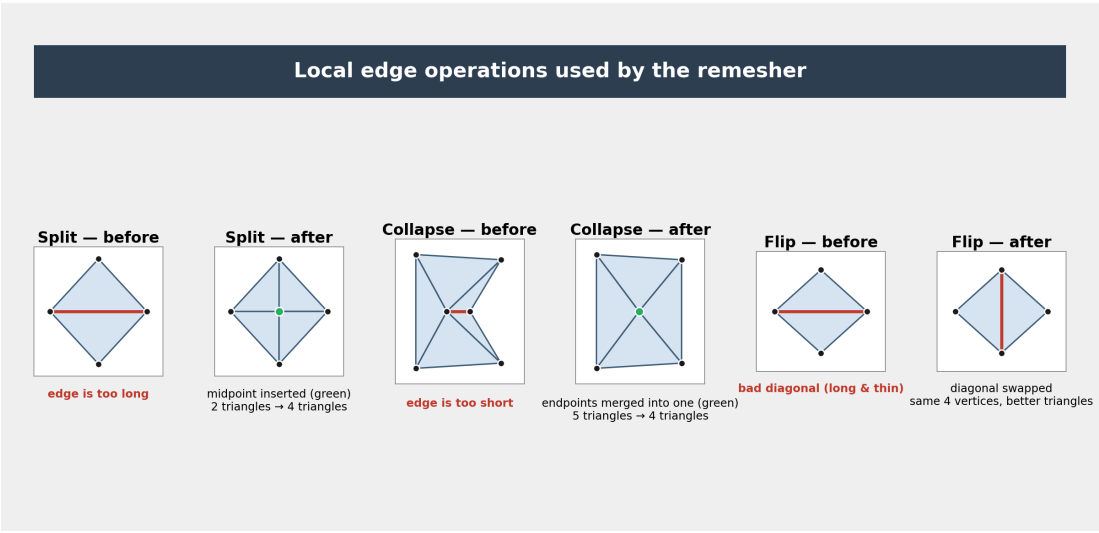

Figure 2.9: The three local edge operations applied by the continuous remesher. **Split:** a long edge (red) is bisected, and a new midpoint vertex (green) is inserted; the two adjacent faces are re-triangulated. **Collapse:** a short edge (red) is removed by merging its two endpoints into one (green). **Flip:** the shared diagonal of two adjacent triangles is swapped to the other diagonal; only the connectivity changes.

Three local edge operations. The repair pass uses three cheap operations on individual edges, illustrated in Figure 2.9. A *split* cuts a too-long edge in half and inserts a new vertex at its midpoint, replacing two triangles with four. A *collapse* merges the two endpoints of a too-short edge into a single vertex, removing one triangle on each side of the edge. A *flip* swaps the shared diagonal of two adjacent triangles when the alternative diagonal would produce better-shaped triangles, without adding or removing any vertex.

To decide what to do where, the remesher compares the current length $\ell(e) = |\mathbf{v}_i - \mathbf{v}j|2$ of every edge against a target length $\ell\text{target}(e)$. The ratio $\delta(e) = \ell(e)/\ell\text{target}(e)$ tells it whether the edge is too long ($\delta \gg 1$: split it), too short ($\delta \ll 1$: collapse it), or fine (leave it). In Palfinger's words, "by immediately collapsing suspicious triangles, we avoid and heal mesh defects".

Closed-loop control of the target length. The interesting design choice is not *which* operations exist, but how ℓ_{target} is set. A naive remesher uses a single target length everywhere, which forces a trade-off: short enough to resolve detailed regions wastes triangles on flat ones, while long enough to keep flat regions cheap loses detail where it matters. Palfinger instead lets ℓ_{target} vary across the surface and updates it in a feedback loop: "we use a closed-loop-controlled location-dependent edge length" [Palfinger, 2022].

The loop is intuitive. After each remeshing pass, the system looks at what just happened in every local region. If a region keeps triggering splits, the gradient is repeatedly asking for finer detail than the current target allows, so the target length there is shortened. If a region keeps triggering collapses, the mesh has more triangles than the geometry needs, so the target length is lengthened. Over several iterations, the target-length field self-organizes into something like Figure 2.10(b): a short target (red) inside regions with detail, a long target (yellow) on flat areas. The triangle distribution follows automatically – dense triangles where the geometry needs them, sparse triangles where it does not.

Why interleave with optimization? Because the remeshing pass is cheap and runs at every step, the mesh never has the chance to drift into a broken state: splits and collapses immediately undo the slivers and flips that the latest gradient update was about to create. Compared with the alternative of doing a single large remesh every few hundred iterations, this

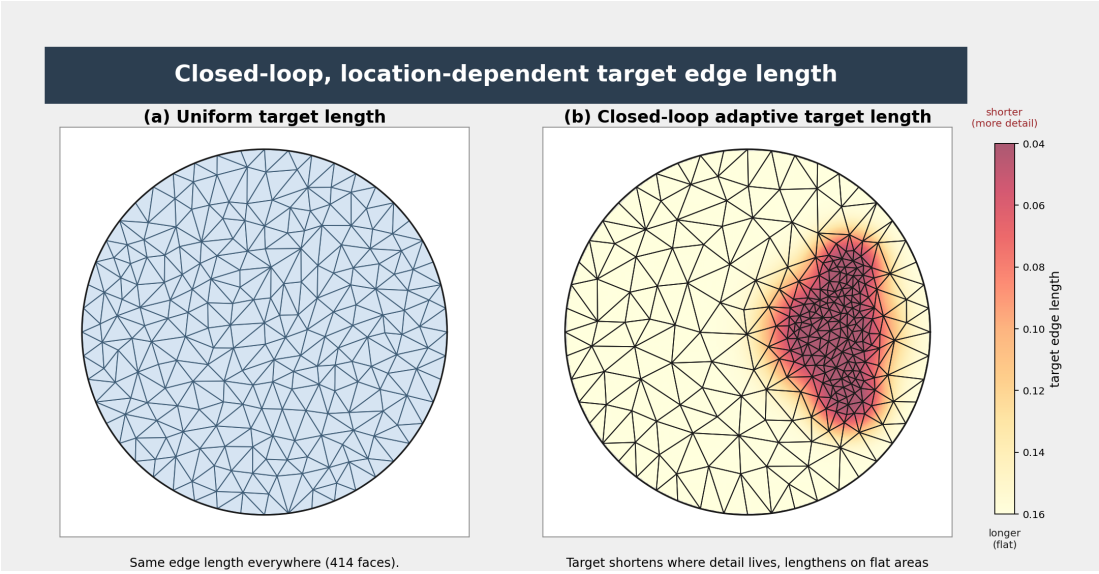

Figure 2.10: The effect of closed-loop edge-length control on a synthetic example with one detailed region inside an otherwise flat domain. **(a)** With a uniform target length, the mesh is uniform: the flat area overspends triangles, and the detailed region is still under-resolved. **(b)** The closed loop has shortened the target where geometric detail lives (red overlay) and lengthened it on flat areas (yellow overlay). With a comparable face count, the detailed region is now densely sampled while the flat surface is described efficiently.

continuous approach produces fewer defects, better triangle quality, and in practice needs far less per-scene parameter tuning [Palfinger, 2022].

2.8 Know3D

Know3D [Chen et al., 2026] tries to solve a common problem in single-image 3D generation: the hidden parts of an object are hard to predict, so the result can look random, unclear, or not match user intention. The paper says that this is especially difficult when only one image is available, because the model must guess the unseen parts.

The main idea of Know3D is to use a vision language model to give more semantic knowledge to the 3D generation process. The system uses Qwen2.5-VL to understand the input image and a diffusion model, Qwen-Image-Edit-2511, to generate the unseen back view of the object

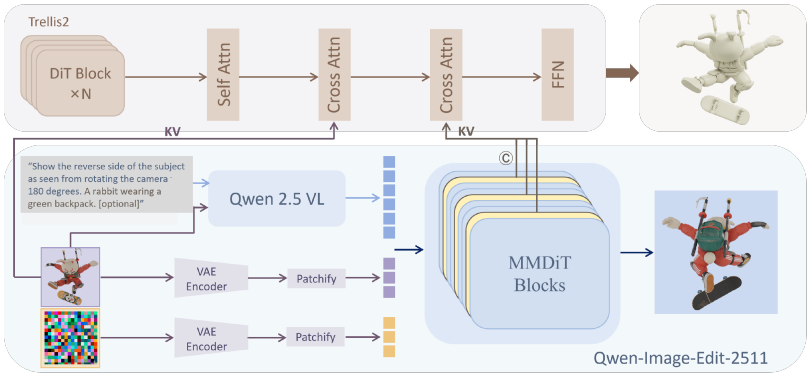

Figure 2.11: The pipeline of Know3D. [Chen et al., 2026]

from both the front image and a text description of the back side. The paper explains that this diffusion model is trained on front-and-back image pairs from 3D assets, with text labels for important back-view parts.

For the 3D step, Know3D does not use the generated image directly. Instead, it takes hidden features from the diffusion model and uses them as guidance for the 3D generator. These features are added through a separate cross-attention branch in a TRELLIS2-based 3D model, while the original attention paths stay unchanged. The paper also says that the 3D generator works in two stages: first, it makes a rough 3D structure, and then it improves the geometry into a more detailed final shape.

The experiments show that Know3D performs well on TexVerse meshes and HY3D-Bench, and it gives strong semantic matching and good geometry compared with other methods. The paper also reports that using hidden states from an earlier diffusion timestep works best. The pipeline is shown in Figure 2.11

Know3D is relevant to RQ2: it demonstrates that semantic enrichment via vision-language models helps clarify ambiguous user intentions. However, it focuses on single-image generation of complete objects rather than local editing, which means it only partially addresses RQ1 and RQ3. The concept of semantic guidance can, however, provide inspiration for processing textual prompts in the proposed system.

2.9 Multimodal interaction and iterative editing

Generative editing systems for images and 3D models can receive various forms of user input to provide control and precision. Key modalities include textual prompts, image-based cues (such as reference images or masks), user interface actions, and combinations thereof. [Shuai et al., 2024]

In addition, structured prompts can be used to assist users in drafting detailed editing instructions. Moenning et al. introduce a system in which short textual prompts are expanded by a vision–language model into long, structured JSON schemas. [Gutflaish et al., 2025] This allows a user to start with a concise description, after which a VLM automatically generates a comprehensive, formal prompt. Crucially, this model is designed such that changing a single attribute in the structured prompt typically modifies only the corresponding visual feature in the image. This means that the prompt control achieves a strong decoupling of image aspects: a single change in the prompt affects only the corresponding part of the generated result.

Interactive, iterative editing and feedback are also essential for user control. Liang et al. describe an agent framework (RefineEdit-Agent) that uses LLMs and LVLMs to perform complex, step-by-step edits based on user instructions. [Liang et al., 2025] In this system, a language model parses the instructions and plans the editing steps, while a vision language model judges the output and provides feedback in a closed loop. This allows the system to improve its output iteratively. After each step, the context is preserved, and the process can be adjusted to maintain both fidelity to the user's assignment and visual consistency. Such methods emphasize that user-driven iteration, in which the system continuously combines user feedback with its own assessment, is important for accurate and controlled generative editing.

These are particularly relevant to RQ2 and RQ4: structured prompts and iterative agent frameworks demonstrate how user input can be effectively decoupled and how iterative feedback improves the user experience. These insights inform the design of the user interface and the evaluation framework using UEQ, although the works discussed focus primarily on 2D images rather than 3D meshes.

2.10 3D Texture Synthesis and Local Retexturing

In 3D pipelines, a model is almost never just geometry; the visual appearance of an object also depends strongly on its *textures*. A texture is a 2D image that is wrapped around a 3D surface to describe color, roughness, normal information, and other material properties. The connection between the 2D image and the 3D surface is established through *UV mapping*: the artist cuts the mesh along seams and unfolds it into a flat layout in a 2D coordinate system called UV space. Each vertex receives a pair of (u, v) coordinates that tell the renderer which pixel of the texture image corresponds to that vertex. After UV unwrapping, a color map (often called the *albedo*) is painted or generated for and is then sampled by the renderer for every visible point on the surface.

A common workflow in production is *texture baking*, in which information from a complex source is precomputed and stored in a texture image. The most typical case is baking from a high-polygon to a low-polygon model: a detailed sculpt is used to generate normal, ambient occlusion, and displacement maps that are then read by the low-poly version, so the cheap model still appears highly detailed at render time. Generative 3D pipelines reuse the same idea: many systems first reconstruct geometry and then *bake* a learned color signal onto the UV layout, so the final asset is a standard mesh plus a standard texture image that can be loaded into any 3D engine [Li et al., 2025].

More recently, diffusion models have also been used to synthesize textures directly. *Text2Tex* [Chen et al., 2023] generates textures for an existing mesh from a text prompt by rendering the model from several viewpoints, applying a depth-conditioned 2D diffusion model to inpaint each view, and projecting the results back onto the UV map. A per-view "generation mask" tracks which faces have already been filled in, so subsequent views only generate the missing or low-quality areas, which reduces seams and stretching. Related multi-view texture-synthesis methods have shown that high-quality, semantically consistent textures can be produced without any manual painting. These approaches, however, typically regenerate the entire texture of the object, which is undesirable when the user only wants to repaint a single region.

Local retexturing is the more focused problem of changing the appearance of one part of a model while keeping the rest of the surface (and ideally the underlying geometry) intact. As mentioned before, *CraftMesh* [Jincheng et al., 2026] does this as follows: once the new region of the mesh has been generated, its texture is parameterized together with the surrounding original texture, and a 2D Poisson solver is used to smoothly merge the two color fields along the boundary. The Poisson formulation matches the gradients of the source texture inside the edit region while constraining the boundary pixels to the colors of the original surface, which produces a transition that looks continuous even when the two textures were generated by different models. Alternative options include simple alpha-based blending of the two textures along the edge of the edit, *projective painting* (rendering the new region from one camera, projecting it back onto the surface and feathering the boundary in image space) and re-baking the merged albedo onto a single new UV layout so that the final model only carries one texture set.

For this thesis, local retexturing is directly relevant to RQ1 and RQ3. Because the proposed system aims to preserve the exact geometry of the unedited parts of the mesh, it must also preserve their original appearance: replacing the entire texture would visually break the parts of the model that, geometrically, were not touched. Understanding how UV-based pipelines, Text2Tex-style synthesis, and Poisson blending in CraftMesh handle this trade-off informs the design of the retexturing step in our pipeline, which has to repaint only the locally regenerated region while staying visually consistent with the original asset.

2.11 User control in image generation models

Modern text-to-image diffusion models such as *Stable Diffusion* can produce highly detailed pictures from a short sentence, but text alone is a very limited interface for design. Many things are hard to put into words: where exactly something should sit in the image, what its silhouette should look like, or how its style should match a specific reference. Several lines of work, therefore, add extra input *modalities* on top of the text prompt to give users more direct control.

The most influential method is **ControlNet** [Zhang et al., 2023]. Next to the text prompt, the user feeds in a guidance image, such as a sketch, line drawing, depth map, segmentation mask, or pose skeleton, and the model produces a picture that follows both the text and the structure of the hint. ControlNet tends to follow the guidance image quite strictly, which is great when the user wants exactly that silhouette, but it also means that rough or imprecise sketches can dominate the output more than intended.

A more flexible approach is taken by *Sketch-Guided Text-to-Image Diffusion Models* [Voynov et al., 2022], which use the sketch only as a *soft* spatial hint. A small predictor is added on top of a pretrained diffusion model and pushes generations to roughly match the target sketch, but without forcing them to copy every stroke. The system, therefore, respects the general shape and position suggested by the user while leaving the model free to interpret the details, which works well even on freehand drawings. This is particularly useful when the sketch is meant to express the *idea* of an edit rather than its exact contour.

A different direction is exemplar-based generation. **Paint by Example** [Yang et al., 2022] takes an input image with a masked region and a separate reference image and inpaints the masked region with content that semantically matches the reference while leaving the rest of the image unchanged. Instead of describing what should appear inside the mask in words, the user simply provides a picture of it, which is a much more direct way to express "put something like this here" than text or style transfer alone.

For this thesis, this body of work is directly relevant to RQ2. In our system, we first generate images to later generate a 3D model. We do this to impose more creative control early on in the pipeline. By leveraging image-generation techniques that provide fine-grained control, we can produce a final 3D result that is more closely in line with the intended concept.

Chapter 3

System Overview

This thesis aims to overcome the limitations typically found in modern Generative 3D model editing systems. We divide it into 2 challenges: the first is adding more user control to enhance creative direction. The second one is preserving the vertices that should not be edited. These are shown in Figure 3.1.

We aim to tackle these challenges by creating a general-purpose 3D model editing system designed to be used after generating a 3D model with a one-shot 3D model generation system, such as Meshy. The user will start with any mesh (furniture, avatar, animal, house, ...), propose an edit, and receive the edited mesh. Users can thus go through the system multiple times, iteratively, to apply certain edits to a model. This system is designed for people with no experience with 3D modeling software like Blender and limited knowledge of 3D modeling. Every "difficult" aspect should be handled by the system, while allowing the user to fine-tune. We make use of existing AI tools like Gemini and Meshy to make it more user-friendly. The implementation should be able to run locally and make the edits in a reasonable time.

This system will serve as a high-level prototype to evaluate the effectiveness and usability of a pipeline-based system, shown in Figure 3.2. The following chapters will detail the methods and challenges of each step.

The pipeline aims to give users more control and creative direction by visualizing the final model's output early on with AI-generated images. These images are generated by a prompt, a sketch drawn over a single viewpoint of the model, and an optional reference image. As a result, users can get an early sense of the final model, making it easier to guide creative vision by editing generated images rather than editing the final model at the very last step, which would be more time-consuming. Additionally, this process helps users fill any creative gaps left out in the prompts.

Based on these images, a 3D model is generated from scratch. The user could use this model as is, or integrate the changes into the model they uploaded. This second part of the pipeline (bottom row on Figure 3.2) will lean further into the second challenge: preserving the original topology. The generated model will retain aspects of the original model because it is trained on these images. It will have similar features, such as the general shape and colors, but it will not match the exact details or topology of the original mesh. To address this issue, the pipeline proposes a process that automatically selects and normalizes the region to be edited. This selected region will be merged with the original model to have the final model as the outcome. If the user wishes, they can rotate, scale, or translate this edited region to add more user control to the final model. We call this second part of the pipeline ASMR-3D (Alignment, Selection, Mesh Reconstruction), and it is heavily inspired by CMD and CraftMesh [Li et al., 2025, Jincheng et al., 2026].

As will be discussed in later sections, the choices made at each stage of the pipeline directly

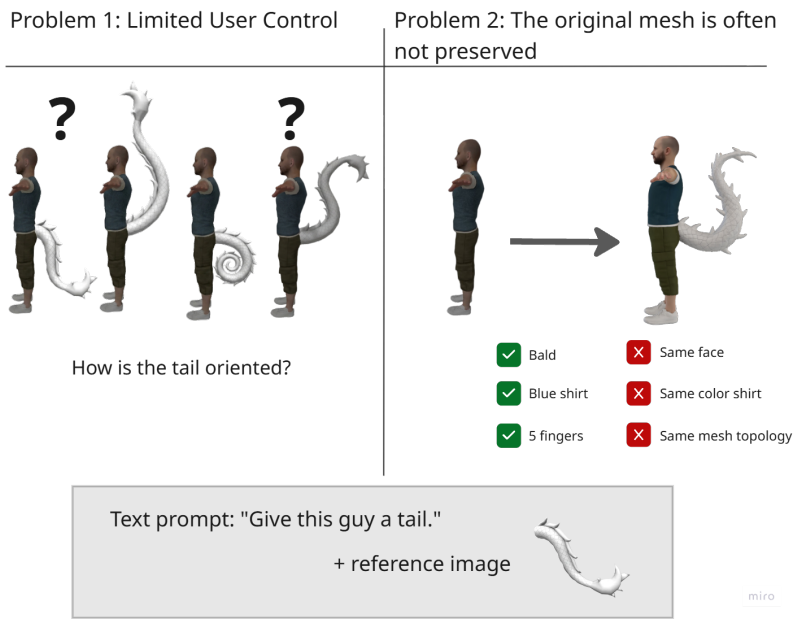

Figure 3.1: Visualization of the biggest challenges in current generative 3D model editing.

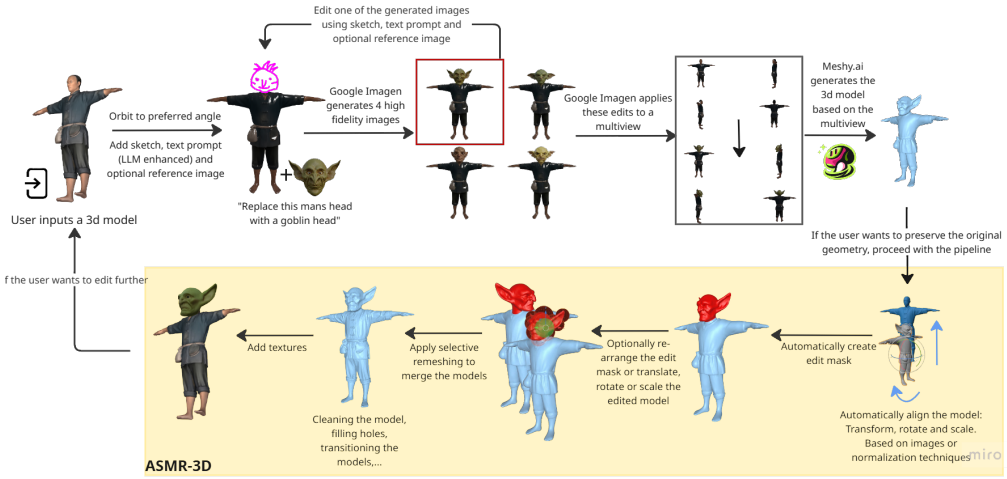

Figure 3.2: The pipeline used in this thesis.

influence those made in subsequent stages. Because of this interdependence, the system is tightly integrated: both input and generated content from earlier steps shape the outcomes of later steps. Therefore, decisions at each stage are carefully considered, as each decision plays an important role in the pipeline's overall effectiveness.

Chapter 4

System design - Part 1: Multimodal Input and 3D Generation

4.1 Input and positioning

The pipeline starts with the user uploading a 3D model. It will work well with any 3D model, but it typically works best with generated 3D models, specifically because they have more evenly distributed vertices (this will be discussed in later topics). This 3D model will now be shown to the user, including the textures. After the model is loaded, the user must select a specific viewpoint to sketch on. This viewpoint is selected by the user rotating the model (with the right mouse button) and zooming in (scrolling). Typically, the user carefully selects this viewpoint to view the most important aspects of the sketch. We chose not to allow the user to translate or rotate the model to make interaction with the sketching later easier. The model will automatically be centered and clipped to fit the window.

4.2 User prompt and sketch

After the user chooses a specific viewpoint, they can sketch on top of it. A snapshot of this viewpoint will be taken, and the user can now move it and zoom in and out. There is no longer a possibility to rotate the model, because this would interfere with the sketching process. Only after the user toggles the sketching mode back off again can they rotate the model. The sketches themselves will be anchored to the image and match the zoom and translations, so they will always stay on the correct part of the snapshot. There are 3 brushes, all with the functionality to undo a stroke or change the brush size. The available brushes are draw, erase, and hide. The 'draw' brush will match the user's mouse clicks on the canvas and have a distinct pink color. We chose this color to make it distinguishable in later steps. This could be used to express the will to 'add' something to the current model; for example, adding a tail to a t-posing avatar would involve drawing a tail with the 'draw' brush. The color, thus, cannot be translated in the sketch, but it could be defined in the prompt later. This is done intentionally to make it easier to use in later steps. The 'hide' brush draws the background color (for example, gray) over the sketch, so that the part of the snapshot is hidden. This could be used by the user to express the will to 'remove' something from the current model, for example, to remove his arms. It is also often used as a baseline to sketch on top of, using the 'draw' brush. Lastly, the 'erase' brush removes elements of the 'draw' or 'hide' brush where the user stroked.

The decision to only let the user sketch from one viewpoint is deliberate. It saves time because

the user does not have to draw the sketch on every viewpoint, and it requires them to think well about which viewpoint to draw on. This will be beneficial in later steps in the pipeline.

After the user finishes sketching, they must add a text prompt expressing the changes they would like to see in the model. This prompt is then refined by an LLM acting as a 3D model and image editing assistant. The LLM is instructed to analyze the user's prompt (along with the original snapshot and a snapshot edited with the sketching tools for context) and perform two tasks. First, it determines whether the prompt is obviously missing a single critical detail that would make the edit ambiguous or impossible, such as "add a hat" without specifying color or style. The LLM is instructed not to be pedantic, to ask at most one question, and to skip asking entirely if the reference image already answers it visually. Second, it always produces an enhanced version of the prompt: a single concise sentence that stays close to the user's original wording, only adding detail when necessary to remove ambiguity and never inventing details the user did not imply. The response is returned as structured JSON, indicating either a clarifying question or the enhanced prompt.

These instructions are fed to the Gemini 3 flash preview model[1]. If the prompt is unclear, the LLM asks a single question to refine it; once answered, or if the original was already clear, the LLM finetunes the user's prompt into its enhanced form. This system was implemented to help users write text prompts, as early tests showed users had difficulty writing effective prompts to express their needs.

On top of this prompt, users can optionally include reference images to provide additional visual context for their desired changes. These images can serve multiple purposes: they can clarify stylistic intent that is difficult to express in words (such as a specific art style, surface texture, or shape language), provide a concrete example of an object to be added to the model, or disambiguate cases where the sketch alone is insufficient. For instance, a user wishing to add a specific type of armor to a character could attach a reference photograph, relieving them of the need to sketch every detail precisely. The reference image is passed alongside the sketch and prompt to the LLM during the prompt refinement step, allowing it to incorporate visual details from the reference when enhancing the prompt. The condition for this, however, is that the user has access to an image that closely represents what they want to add or modify. When such an image is available, the model weighs it heavily during the editing process, making it one of the strongest signals for guiding the output.

Users can choose to use the generated prompt or stick to their initial prompt. Alternatively, if needed, they can edit the chosen prompt.

A few examples of the sketch and prompts, chosen by users in the user study described in chapter6 are shown in Figure 4.1. In these examples, the "hide mask" is the "erased" pixels of the original image that blend in with the background. The maks itself is visually clearer in Figure 4.3.

4.3 Single image generation

A common issue with generative 3D model editing/generation models is that they generate the final model instantly, without providing intermediate feedback to the user. This stage of the pipeline addresses that issue by giving the user intermediate feedback earlier, in the form of images. This will help the user verify/further polish their creative direction very early and allows them to make more fine-grained edits to the given images if desired.

Based on the given user input, the original/unedited snapshot, the snapshot edited with the sketching tools, the selected text prompt, and the reference images, a generative image generation model will now visualize four different versions of the edit the user proposed, so the user can select the one that best fits their vision. These images are generated using Gemini 3 pro image preview. Ideally, the four images would be generated by four distinct image generation

[1] https://ai.google.dev/gemini-api/docs/models/gemini-3-flash-preview

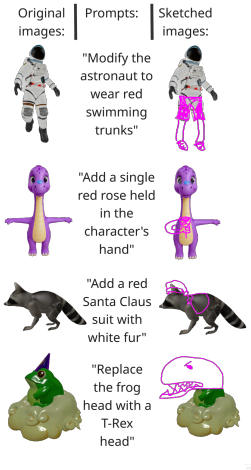

Figure 4.1: Examples of the original images after positioning the model, the sketches made on top of it, and the given prompts.

models to ensure sufficient variation. Having multiple image generation models can increase creative diversity, as different models may capture complementary aspects of the user's intent, improving the likelihood that at least one output aligns well with the desired result. [Hegazy, 2024]

A very important requirement for later steps is that the image generation model exhibits exceptionally high fidelity to the original image and is highly capable of understanding users' desires. High fidelity is very important for later pipeline phases and means that the original image (geometry, lighting, position, ...) is preserved exactly in regions where the proposed edit does not affect the image. This means that if a user wants to add a tail to the model, the pixels representing everything except the zone where the tail should go should remain untouched. Having a model that reliably makes high-quality changes while maintaining high fidelity is currently rare. Therefore, in this thesis, the decision was made to use only the Gemini model, since it can reliably make these high-quality changes while preserving the original image. Since the implemented system is a high-level prototype, it should suffice. In the user study described later, this did not cause any issues. Figure 4.2 illustrates the high fidelity of the image generation model by showing the pixel-wise difference between the two images. As shown, the differences are minimal and are visible only in the edited regions.

We generate four different images concurrently using the same prompt, executed four times in parallel. The prompt instructs the model to act as an expert image editor tasked with modifying the provided image with high fidelity to the original. It must preserve all aspects of the original that are not mentioned in the edit request, maintaining the exact composition, lighting, colors, style, framing, and aspect ratio, as well as the appearance, clothing, and pose of any person or object unless explicitly told otherwise. Only the requested change should be applied, and it must look natural and seamlessly integrated. The output is required to have a pitch-black background.

The prompt also includes detailed sketch guidance. A sketch overlay is provided showing the original image with pink lines drawn on top, which indicate the placement, position, orientation, and approximate shape of the requested edit. The model is told to treat the pink lines as rough directional guidance rather than pixel-perfect instructions, paying attention to where elements should be placed, their angle and orientation, their general shape and flow, their proportions, and their position relative to the underlying subject. The sketch communicates "how" and "where" the edit should occur, while any reference images communicate "what it should look like." The pink lines themselves are not to appear in the final output; they serve only as

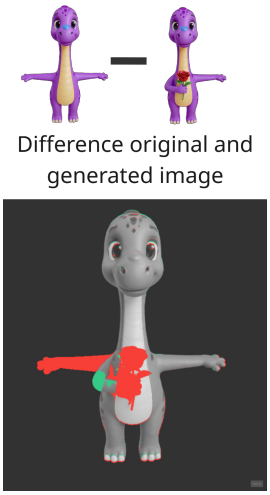

Figure 4.2: The difference between the original and generated image, generated with Gemini 3 flash preview model[2].

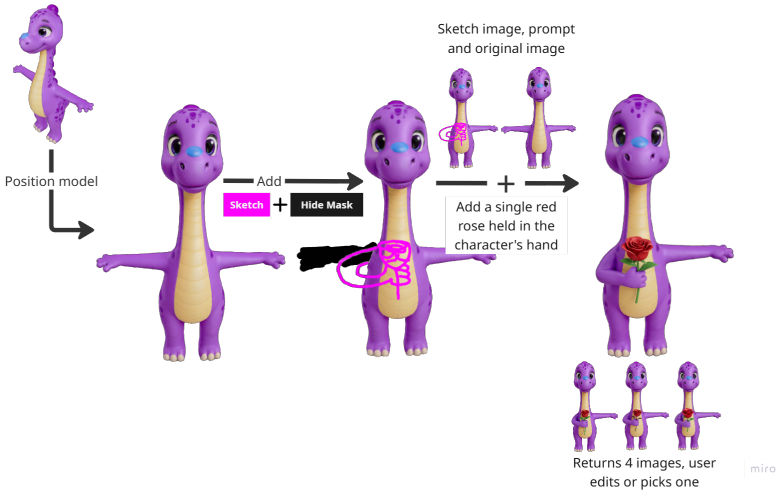

Figure 4.3: Visualization of the first steps of the pipeline, using images directly from the User Study.

placement guides.

After the four images are generated, the user can iteratively edit any one of them using the same sketching and prompting methods described earlier, refining it to better fit their creative vision. Once satisfied, the user selects an image to proceed with in the pipeline. The pipeline up to this point is visualized in Figure 4.3.

4.4 Multiview generation

Because we later use a one-shot 3D generative model that takes a text prompt and a single or multiple images as input, we first generate a multiview representation to provide additional context for the model. This is done by using the same high-fidelity image editing model introduced earlier, which is conditioned on a multiview of the original object. The purpose of including the multiview is to preserve as much of the original geometry as possible and to reduce the likelihood that the model hallucinates unseen parts of the object, particularly the backside [Chen et al., 2026]. In addition to the original multiview, we include the single edited

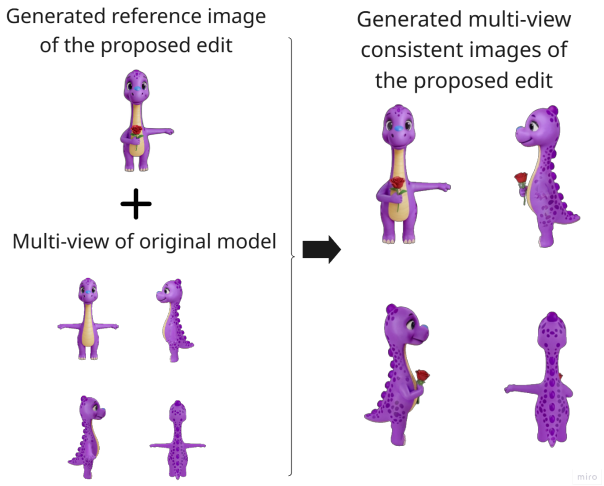

Figure 4.4: Visualization of the generated multiview consistent images of the proposed edit.

image generated in the previous step and the previously selected text prompt. The edit is then consistently applied across all views of the multiview representation of the original model. This is visualized in Figure 4.4

The multiview of the original model is constructed by rendering it from four fixed viewpoints: front, left, right, and back. We restrict the representation to these four views for efficiency and simplicity during image editing, as well as because the 3D generative model we use later accepts only four input views.

One could argue that this multiview should be verified or further adjustable by the user, since the editing process may still introduce inconsistencies or hallucinations in how the added or modified geometry appears across different viewpoints. While this is a valid concern and may lead to more stable results, in our current implementation and user studies, it has not been a significant issue. Moreover, omitting an explicit verification step reduces overall interaction time, thereby improving usability. The resulting quality and fidelity to the original image were deemed sufficient for this implementation.

This step was influenced by CMD [Li et al., 2025], which uses CondMV to apply the edit across multiple views. Ideally, the step in this thesis would be done similarly, with more views having the edit applied. Additionally, these views could be verified by the user to ensure that the generated output fully matches the intended modification. By doing this, it could even make the next step in the pipeline, where the 3D model is generated, obsolete. Instead, we could immediately perform continuous remeshing on the images and normals, as described in CMD. We still chose to implement this step because, in some cases, it is sufficient for users to obtain a model that is not geometrically consistent. It provides a model that preserves the same properties but not the exact geometry, if the user only requires this. This is particularly useful when users do not have the time or need to proceed through the rest of the pipeline.

Originally, we did not include this step in the pipeline, generating the 3D model from a single image and placing the edit mask based on this. But because this method produced inferior results and the 3D model generation hallucinated what was on the “back” side of the model, as described in [Chen et al., 2026], we decided to add this step.

4.5 3D Model generation

Based on the generated multiview and the user's previously selected prompt, a 3D model is created using Meshy.ai[3]. Meshy.ai is an AI platform that generates high-quality 3D models from text prompts and images, using a diffusion-based generative pipeline combined with learned 3D priors to turn text prompts or images into an initial 3D mesh with textures. The model then refines this output through steps like multiview consistency reasoning, mesh reconstruction, and post-processing to produce a clean, exportable 3D asset. While alternative 3D generation tools could be used, Meshy.ai was selected due to my prior experience with the platform and its role as an inspiration for this thesis. It consistently produces high-quality results and is also used later in the pipeline for texturing models.

For this step in the pipeline, inverse rendering approaches could also be considered, such as continuous remeshing as proposed in CMD [Li et al., 2025]. However, these methods typically require more than four input images to achieve reliable results, which in turn would necessitate a more advanced high-fidelity multiview generation and editing stage. Although such an approach could yield strong results, Meshy.ai offers several practical advantages. In particular, it can produce high-quality meshes while also filling in missing or ambiguous regions, and it is straightforward to use, making it well-suited for the proposed implementation. This choice is further supported by user studies that did not indicate this was a limiting factor. Meshy generates high-quality meshes that preserve the overall properties of the multiview input but are not necessarily geometrically consistent with it. In some cases, the inconsistencies can be significant; for example, limb proportions such as arm length may differ from the original input without explicit intent. As shown later in the pipeline, these inconsistencies are mitigated by subsequent processing steps, making sure that only the relevant components of the model, as defined by the proposed edit, are retained.

[3] www.meshy.ai

Chapter 5

System design - Part 2: ASMR-3D

5.1 ASMR-3D Overview

To merge the original model and the generated models, we created a system called ASMR-3D, short for Alignment, Selection, Mesh Reconstruction 3D. This second part of the pipeline acts as a way to apply the edits made in the generated model to the original model without losing the geometry unnecessarily. In the implementation, every step takes place separately and needs to be confirmed by the user before proceeding. This extra confirmation step is implemented as a backup, because these steps are each very important to the final result. We aim for high user control, so every step of the pipeline can be manually finetuned to the users need. At the end of this pipeline, the original model merged with the edits in the generated model will be returned. We aim to make this integration seamless, so it should not be visible that the added part does not belong to the original model, as if the model were created manually.

As described in the literature, some systems do this by segmentation, selecting, replacing, and merging only the section that they need, like PREditor3D [Erkoç et al., 2024]. Other systems like CraftMesh [Jincheng et al., 2026] first generate a reference mesh and extract the edit region mesh out of it based on the generated images. They then merge this mesh with the original mesh using Poisson geometry blending and texture harmonization to get the final model. This thesis proposes a similar implementation to CraftMesh, combined with the continuous remeshing and other methods used in CMD [Li et al., 2025].

5.2 Aligning the generated model with the original model

Before two meshes can be merged, they must first be brought into a common coordinate frame. In our setting, one of the meshes is the original asset that the user wants to keep, and the other is the generated model, which includes the user's proposed edit. As mentioned before, even when both meshes depict "the same object" they are in practice never identical: the generation typically introduces a deliberate change of geometry (the actual edit the user asked for) and on top of that, the image-to-3D generator does not preserve the exact geometry of the input, so even unedited regions of the mesh can differ slightly. Because of this, the centroid, the bounding box, and the principal axes of the two meshes do not match, and a naive alignment based on, for example, center-of-mass plus axis-aligned bounding box rescaling fails almost every time. A visualization of a correct alignment is shown on the right side of Figure 5.1

To deal with this, we propose two complementary alignment methods. The first one is purely geometric and works directly on the 3D points. The second one uses the rendered reference

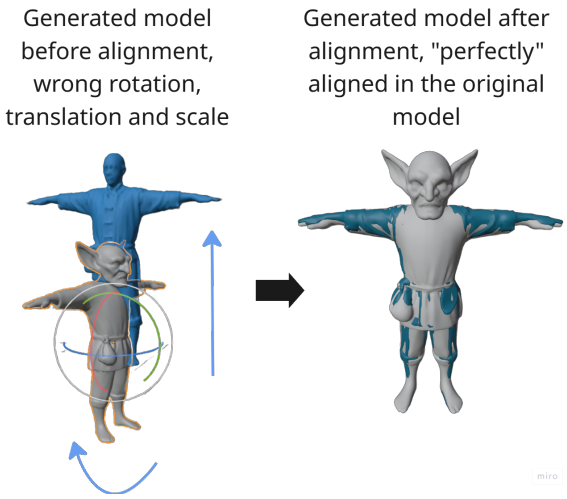

Figure 5.1: Visualization of the original and generated model before and after alignment.

images that are available in our pipeline and is, therefore, image-driven. If, for example, the generated model differs too much from the image and the original model, it may introduce some errors. This is why we expose both of them to the user and let the user pick the one that looks best. In practice, we observed that one of the two methods almost always produces a usable alignment, but that it is not always the same one, hence the choice is left in the loop.

A small but important detail is that we do not feed the meshes to the alignment routines as they come out of the generator. Both alignment methods first run on a *pre-processed* pair: the original mesh is normalized so that all of its dimensions sit in a known interval, and the edited mesh is then rescaled so that its overall size matches what the reference images suggest it should be. The two pre-processing steps are described next, before we get to the alignment methods themselves.

5.2.1 Pre-processing: normalization and image-based rescaling

Normalizing the original mesh. Generated meshes come in arbitrary scales, in arbitrary positions, and sometimes upside-down. To make every later step reproducible, we first push the original mesh through a simple normalization: the model is translated so that its lowest point sits at $y = 0$, it is centered on the x and z axes around its bounding-box center, and it is uniformly scaled so that its largest dimension equals 2. After this step, the model lives inside a known region of space (roughly the slab $y \in [0, 2]$, with x and z centered on the origin) and the "feet" of the object sit on the ground plane. Doing this once at the start of the pipeline removes a lot of arbitrary variation later on and makes everything that follows reproducible, regardless of how unusual the raw mesh coming out of the generator looks. It will also help visualize the models in the frontend.

We assume that the original model is provided in an upright orientation, which is generally the case when the model comes from the same generator as the edited mesh, since these tools tend to output objects in a consistent canonical pose. If this assumption does not hold, the normalization step may introduce misalignment later in the pipeline. In practice, however, this was never reported as an issue during the user studies; the same applies to the rescaling.

Rescaling the edit using the reference images. The edited mesh that comes out of the generator is almost never the right size. The generator picks a bounding box that looks fine on its own, but does not match the size the object should have inside the original scene. To fix this, we use the rendered images that we already have available in the pipeline: a render of the

original object and a multiview render of the edited mesh, both made with the same virtual camera.

The idea is simple. We compare how tall the object is in each image and use that ratio to rescale the 3D mesh. Concretely, we take the front view of the edited multiview render, extract a foreground mask from both that view and the original reference image, and measure the height of each silhouette. The ratio between these two heights, $r = h_{\text{final}}/h_{\text{original}}$, tells us how much taller or shorter the edited object should be compared to the original. We then apply this ratio as a uniform scale factor to the edited mesh.

We use only the height of the silhouette, not its width. Width is much more sensitive to small pose changes and to how the object is projected, while height stays reliable because the camera is roughly horizontal. Scaling uniformly along all three axes also prevents the edit from being stretched or squashed, which would distort the very feature the user asked us to add.

We cannot apply the pixel ratio r directly to the mesh, because the mesh's current world height is arbitrary. Instead, we convert the ratio into a 3D scale factor by computing $s = r \cdot h^{\text{3D}}_{\text{orig}}/h^{\text{3D}}_{\text{edit}}$, where $h^{\text{3D}}_{\text{orig}}$ and $h^{\text{3D}}_{\text{edit}}$ are the heights of the original and edited bounding boxes in 3D. We then apply s uniformly to the x, y, and z coordinates of every vertex.

Note that we do not need to know the exact camera parameters (focal length, sensor size, distance) for this to work. Because both images are rendered from the same canonical front view at the same resolution, the relationship between world height and pixel height is the same for both, and any common projection factor cancels out in the ratio.

This step worked well in our experiments, but it is not flawless. It assumes that the silhouette of the edited mesh in the front view faithfully reflects the proportions of the reference image. If the generator produces a mesh with proportions that differ significantly from the reference, the recovered scale factor will be off. We observed small deviations of this kind in practice, but they did not affect the final result noticeably.

After this step, the edited mesh has approximately the right size, but it is still in the wrong place and orientation. Fixing that is the job of the two alignment methods described below.

5.2.2 Method 1: feature-based registration with ICP refinement

The first method does not look at the images at all. It works purely on the geometry of the two pre-processed meshes and uses a classical 3D registration pipeline made of three stages: a global initialization with RANSAC on FPFH features, a similarity refinement with the Umeyama closed-form solution, and a final fine alignment with point-to-plane ICP. Each of the three stages exists for a specific reason, which we explain below.

Sampling and downsampling. Both meshes are first converted into point clouds by uniformly sampling 40,000 points from their surfaces. Working with points rather than with the raw triangles allows us to use the standard registration tools and makes the rest of the pipeline insensitive to the (often very different) tessellations that the generator produces. The clouds are then downsampled with a voxel grid whose size is set to roughly 1/30 of the scene extent. The downsampled clouds are what we use for the expensive parts (feature computation and RANSAC); the dense clouds are only used at the very end for the ICP refinement.

Why we cannot just run ICP from the start. The Iterative Closest Point algorithm, originally proposed by Besl and McKay [Besl and McKay, 1992] and by Chen and Medioni [Chen and Medioni, 1991], alternates between two simple steps: for every point of the source cloud, it finds the closest point in the target cloud, and then it computes the rigid transform that minimizes the mean squared distance over those correspondences. The algorithm is repeated until the error stops decreasing. ICP is fast, easy to implement, and very accurate, but it has one well-known weakness: it converges only to the local minimum that is closest to its starting

position, so it requires the two clouds to already overlap reasonably well before it is launched. If we apply it directly to our edited and original meshes, which can sit far apart and at arbitrary orientations, it confidently locks onto the wrong solution.

Stage 1 - Global initialization with FPFH + RANSAC. To give ICP a sensible starting point, we first run a global registration based on local feature descriptors. For each point in the downsampled clouds, we compute a Fast Point Feature Histogram, or FPFH, introduced by Rusu et al. [Rusu et al., 2009]. An FPFH is a 33-dimensional histogram that summarises how the surface around a point is curved relative to its neighbors; importantly, it is rotation- and translation-invariant, which is exactly what we need because at this stage the two clouds are in completely different poses. Points that have a similar FPFH descriptor are likely to correspond to the same physical location on the object, even if they live in unrelated frames.

We then use the RANSAC algorithm of Fischler and Bolles [Fischler and Bolles, 1981] to turn these candidate matches into a single rigid transform. RANSAC repeatedly picks four matches at random, fits the rigid transform that they imply, and counts how many of the remaining matches agree with it within a small tolerance. The transform with the largest agreement is kept. The advantage of this scheme is that it is extremely robust to wrong matches: even if most of the FPFH correspondences are bad, as long as a few of them are correct, RANSAC will find them. This gives us a first "rough" alignment that is usually within a few degrees and a few millimeters of the right answer, but never pixel-perfect.

Stage 2 - Similarity refinement with Umeyama. The RANSAC output is a rigid transform (rotation + translation), but the edited mesh may still need a small uniform scale correction on top of the one we already applied during the image-based rescaling. To recover this scale together with a refined rotation and translation, we apply the closed-form solution of Umeyama [Umeyama, 1991]. Given a set of point correspondences. Umeyama's formula returns the similarity transform (scale, rotation, translation) that minimizes the least-squares error in a single SVD. Compared to the older Arun-Horn formulation, it has the practical advantage of never returning a reflection by mistake when the data is noisy, which matters in our case because some of our generated meshes are near-symmetric and reflections are real failure modes.

Stage 3 - Fine alignment with point-to-plane ICP. The output of stages 1 and 2 is good enough to put the two clouds in overlap, which is exactly the regime where ICP shines. We finish the alignment with a point-to-plane variant of ICP. Where standard ICP minimizes the squared distance between matched points, the point-to-plane variant minimizes the squared distance from each source point to the tangent plane of its target, which corresponds to a local linear approximation of the target surface [Chen and Medioni, 1991]. This variant converges substantially faster and is more accurate on smooth surfaces, which is exactly the kind of geometry we are dealing with. The final transform is the composition of the RANSAC, Umeyama and ICP transforms.

Why this method is a good fit. The whole pipeline is implemented on top of Open3D [Zhou et al., 2018]. What makes it well-suited to our problem is that none of the three stages assumes that the two meshes are identical: RANSAC tolerates a large fraction of wrong correspondences, Umeyama provides a closed-form similarity that absorbs small global scale errors, and point-to-plane ICP only needs the two surfaces to overlap, not to be the same surface.

This robustness to partially differing geometry has a straightforward geometric explanation. ICP minimizes the mean squared distance over all matched point pairs. When the majority of the surface is shared between the two meshes, like for example when one model is a t-posing human figure and the other is the same figure with an added tail, the large number of correctly matched inlier pairs dominates the objective. The cluster of unmatched points belonging to the unique feature finds no consistent counterpart and is instead matched to the nearest surface

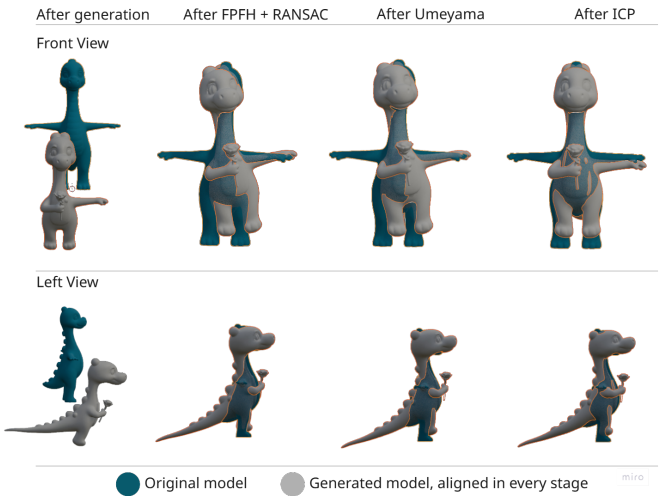

Figure 5.2: Visualization of the different steps of the alignment method 1.

region of the other mesh. Formally, if n_{in} inlier pairs each contribute a residual ε_{in} and n_{out} outlier pairs each contribute ε_{out}, the total objective is:

$$\mathcal{L} = n_{\text{in}}, \varepsilon_{\text{in}}^2 + n_{\text{out}}, \varepsilon_{\text{out}}^2$$

As long as $n_{\text{in}} \gg n_{\text{out}}$, minimizing $\mathcal{L}$ drives the transform towards the inlier consensus, and the outlier contribution is suppressed. This behavior is well documented in the partial-overlap registration literature: Chetverikov et al. [Chetverikov et al., 2002] show that ICP remains reliable well below 50% overlap.

So as long as a reasonable portion of the geometry is shared between the original and the generated model, this method finds the right alignment even when the edit adds, removes, or deforms part of the mesh. If the proposed edit is so large that the largest part of the geometry is not shared between the original and generated model, like for example when the original model is a head and the proposed edit is to add a body, method 2 should be able to handle the alignment correctly. We encountered this exact issue during testing, which is why we introduced a second alignment method that is able to handle such cases correctly.

The different alignment steps in this method are visualized in Figure 5.2. These images are taken directly from a model created and aligned in the user studies.

5.2.3 Method 2: Bounding the models based on multiview renders

The second method takes a completely different angle from the geometric registration of Method 1: it does not run any kind of point-cloud or surface registration at all. Instead, it relies on the fact that we already have a multiview render of the original object and a generated multiview render of the requested edit, and that those two images already encode the correct spatial relationship between the two objects in pixel space. The idea is to read off that relationship from the images and convert it directly into a rigid transform that is applied to the edited mesh. The transform consists of a Y-axis rotation chosen from $\{0°, 90°, 180°, 270°\}$ and a 3D translation; no continuous rotation and no extra scale are estimated.

The reason for this design is that the geometric pipeline of Method 1 spends a lot of effort recovering rotation and scale that we can already trust. The original mesh has been normalized, the edit has been rescaled from the reference images, and our experience with the image-to-3D generator showed that it always produces models in a canonically upright pose, with the in-plane orientation matching the reference image. The only genuinely unknown degrees of freedom that remain are (i) which multiple of 90° around the vertical axis the generator settled

on, and (ii) the 3D translation that places the edit relative to the original. These are exactly what Method 2 estimates.

The multiview input. Each render is a 2×2 grid containing four canonical orthographic views of the object: front (top-left), right (top-right), left (bottom-left), and back (bottom-right). The first thing the method does is split each render into these four sub-images. The front and back views give information about the x and y axes; the left and right views give information about the z and y axes. With four views, any axis-aligned offset is visible in at least two of them, which lets the method cross-check estimates and reject outliers via a per-axis median. In the following steps, it is worth mentioning that we take into account that the resolution of the original render can be different from the resolution of the generated render. Therefore, we reason in a resolution-agnostic way.

Stage 1 – Silhouette extraction. For each of the eight sub-views, we extract a binary silhouette of the object. Instead of thresholding against a hard-coded background color, we estimate the background directly from the image. We sample four thin border strips (top, bottom, left, right) and pick the two with the lowest color variance as the cleanest background samples. This avoids picking up the foreground when the object touches one image edge (for example, a tall character whose head almost leaves the front view). Their median color is used as the background reference, and a pixel is classified as foreground if its color distance exceeds $\mu_d + 2\sigma_d + 15$. This adaptive scheme handles the slightly varying, often gradient-like backgrounds that image generators tend to produce [Frank et al., 2020]. The resulting mask is then cleaned with a single close-and-open morphological pass, and the tightest axis-aligned bounding box is computed.

An alternative would be to use a pre-trained foreground extraction model, which generally works well and removes the need for the hyperparameters used above. We chose the lighter, threshold-based approach because it was sufficient for our renders, but for noisier or more diverse inputs, a learned segmentation model would likely be a more robust choice.

Stage 2 – A consistent pixel-to-world ruler. Both later stages need to convert pixels into world units, and they need to do so consistently across two renders that may have different resolutions. We built this ruler in two steps.

First, we calibrate against the original render. The original mesh has known world extents, and its silhouette in each view has a known pixel size. Their ratio gives a pixels-per-world estimate; we collect one such estimate per view per axis and take the median as the global s_{orig}. The median guards against a single view with a clipped silhouette, pulling the estimate.

Second, we transfer this scale to the edit render. Both renders share the same camera, so one unit of world distance projects to the same *fraction* of the image in each. Pixels-per-world, therefore, scales linearly with image height:

$$s_{\text{edit}} = s_{\text{orig}} \cdot \frac{H_{\text{edit}}}{H_{\text{orig}}}$$

This single relation is enough to make every pixel measurement on the edit render comparable to the original.

Stage 3 – Detecting the Y-axis rotation. The generator usually preserves the reference orientation, but we occasionally observed 90°, 180°, or 270° flips around the vertical axis – typically when the object's front, back, or sides look similar. We therefore search over $0°, 90°, 180°, 270°$.

For each candidate rotation, we project the rotated edit vertices into each view, rasterize them as a binary silhouette via convex-hull fill, recenter that silhouette on the measured one, and compute the IoU. Intersection over Union (IoU) is a similarity measure between two binary

masks defined as the area of their intersection divided by the area of their union, used here in Stage 3 to score how well each candidate $0°, 90°, 180°, 270°$ rotation of the edit mesh, matches the measured silhouette in each view, with the per-rotation score averaged across the four views; the metric is standard in object detection and segmentation, going back to the PASCAL VOC benchmark [Rezatofighi et al., 2019].

The selection rule is biased toward 0°: a non-zero rotation is chosen only if it beats 0° by more than a margin (default 0.15 IoU) *and* achieves an absolute IoU above a quality floor (default 0.45). This is implemented because, in our experiments, we noticed that most of our models did not need rotation at all and were already generated in the correct rotation. If we do not add this bias, we notice some incorrect predictions, especially in very "symmetric" models.

Stage 4 – Translation via proportional bbox mapping. With rotation fixed, we estimate the translation directly from bounding boxes. The idea: the original render is a calibrated ruler, so reading the edit silhouette against that ruler tells us where the edit's outermost points should sit in world space.

Consider one view and one axis. We need four pixel measurements and two world measurements. From the original render, we read the pixel positions of the silhouette's two extremes along this axis, $O_{p,\min}$ and $O_{p,\max}$ (for example, the leftmost and rightmost pixel of the silhouette in the front view). From the edit render, we read the same two extremes, then rescale them into the original render's pixel frame to account for any resolution difference; we denote these rescaled values $E'_{p,\min}$ and $E'_{p,\max}$ (the prime simply marks "expressed in original-render pixels"). Finally, from the original mesh itself, we know the true world-space extremes along this axis, $O_{w,\min}$ and $O_{w,\max}$.

Because the original silhouette and the original mesh describe the same object, their extremes correspond, which calibrates a pixels-per-world ratio for this axis:

$$s_{\text{axis}} = \frac{O_{p,\max} - O_{p,\min}}{O_{w,\max} - O_{w,\min}}.$$

We recompute this per axis per view rather than reusing the global s_{orig}, which absorbs small renderer anisotropies and ties the recovery directly to the original mesh's true extrema.

We now use the same ruler to convert the edited silhouette's pixel extremes into world coordinates. Writing $\bullet$ as a placeholder for either min or max (the same formula applies to both ends), the world position implied by the edit silhouette is

$$E_{w,\bullet} = O_{w,\bullet} + \frac{E'_{p,\bullet} - O_{p,\bullet}}{\sigma \cdot s_{\text{axis}}},$$

where $\sigma \in +1, -1$ flips the sign for views whose world axis runs opposite to the image's left-to-right direction (e.g. back vs front, left vs right). For the vertical axis, the same idea applies with one twist: pixel-y grows downward while world-y grows upward, so the top of the bbox maps to $y_{\max}$ and the bottom to $y_{\min}$.

The edit mesh's vertices currently span $[M_{\min}, M_{\max}]$ along this axis, so each extremum gives one candidate translation – how far we'd need to shift the mesh so that the end lands where the silhouette says it should:

$$d_{\min} = E_{w,\min} - M_{\min}, \qquad d_{\max} = E_{w,\max} - M_{\max}.$$

We take their average (a centered fit that splits any residual scale mismatch between the two ends) and then take the median across views to obtain Δx, Δy, Δz.

Why centered fits, not extremum anchoring? The two candidates $d_{\min}$ and $d_{\max}$ rarely agree exactly: the edit mesh and the edit silhouette come from the same generative pipeline, but are not pixel-perfect reconstructions of each other, and silhouette extraction itself introduces a

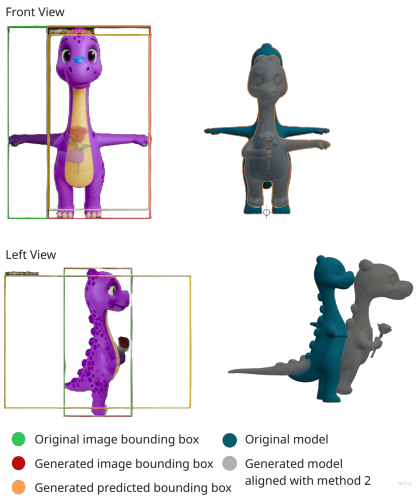

Figure 5.3: Visualization of the bounding boxes in alignment method 2, we can see that the generated model extended the tail by a lot. Therefore, the Front view matches correctly, but the Left view does not.

few pixels of slop at each bbox edge. Anchoring to either extremum forces all of that residual mismatch onto the opposite end. Averaging the two candidates splits the residual evenly, halving the worst-case error at either extreme.

Why this method is a good fit. What makes this method work so well in practice is that, given the pre-processing of the models (normalizing and resizing), almost everything except a discrete rotation and a 3D translation has already been resolved. The renders themselves provide a very high-fidelity, almost ground-truth signal for these two unknowns, because the camera framing is consistent across both renders, and the silhouettes can be extracted reliably. The method uses the mesh's own world extrema wherever pixel measurements would be misleading (the resolution ratio is folded into the ruler exactly once, not into individual estimates), it cross-checks across four views via per-axis medians, and it biases rotation selection toward the prior of an unrotated generator output. As long as the generated mesh genuinely depicts the requested edit, even if the edit changed the object's shape substantially, the four views give a tight estimate that, in our experience, no amount of ICP iteration on the surfaces themselves could improve, because the surfaces have genuinely changed.

Ideally, we would implement a similar method to ICP afterward, to fine-tune the alignment exactly. But we noticed that in the case where the largest part of the geometry is not shared between the original and generated model, it would pull the vertices in the wrong direction. Also, this method is purely reliant on how well the images have been generated; if something went wrong in this step, the whole method would prove obsolete.

Both methods work well on a wide range of inputs, but they are sensitive in different ways to the same underlying problem: the fidelity of the generated mesh. If the 3D generator produces a mesh whose silhouettes do not match the reference images, because the generator hallucinated extra detail, missed a limb, or simply got the proportions wrong, both methods degrade. Method 2 is the more fragile of the two in that case, because the bounding boxes assume that the mesh, once placed at the correct pose, will actually project onto the reference image. Method 1 is geometric, and so it is more forgiving on this point. This is illustrated in Figure 5.3, where the generated mesh of the dinosaur does not accurately match the scale of the reference images. While the front view appears well-aligned, as the mesh scaled correctly along those axes, the side view reveals a discrepancy: the model generated a disproportionately elongated tail relative to the reference images. Notably, alignment method 1 successfully corrected this misalignment despite the tail's excess length. The effectiveness of both models will be evaluated

in the experimental design chapter6.

In practice, this is the main reason we expose both methods to the user. They have alternating success: when the generated mesh is a faithful 3D version of the reference images and differs a lot from the original, method 2 tends to win because it concentrates the alignment effort where the user wants it. When the generated mesh deviates more from the images, method 1 tends to win because it is happy to align "whatever overlaps". We always present both results and let the user pick.

The remaining failure cases are the ones where the generated mesh simply does not look like the images. There is little that an alignment routine can do in that situation, since the very signal it relies on is wrong. Two natural directions to fix this would be to (i) plug in a more reliable image-to-3D generator, which would shrink this failure mode at the source, and (ii) combine the methods with a semantic segmentation step on the renders, so that the alignment could be performed part-by-part on regions that are known to be shared between the two meshes. Similar to what they do in [Erkoç et al., 2024], but as explained in the next chapter, this could lead to inaccuracies.

5.3 Selecting and Editing the Edit Region

Once the original and edited meshes are aligned, the next step is to determine where on the mesh the user's edit should be applied. Without such a localization step, there is no principled way to preserve the geometry that should remain unchanged: any blending or fusion between the two meshes would inevitably leak into untouched regions. As mentioned before, PrEditor3D approaches this problem with 3D segmentation, propagating a 2D selection into a volumetric segment that defines the editable area [Erkoç et al., 2024]. While intuitive, this strategy is tightly coupled to the quality of the underlying segmentation: inaccurate or misaligned segments propagate errors into the editing stage, predefined segment boundaries restrict edits that do not align with semantic parts, and the method inherits the classical trade-off between over- and under-segmentation. Segmentation is therefore a useful but imperfect prior for deciding where edits should occur.

We instead take inspiration from CraftMesh [Jincheng et al., 2026], which leverages multiview renderings as a strong cue for what a 3D edit should look like. The multiview signal directly reflects user intent: it shows, from several viewpoints, exactly which parts of the object have changed. However, in our setting, only four views are available (front, left, right, back), and the AI-generated edited mesh can deviate noticeably from the rendered images due to reconstruction artifacts. Relying on a single source of evidence is therefore unreliable, so we developed a tailored method that fuses several complementary signals into a robust selection. The method takes as input the original and edited meshes, a single rendered view of each (the original being a render and the edited being an AI-generated image of the desired result), the camera parameters of that view, a 2D sketch drawn on top of the rendered view that highlights the intended edit and four multiview images of the edited and original model (again the original multiviews being renders and the edited being generated). From these inputs, it produces a set of vertices on the edited mesh that constitute the edit region.

5.3.1 Method Overview

The selection method is built around a voting system. Each piece of evidence we extract is independently noisy, but combining them yields a reliable selection. We compute three independent cues and project all of them onto the vertices of the edited mesh, then select vertices that gather enough cumulative evidence. The cues are: (i) a geometric difference between the original and edited meshes, (ii) a per-view 2D image-difference signal projected back onto vertices via four-view raycasting, and (iii) a 2D image-difference mask derived from the single rendered view, expanded around the user's sketch. Each cue is described below, followed by the voting and post-processing stages.

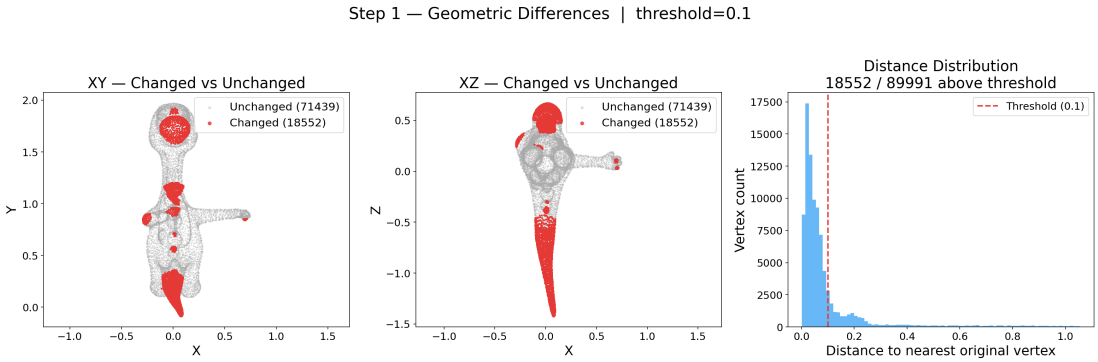

Figure 5.4: Geometric-difference cue. Each vertex of the edited mesh is matched to its nearest vertex on the original mesh, and vertices whose distance exceeds the threshold are marked as changed. The left and middle plots show the changed (red) versus unchanged (grey) vertices in the XY and XZ projections of the 3D mesh. The right plot shows the distribution of nearest-neighbor distances, with the dashed line indicating the threshold above which vertices are selected.

5.3.2 Cue A - Geometric Difference Between Meshes.

The most direct signal is the geometric distance between the two meshes. For every vertex of the edited mesh, we query a k-d tree built on the original mesh's vertices and obtain the distance to its nearest neighbor. Vertices whose nearest-neighbor distance exceeds a threshold are flagged as geometrically different. This cue is fast and parameter-light, but it is also the noisiest: because the edited mesh is produced by an image-to-3D generator rather than by a localized deformation of the original, global reconstruction differences appear all over the surface, even in areas the user did not intend to modify. A normalization pass shifts and scales the meshes into a common frame, but small drifts remain. Another issue is that some edits might include vertices that are close to the original mesh, for example, when adding a tail to a person, the vertices at the base of the tail might not be included in this cue. For these reasons, the geometric cue is treated as a weak signal in the voting stage. An example of this geometric difference is visualized in Figure 5.4.

5.3.3 Cue B - Sketch Mask and Image-Difference Expansion.

The user expresses the intended edit by drawing on top of the rendered original view, in a fixed magenta color. We extract the sketch by color-thresholding, then dilate, close, and fill the resulting mask before computing its convex hull. The convex hull is important: users typically draw outlines or scribbles rather than fully filled regions, so taking the hull turns a sparse stroke into a coherent 2D area that conservatively covers the user's intent. Ideally, the sketch would be saved as a separate image instead of being extracted afterward through color thresholding. The current approach could, in theory, cause problems if part of the underlying figure happens to share the magenta color, which would then be incorrectly picked up as part of the sketch. We chose this simpler design because keeping the sketch in the same image is more straightforward to implement, and it has the additional benefit that the combined image can be passed directly to the prompt by telling the model that the magenta strokes represent the user's intended edit. This is admittedly a minor limitation that could be improved with minimal effort in future work, but it did not cause noticeable issues in our experiments.

The sketch alone, however, only captures the rough region the user pointed at; the actual edit produced by the AI may extend slightly beyond the strokes or land slightly to another position. To compensate, we additionally compute a pixel-difference mask between the original render and the AI-generated edited image. Both images are passed through the same background-removal step mentioned before so that differences in the background do not pollute the foreground signal. We then mark as different any pixel where the color distance exceeds a threshold (with

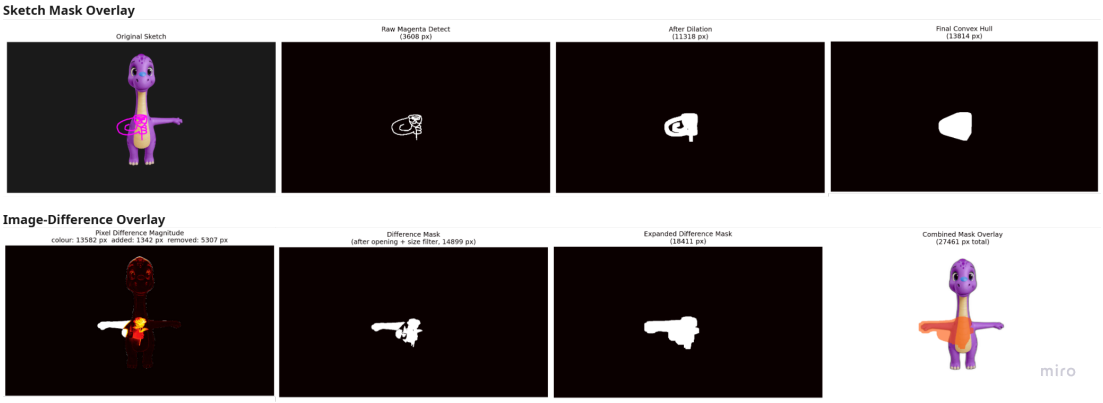

Figure 5.5: Graphs showing the selected vertices using Sketch- and Image-Difference mask.

both images foreground), as well as any pixel that is foreground in only one of the two images, which captures geometry that has been added or removed. Connected components below a minimum size are discarded as noise, and the mask is dilated to absorb thin edge effects. The image-difference mask and the convex sketch hull are merged by taking their union, restricted to the connected components of the difference mask that touch the sketch hull. This removes spurious differences that lie far from where the user actually drew.

The combined 2D mask must finally be projected onto the vertices of the edited mesh in order to contribute to the voting. We use the camera parameters provided alongside the rendered view, namely its view matrix V, projection matrix P, and viewport, which together describe exactly how the original render was produced. For every vertex $\mathbf{v}$ of the edited mesh, we compute its pixel position with the standard projection.

$$(x, y); =; \pi\big(P, V, \mathbf{v}\big),$$

where π denotes perspective division followed by the viewport mapping. Because of the alignment computed in an earlier step, the projected silhouette of the original mesh already matches the foreground content of the rendered image, so the same projection lands the edited-mesh vertices in the correct pixel frame. A vertex receives this cue's vote if it lies in front of the camera, falls within the image bounds, and the pixel (x, y) lies inside the combined 2D mask.

An example of this, gathered from the user studies, is shown in Figure 5.5.

5.3.4 Cue C - Multiview Difference and Per-View Voting.

The four multiview images of the edit are the strongest signal we have, since they observe the change from independent viewpoints and are direct outputs of the editing model. The output multiview (edited) and the input multiview (original) are split into their four quadrants, each quadrant is background-removed, and a per-view difference mask is computed in the same way as in Cue B (color difference for pixels that are foreground in both images, plus added and removed foreground regions). Connected components below a minimum size are discarded, and the mask is slightly dilated.

To project these per-view masks onto the vertices of the edited mesh, we synthesize four virtual cameras, one per canonical view, with parameters chosen to match the layout used by the multiview generator. For each view, we project all original-mesh vertices through the corresponding virtual camera, fit a similarity transform that maps the projected silhouette onto the detected content bounding box of that quadrant, and apply the same transform to the projected edited-mesh vertices. A vertex receives a vote from a given view if its aligned 2D projection falls inside that view's difference mask. The number of views in which a vertex was

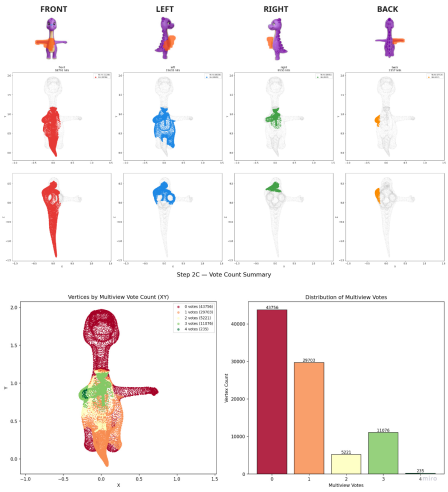

Figure 5.6: Figure of graphs showing the selected vertices using Multiview Difference and voting. The topmost graphs show the front, left, right, and back masks of the views being projected onto the model. The bottom left graph shows the vertices by vote count, with the distribution being visible in the bottom right graph.

hit becomes its multiview vote count, ranging from 0 (no view sees a change there) to 4 (all views agree). This vote count can be changed by the user, but by default, we select 2. The reason is the following. If we select 1, we are sure that the edit is encapsulated in the mask, but there will be too many vertices selected, because of the dilation of the mask. If we select 3, although generally more accurate in its vertex selection, it is possible that views that are only visible from 2 views are not selected, for example, adding a hole in a person's chest. Therefore, we select 2. We are aware that even with 2 votes, it is possible that an edit only visible in one view will not be selected, but in our experiments, this does not occur, plus the user could manually change it later on. Another step explained in later steps will also act as a fallback in similar situations. 2 votes seemed to be a good middle ground, selecting the vertices we wanted in most cases.

This step is shown extensively in Figure 5.6, which shows a visualization gathered from the user study, similar to Figure 5.4 and Figure 5.5. We can see the multiview projection and the number of multiview votes the mesh gets.

5.3.5 Voting and Threshold

The three cues are combined into a single vote count per vertex. The geometric cue contributes +1, the image-difference cue contributes +1, and the multiview cue contributes +2 if the vertex was hit in at least 2 of the four views (otherwise it contributes nothing through this channel). A vertex is selected as a candidate if its total vote count is at least 3.

We also experimented with using the raw view count directly (so a vertex hit in k views contributes +k), but found that this produced noisier results in practice: a single view hit is often a false positive, and giving it any weight tended to pull in unrelated vertices. The hard threshold, therefore, acts as a simple noise filter and gives consistently cleaner selections.

The unequal weighting reflects the relative reliability of the three signals. The geometric cue is the noisiest because reconstruction differences appear globally, so it deserves only a single vote. The single-view image-difference cue is more directly tied to user intent, but is observed from only one angle and depends on a 2D-to-3D projection that can drift. The multiview cue is the most informative: it observes the change from four independent viewpoints, it directly reflects what the editing model produced, and a vertex that survives projection into three or more views is very likely a real edit and not a projection artifact. Granting it two votes ensures

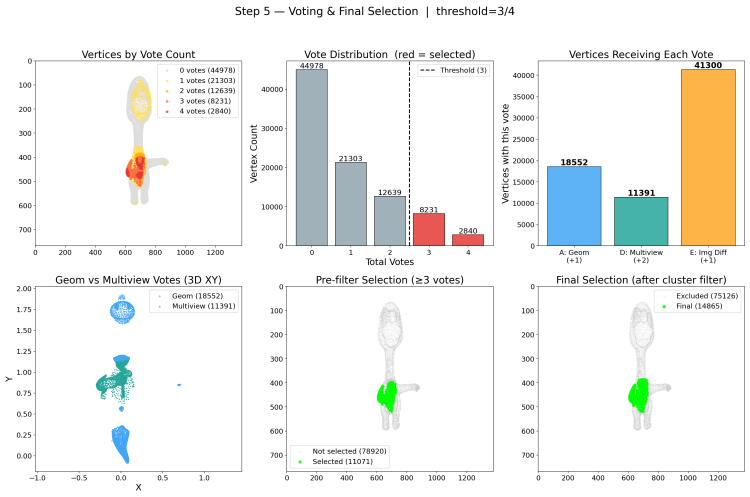

Figure 5.7: Voting stage with a threshold of 3. The top-left plot shows every vertex coloured by its total vote count in image space. The top-middle plot shows the distribution of vertices over vote counts, with the selected counts in red. The top-right plot shows how many vertices each individual cue contributes (A: geometric, D: multiview, E: image-difference). The bottom-left plot shows the geometric and multiview votes side by side in 3D. The bottom-middle plot shows the candidate selection before filtering (all vertices reaching the threshold), and the bottom-right plot shows the final selection after the connectivity-based filtering cluster filter.

that vertices strongly agreed upon by the multiview cue are selected even when one of the weaker cues is silent, while still requiring corroboration from at least one other source. The threshold of 3 has the practical consequence that no single cue can select a vertex on its own: every selected vertex is supported by at least one independent signal, which is the property we want from a voting system.

If no vertex meets the requested threshold, it subtracts this threshold by 1 and tries again until one or more vertices meet the threshold or the threshold is 0 (selecting all vertices). This acts as a fallback if the requirement was too strict. It also adds a solution to the issue mentioned in Cue C, where if an edit is only visible in one viewpoint, the edit will not be included in Cue C and thus will not be included in the final vote (because it contributes 2 points and u will need 3/4 votes). But because the change must be visible in Cue C, which is the view where the user sketched, it will contribute a point and will be included in the final mask.

The specific weights and threshold values were chosen empirically, through an informal ablation-like exploration in which a range of weighting schemes and thresholds were tested across various edits, combined with the qualitative reasoning outlined above. A more formal, quantitative ablation study would be a valuable direction for future work to further validate these choices.

Similarly to before, Figure 5.7 illustrates this voting step with a threshold of 3 votes.

5.3.6 Post-Processing

The voted candidates form a coherent region but tend to have ragged boundaries and isolated specks, both of which would cause artifacts during downstream blending. We address these with two passes.

First, we expand the candidate set along mesh connectivity. Using the face list of the edited mesh, we build a vertex adjacency graph and grow the selection by a fixed number of one-ring layers. This closes small gaps in the selection and ensures that the boundary of the region runs smoothly along the surface rather than zig-zagging between selected and unselected vertices.

Second, we filter by spatial connectivity. We build a k-d tree on the selected vertices and treat two vertices as connected if their Euclidean distance is below five times the median edge length of the edited mesh. We extract the connected components of the resulting graph using a sparse representation, then keep every component whose size is at least 15% of the largest component's size. We do not simply keep the largest component because edits can legitimately consist of several spatially separated parts: a user adding wings, for example, produces two clusters of similar size, both of which are valid. At the same time, we do not keep all components, because the projection-based cues occasionally raycast onto unrelated parts of the mesh when the multiview images do not translate cleanly to the generated geometry. The relative threshold strikes a balance: it preserves multi-part edits whose pieces are of comparable scale, while discarding small spurious clusters that are dominated by a much larger main edit.

This relative threshold of 15% can, in some cases, be fragile, particularly when the edit is small compared to the overall mesh. In those situations, a small but legitimate edit may be discarded because it falls below the percentage of the largest component. However, without this filter, we observed that too many spurious clusters were retained, leading to selections that included clearly unrelated parts of the mesh. In practice, users facing this situation can address it by running the pipeline iteratively: applying the large edits first and then performing the smaller edits in a separate run where the dominant component is no longer present.

5.3.7 Hide Mask for Removal Edits

The pipeline so far identifies vertices on the *edited* mesh that correspond to additions or modifications. Some user edits, however, consist of removing geometry from the original mesh, in which case there needs to be a way to select vertices on the original mesh. To support such edits, the user can paint a separate *hide mask* on the rendered original view, marking the geometry that should be removed. We process this mask with the same color-threshold and dilation pipeline as the sketch and project it back onto the original mesh by raycasting through the calibrated camera. Vertices of the original mesh whose 2D projection falls inside the hide mask are flagged for removal.

Unlike the addition/modification case, this step does *not* use the voting mechanism and is only applied to the first (front) view. We deliberately kept this simpler because removal edits are, in practice, far more straightforward than additions: the user almost always wants to remove the full geometry that is clearly visible from a single chosen viewpoint, so the extra robustness of multiview projection and cue voting is not needed. A more principled approach would also leverage the image-difference signal on the original mesh, or extend the voting scheme to removals, but in our experiments, the simple raycast-based hide mask was sufficient and produced reliable removal edits in our prototype.

This is particularly useful in edits where certain vertices from the original model are no longer needed. A clear example is the purple dinosaur shown in Figures 4.14.24.34.45.25.3, where the user wants the dinosaur to hold a flower. Since the right arm is originally in a T-pose, its distal vertices become irrelevant to the edit, and as noted earlier, the edit mask will not cover them. It is therefore good practice for the user to hide the arm before drawing, which is what occurs in our experiments. However, since this cannot always be guaranteed, the approach described in the previous section provides a more robust solution.

5.3.8 From Vertices to Editable Primitives

The output of the selection stage is two sets of vertices: the edit region on the edited mesh and, optionally, the hide region on the original mesh. For the downstream editing interface, we convert each of these sets into a small collection of spheres that approximately cover the selected vertices.

The choice of spheres is purely a prototyping convenience: a sphere has only a center and a radius, so it can be displayed and manipulated in the frontend without exposing rotation handles,

which keeps the user interface simple. Any other primitive (boxes, capsules, ellipsoids) could replace spheres without changing the rest of the pipeline, since the role of these primitives is solely to give the user editable handles over the region produced by the selection method.

A sphere is allowed to engulf the vertices of the edit region, but it must not reach into the parts of the original mesh that the user did *not* mark for editing, otherwise downstream blending would damage untouched geometry. We therefore label every original-mesh vertex as either *safe* (inside of the edit region) or *unsafe* (outside the edit region). The unsafe vertices act as walls that no sphere is allowed to cross. For each candidate sphere center $\mathbf{c}$, we cap its radius at $0.9 \cdot d(\mathbf{c})$, where $d(\mathbf{c})$ is the distance from $\mathbf{c}$ to the nearest unsafe vertex, leaving a small safety margin.

Inside that cap, the spheres are fitted in three stages. First, we partition the selected vertices into k groups with k-means (we use $k = 20$, large enough to cover most edits but small enough to keep the user interface uncluttered); for very large vertex sets, we run k-means on a fixed random subsample for speed and assign every original vertex to its nearest cluster center afterward. For each cluster, we then need a sphere that tightly encloses its points so that the resulting mask hugs the edited geometry without leaking into unmodified regions of the mesh. Computing the exact minimum enclosing sphere is expensive, and since we fit one sphere per cluster per edit and per frame of preview, the cost compounds quickly. We therefore use Ritter's bounding sphere algorithm [Ritter, 1990], which trades a few percent of tightness for a strictly linear-time, single-file implementation.

Ritter's algorithm runs in two passes over the points of a cluster. The first pass produces a reasonable initial sphere that already encloses a large portion of them, so that the second pass has as little correction work to do as possible. Rather than picking two points at random, Ritter exploits the fact that the diameter of any point cloud is at least as large as its extent along any single coordinate axis. Concretely, the algorithm sweeps once through the cluster's vertices and records six axis-extreme points: the points with minimum and maximum x, y, and z coordinates. Among the three resulting axis-aligned pairs (min-x/max-x, min-y/max-y, min-z/max-z), it selects the pair whose two members are farthest apart in Euclidean distance. Call these two points p and q. The initial sphere is placed with its center at the midpoint of p and q and its radius equal to half of $|p - q|$, so that p and q lie exactly on its surface. This pair is what we earlier referred to as "nearly-antipodal": it is not guaranteed to span the true diameter of the cluster, but using axis extremes makes it a cheap and surprisingly tight starting estimate, which is important when the cluster represents a localized edit region whose shape is anisotropic (e.g. a long thin protrusion on the mesh).

The second pass then walks through every vertex in the cluster and tests it against the current sphere. If a vertex lies inside, nothing needs to be done; the sphere already covers it. If a vertex x lies outside, the sphere must be enlarged *just enough* to swallow it, while keeping all previously-covered vertices inside. Ritter does this with a small local update rather than recomputing the sphere from scratch. Let c and r be the current center and radius, and let $d = |x - c|$ be the distance from the offending vertex to the current center. The new radius is set to the average $r' = \frac{1}{2}(r + d)$ and the new center is shifted along the line from c towards x by exactly the amount needed so that x ends up on the new sphere's surface and the far side of the old sphere also still lies on (or inside) the new surface. Geometrically, the new sphere is the smallest sphere along that line that contains both the old sphere and the new vertex: its diameter is the segment from x to the antipodal point of x on the old sphere. Because the update only ever grows the sphere and always continues to contain the previous one, every vertex processed earlier in the pass remains enclosed, so a single sweep is sufficient.

The whole procedure runs in $O(n)$ time for n vertices in a cluster, since each pass is a single linear scan and each per-vertex update is constant work. Ritter reports that the resulting sphere is typically about 5% larger in radius than the true minimum enclosing sphere, with worst cases in the 5–20% range depending on the point distribution. For our mask construction, this overshoot is harmless: we immediately clip each sphere's radius by the safety cap (the distance

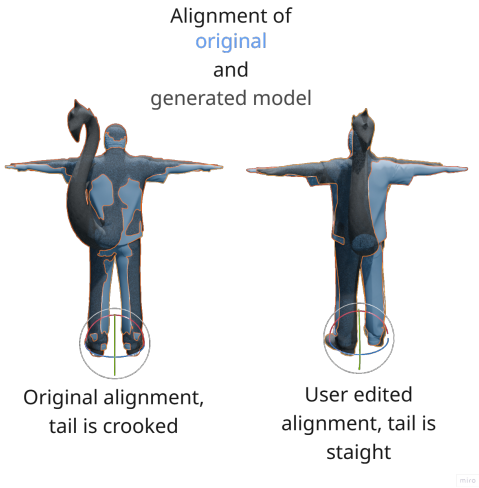

Figure 5.8: Image showing how users can rotate the model to reposition the edited region.

from the sphere's center to the nearest *unsafe* original vertex, scaled by 0.9), which guarantees that the sphere cannot reach into geometry that should remain untouched, regardless of how loose Ritter's estimate was. In practice, the safety cap, not Ritter's sub-percent looseness, is the binding constraint on the final radius. After clipping, spheres covering fewer than three vertices are pruned, since they correspond to sparse outliers and would clutter the interface. For the "hide region" we implement a similar method.

Finally, every sphere radius is multiplied by a small inflation factor to leave a soft margin for downstream blending. Every sphere is also given a standard "smoothing weight"; this can be modified by the user in a later step.

5.4 Optional Repositioning of the Generated Model and the Edit Spheres

Before the merging step, the user can optionally reposition the generated model together with its corresponding edit mask. This provides additional user control by allowing the model to be translated, rotated, and scaled to improve the alignment between the generated model and the original model. As a result, the user can influence the appearance of the final merged output.

This functionality is particularly useful for fine-grained edits. For example, when adding a tail to a human model, the generated tail may initially be slightly too small, rotated incorrectly, or misaligned with the center of the body. The user can then adjust the scale, rotation, and position of the generated model to achieve the desired alignment before merging. This is visualized in Figure5.8

In practice, the generated mesh and the edit mask are transformed together, since the edit mask is already correctly aligned with the generated model. The user can also separately add, remove, and move any edit sphere. This is useful if the edit masks encapsulate too much/little of the proposed edit (which can happen, especially if the model does not match the images properly), or if the user wants to add a part of the geometry from the edited mesh.

The user can also change the smoothing factor for each sphere; this can be useful for the later step, to have some regions smoother than others. In theory, it is very hard to assign this smoothing weight automatically/algorithmically; in our experience, it is based on user preference, and therefore, we give the users the freedom to set it themselves.

5.5 Selective Remeshing and Mesh Reconstruction

The previous stages give us everything that the final step needs: an original mesh, a generated mesh that contains the desired edit, an alignment between the two, and a region selection that tells us which vertices on each mesh belong to the edit. The job of the reconstruction stage is to combine all of this into a single output mesh in which the geometry inside the selected region comes from the generated model, the geometry outside it comes from the original, and the seam between the two is invisible. An important requirement here is that the geometry of the original part stays exactly intact.

The naive approach, which is to cut both meshes along the selection boundary and stitch the two open patches edge-to-edge, fails for the same reason that motivated this thesis in the first place: the two patches were never built to fit together. Their boundary loops have different lengths, different vertex counts, and different curvatures, and they are usually a few centimeters apart even after alignment. Welding them directly produces a long, jagged crack that no amount of smoothing can hide. In addition to this, the vertex density of the two models might not be the same, making it very visible that the models don't "belong together".

An alternative approach is presented in CraftMesh [Jincheng et al., 2026], which applies Poisson blending to seamlessly merge two parts of a mesh. As the authors themselves note, however, classical Poisson Mesh Editing cannot be applied directly to meshes, since it requires a one-to-one vertex correspondence between source and target, which is generally unrealistic [Jincheng et al., 2026]. To circumvent this, CraftMesh first converts the mesh into a Signed Distance Field (SDF), performing Poisson blending in the SDF domain before extracting a new mesh. While this produces visually seamless results, it comes at the cost of discarding the original mesh's explicit geometry: vertex positions, connectivity, and topology are all lost during the conversion to the implicit SDF representation. The resulting mesh is an entirely new surface extracted from the SDF, bearing no correspondence to the original vertices. Using this method directly is thus unsuitable for our use case, where preserving the original mesh structure is a strict requirement.

We therefore take a different route; we use the continuous remeshing pipeline of Palfinger [Palfinger, 2022], which is the same backbone that CMD [Li et al., 2025] uses for its inverse-rendering loop. The key idea is to do inverse rendering to approximate the model generated with Poisson blending, since this produces a visually seamless and correct result. The only difference here is that we do not move the original vertices that are outside of the edit mask. Instead, we only touch the vertices inside of the edit mask and in the "transition zone" to make the models seamlessly integrate with each other. This also gives us more user control in terms of smoothing the model and matching the average inter-vertex distance of the original model. Additionally, like the paper notes, it is a faster alternative to inverse rendering.

5.5.1 Initial Mesh Assembly

In an earlier version of the program, the pipeline began with the original mesh and applied continuous remeshing directly on it. This is also the initial model used in CMD [Li et al., 2025]. This approach worked well for certain proposed edits, primarily because the original mesh does not contain any holes or missing faces in its geometry. When such topological defects are present, however, a different problem emerges, which we refer to as the *propagating holes problem*.

We define the propagating holes problem as an issue that arises in Palfinger's implementation of continuous remeshing, stemming from the fact that the method cannot resolve topological defects [Palfinger, 2022]. Specifically, if the initial model contains even a single missing face, the remeshing process cannot fill it, and the defect persists throughout the reconstruction. As illustrated in Figure 5.9, the hole present in the initial model continues to propagate across iterations and remains unfilled in the final result. This is undesirable because, unless explicitly requested, the reconstructed model should be airtight and free of missing faces.

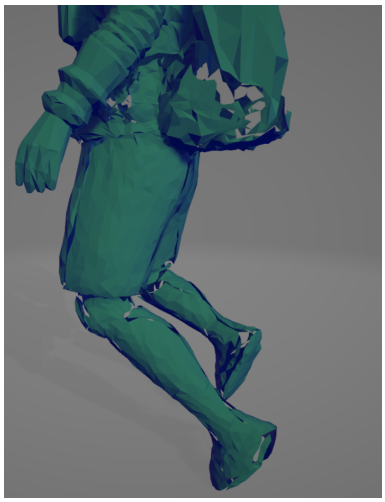

Figure 5.9: Image showing the propagating holes problem. This is what happens if I do not run the hole-filling algorithm first.

Since the initial model in this version of the pipeline is the original mesh, it will naturally be free of missing faces unless it was specifically designed to contain them. Consequently, the final result will also be free of holes, which is the desired outcome. However, a different limitation arises in this setting. When the intended edit requires a substantial addition of geometry, for example, extending a face mesh into a full-body model, the remeshing algorithm must generate a large amount of new geometry. Due to several factors discussed in later sections, the scale of this geometric expansion exceeds what the system can handle reliably. The vertex displacement becomes too large, causing holes to appear in the mesh, and the reconstruction fails to converge to the desired result.

To address this limitation, we explored an alternative approach for constructing the initial mesh. Rather than using the original mesh in its entirety, the initial model is assembled as a hybrid: vertices within the edit mask are taken from the generated model, while vertices outside the edit mask are taken from the original model. This preserves the unedited regions of the original geometry while incorporating the target shape in the edited region. However, since the original and generated meshes do not align perfectly at their boundaries, this combination introduces a gap between the two sections. To close this gap, we designed a hole-filling and stitching algorithm, described in the subsection below 5.5.2.

The model produced by this method appeared to work well initially, but during our experiments, we encountered a recurring issue: when the two meshes differed significantly at the boundaries, the hole-filling algorithm would occasionally snap a vertex to a distant, unrelated vertex rather than its nearest neighbor on the opposite boundary loop. Although we don't know the exact cause of this issue, we attribute it primarily to the greedy nature of the algorithm. Since the algorithm performed correctly in most cases, we addressed the problem as follows. Rather than taking the vertices strictly inside the edit mask from the generated model and strictly the outside vertices from the original, we introduced an overlap of two "boundary rings," so that the boundary regions contained vertices from both meshes. Instead of immediately closing the resulting gaps, we first ran several iterations of the continuous remeshing step described in Section 5.5.3. After this, any holes or gaps present in the initial mesh persist through the remeshed result, but the distance across those gaps is substantially reduced, making the subsequent hole-filling step considerably more reliable. This will be the initial mesh we start the reconstruction with in our final implementation.

5.5.2 Hole Filling and Stitching

Combining the original mesh outside the edit mask with the generated mesh inside the edit mask produces the right hybrid shape, but it leaves a band of unfilled space along the boundary between the two regions. Because the two meshes were sampled independently, they have no reason to share vertices on the cut surface, so the gap shows up as one or more closed boundary loops where triangles are missing on both sides. The role of this stage is to close that gap and produce a single watertight mesh that can be passed on to the remeshing step.

Before explaining our method, it helps to understand why the usual way of filling holes in 3D meshes does not work well here. The classic technique was created by Liepa [Liepa, 2003], building on earlier work by Barequet and Sharir [Barequet and Sharir, 1995]. The basic idea is simple: given a hole in a 3D mesh, you look at the edge around it and figure out the best way to cover it with triangles, where "best" means small triangles that meet their neighbors at smooth angles rather than sharp ones. This works very well when the hole is small, flat, and has a single clean edge running around it.

The situation we face, however, is different. The gap between the original and generated regions is rarely a single hole; it consists of *two* boundary loops facing each other across a non-trivial distance, one belonging to the original mesh and the other to the generated mesh. If we simply apply Liepa's algorithm, it has no idea which loop a vertex belongs to and tends to close each loop on its own, leaving the two halves of the mesh disconnected rather than joining them. We confirmed this empirically. Our algorithm is therefore designed around the specific structure of the gap: a pair of nearby loops that face each other, which we close with a strip of triangles whose edges only ever connect vertices on opposite sides. This stitching idea goes back to Barequet and Sharir [Barequet and Sharir, 1995], who used it to repair small CAD defects. We adapt it to the kind of gaps that hybrid mesh assembly produces.

The procedure consists of a few sequential steps, executed in order on the assembled mesh.

Cleanup and vertex merging. The hybrid mesh is first passed through a quick cleanup that removes triangles with zero area and exact duplicates that can be introduced by the assembly process. After that, we deal with another subtle issue: the two regions can contain vertices that occupy almost the same point in space but have different indices, simply because they were sampled from two different meshes. If we leave them as is, the edges between them are not recognized as shared, and the gap gets fragmented into many tiny holes rather than one or two large ones. To fix this, we build a k-d tree over the vertices inside the edit mask and merge any pair whose distance is below a small threshold ε. The merge is restricted to the mask region so that the rest of the original mesh is preserved exactly.

The chaining problem. A naive merge has a well-known failure mode called *chaining*. To see why it arises, imagine four vertices a, b, c, d arranged along a smooth curve in space. Each consecutive pair is within ε, so the algorithm links them all into a single cluster. But the endpoints a and d can be arbitrarily far apart. The threshold ε was meant to express "these two points are essentially the same location," but through a chain of small hops, the algorithm inadvertently merges points that clearly are not. This is illustrated in Figure 5.10.

To prevent this, we cap the *diameter* of each merged cluster, meaning the largest distance between any two of its members. Clusters that grow too large are split using hierarchical clustering, which guarantees that no two points in the same cluster end up further apart than the cutoff. Each remaining cluster is then collapsed to a single point at its centroid, and the face indices are updated accordingly.

Boundary detection. To know where the gaps are, we need to find the open edges of the mesh. We do this by walking through every triangle in the order its vertices are listed and recording each of its three edges as a directed half-edge. An interior edge shared by two triangles is traversed in opposite directions by its two neighbors, so both directed versions appear in our

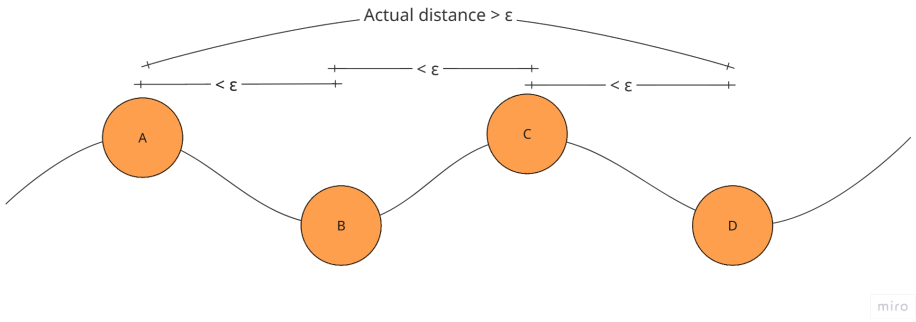

Figure 5.10: Visualization of the chaining problem.

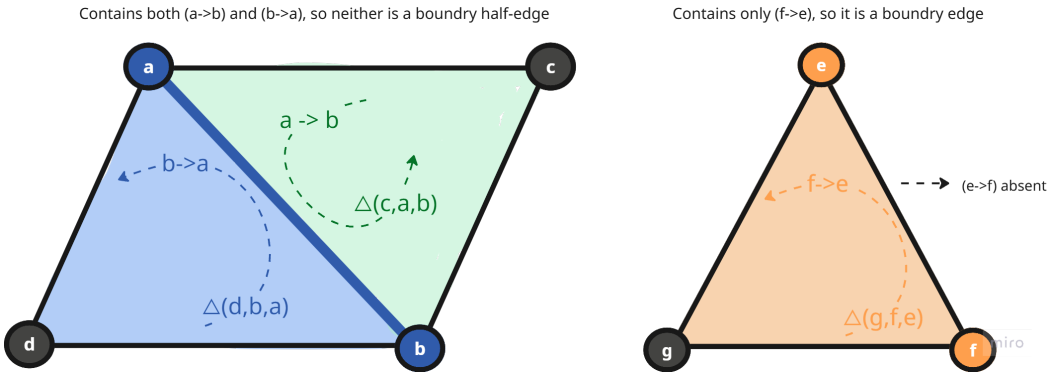

Figure 5.11: Visualization of boundary edges.

record. A boundary edge, on the other hand, has only one adjacent face, so only one direction appears. As shown in Figure 5.11 (left), the shared edge a–b between triangles $\triangle(d, b, a)$ and $\triangle(c, a, b)$ appears in both directions and is therefore interior. In Figure 5.11 (right), the edge e–f of triangle $\triangle(g, f, e)$ has no counterpart on the other side and is recognized as a boundary edge. Together, the boundary edges form one or more directed cycles around the holes. If none are found, the mesh is already watertight, and the rest of the procedure can be skipped.

Tracing the loops. The boundary edges then need to be assembled into closed loops before they can be stitched. In a clean mesh, this is straightforward, but the merge step can occasionally create vertices that belong to two different loops at the same time. At such a vertex, a naive walk does not know which way to go, so we pick the candidate whose endpoint is closest in 3D, on the assumption that the unrelated loop's edge is almost always farther away. The walker continues until it returns to its starting vertex, and the result is a list of closed loops, each represented as an ordered sequence of vertex indices.

Closing the gap. Once the loops have been traced, we close the gap. Not every loop is part of the inter-region gap; small holes elsewhere in the mesh produce their own loops, too. To tell them apart, we compare every pair of loops by how closely and uniformly they run parallel to each other: two rims bordering the same gap face each other closely, while two unrelated loops on opposite sides of the mesh do not. The closest-facing pair is matched first, then the next, and so on.

Each matched pair is then connected by a *zigzag stitch*. Two cursors start at one vertex of each loop, and at every step, the algorithm looks at the two possible next triangles, one that advances along the first loop and one along the second, and picks whichever has the shorter cross-edge. That triangle is added to the mesh, the winning cursor moves forward, and this

continues until both loops are fully consumed. Because every new triangle connects a vertex on one loop to a vertex on the other, the new geometry stays local to the gap and never spans across the model interior. The greedy advance also handles loops of unequal length naturally. Since the relative orientation of the two loops is unknown beforehand, the procedure is tried in both directions with several different starting offsets, and the configuration with the lowest total edge length is kept.

Loops that did not find a partner are typically small holes inside one of the two regions, and we close them with *ear clipping*: at each step, we pick the vertex whose two non-adjacent neighbors are closest together, form the triangle between them, and remove the vertex from the loop. This repeats until only three vertices remain. Loops larger than a configurable size limit are skipped, as a flat cap is unlikely to be correct for a large hole and usually points to a deeper problem upstream.

After stitching and capping, a final cleanup pass removes any new degenerate or duplicate triangles and makes the winding order of the new faces consistent with the surrounding surface. The result is a clean, closed mesh that is ready for the next processing stage.

5.5.3 Selective optimisation with continuous remeshing

At this point, we have an initial mesh that is topologically correct: the unedited regions still contain exactly the original vertices, the edited region contains the generated geometry, and the gap between them has been bridged by the stitching procedure of the previous section. But this mesh is not yet geometrically "perfect". The stitch strip is a band of small triangles whose orientation and density do not match either neighbor; the boundary between original and generated geometry is still visible as a discontinuity in shape and curvature, and the vertex density across the edited region rarely matches that of the original. The remaining task is to deform and resample the mesh inside the edited region so that it blends smoothly into the surrounding original surface, while leaving the original geometry untouched outside. As mentioned before, we perform this step using the continuous remeshing framework of Palfinger [Palfinger, 2022], which is also the inverse-rendering backbone of CMD [Li et al., 2025]. The framework alternates between two operations: a gradient step that moves vertices to minimize an image-space loss and a remeshing step that locally collapses, splits, and flips edges to keep triangles well-shaped at a target edge length. Our use of this framework differs from the original in several important ways, all of which follow from the constraint that the original geometry must be preserved exactly. We describe the loss we minimize, the way vertex movement is restricted, how smoothing is applied, and how the remeshing schedule is structured in the following paragraphs.

The optimization target. Recall from earlier in this chapter that we use the Poisson-blended mesh as the reference: it already contains the seamless transition we want, but its explicit vertex positions cannot be adopted directly because they do not correspond to any vertex of the original mesh. Inverse rendering allows us to use this reference *visually* rather than geometrically. We render the Poisson reference from a ring of cameras placed around the model and store, for each view, the resulting normal map and silhouette alpha. During optimization, we render the current working mesh from the same cameras and compute the squared error between its normals and silhouette and those of the reference. Minimizing this loss pulls the working mesh towards a shape that *looks* like the Poisson reference from every direction, without ever copying its vertex positions. This is what allows us to inherit the seamless blending of Poisson while keeping our own explicit topology. An example of the Poisson blended result is shown in Figure5.12, here we can see that the two models have indeed been smoothly connected to each other. The downside here is that, as you can see in the middle and right-most images, the edges are not preserved perfectly.

Restricting movement to the edited region. The image-space loss alone has no notion of which vertices are allowed to move. If we let it act on the entire mesh, it would happily nudge unedited vertices to better explain the rendered pixels, and the original geometry would no

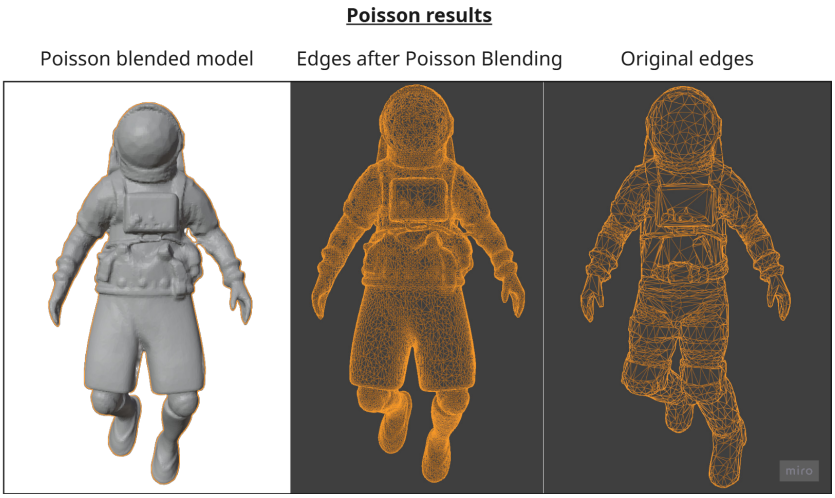

Figure 5.12: Image showing the Poisson blended model, the edges after Poisson blending, and the original edges.

longer be preserved exactly. This is what happens in CMD [Li et al., 2025]. We prevent this with two complementary mechanisms. First, an *anchor loss* measures the squared displacement of every vertex outside the edit region from its initial position, with a very large weight; in practice, this term forces those vertices to stay where they were. Second, after "backpropagation," we explicitly zero the gradient on every vertex outside what we call the *optimization mask*. The optimization mask is the edit mask grown outward by a few rings of mesh neighbors: it contains the vertices that are clearly part of the edit, plus a thin band of original-side vertices immediately around the seam. The band is necessary because the seam itself runs through original-side vertices, and these need a small amount of freedom to relax into the new geometry for a "smooth" transition. Outside this band, the gradient is hard-zeroed, and the anchor loss does the rest. Together, these two mechanisms guarantee that the unedited part of the original mesh remains bit-for-bit identical to the input.

Matching vertex density. A visually seamless edit requires more than a smooth shape transition: the average distance between neighboring vertices on the two sides of the seam should also match, otherwise the eye picks up the change in mesh resolution as a visible scar even when the surface itself is smooth. Before optimization begins, we compute the average edge length of the original mesh, and we configure the remesher's target edge length and the lower and upper bounds it is allowed to drift between, around this value. The effect is that whenever a triangle inside the edit region becomes too small or too large compared to the surrounding original mesh, the remesher collapses or splits it back to the surrounding density. The original vertices themselves are never touched, since they are anchored, so the matching happens entirely on the edit side.

The transition zone. A naive selective optimization would move only the strictly-inside-mask vertices and leave everything else fixed. The result is a clean shape inside, a clean shape outside, and a sharp crease at the boundary where the two meet, exactly the artifact we are trying to avoid. We borrow the same idea that makes Poisson blending work: instead of a hard interior/exterior split, we define (as mentioned before) a transition zone of a few rings of vertices straddling the mask boundary, and we let movement and smoothing fade in gradually across this zone using a cosine ramp from zero at the boundary to full at the deep interior. Concretely, the gradient that the optimizer applies to a vertex is multiplied by a per-vertex transition weight: a vertex sitting right on the boundary barely moves, a vertex two rings inside moves a little, and a vertex deep inside the edit region moves freely. The same ramp is applied to the smoothing term, so that smoothing also fades to zero exactly at the boundary and

does not pull the anchored original vertices. This is what makes the seam genuinely invisible: the geometry, the smoothing, and the vertex motion all blend continuously across the boundary instead of switching on at a single edge.

Smoothing. The continuous remeshing framework provides a built-in Laplacian smoothing term that uniformly attracts each vertex towards the centroid of its neighbors. We tried using it directly and found that it works well on roughly convex shapes, but produces severe shrinkage on thin or extruded features. A clear example is a winged character: the wing tips have only inward-pointing neighbors, so a uniform Laplacian pulls them steadily towards the body, eroding the wing edge over the course of the optimization. We therefore disable the built-in smoothing entirely and inject our own Laplacian term into the gradient with two modifications. First, it is gated by the same transition-zone ramp as the rest of the gradient, so the mesh-based smoothing strength very slowly fades at the boundary and never acts on anchored vertices. Second and most importantly, it is scaled per vertex by a user-controllable smoothing factor attached to each mask shape: a region marked with a low smoothing factor receives a gentle Laplacian and keeps fine detail, while a region marked with a high smoothing factor is smoothed aggressively. This per-region control might be important because different parts of an edit can have different needs; the neck of an extended body might want strong smoothing to blend into the head, while the body itself wants light smoothing to keep the generated detail intact. In practice we observe that we get the best results when all edit spheres have the same smoothing factor because it makes the edit more "seamless" and it can look like it "belongs to" the original model. These smoothing factors work together. When a vertex is far away from the boundary but has a high mask shape smoothing factor, it will still be smoothed accordingly.

Scheduling remeshing and inner iterations. The original framework remeshes after every gradient step. This is well-suited to its setting, where the entire mesh is a single optimization target initialized from a sphere or similar primitive, but it is not what we want here. Remeshing every step in our setting introduces vertices faster than the optimization can place them meaningfully, and each new vertex starts at an interpolated position that does not yet reflect the loss, so the mesh accumulates noise. Most importantly, when remeshing at every step, the vertex distance will be far greater than the original.

We also observed that the way remeshing should behave changes across the optimization. Early on, the mesh is still further from the reference and benefits from being resampled often, so that the vertex distribution can reorganize as large-scale shape changes occur. This was especially important in an earlier phase of the implementation, when the initial model was the original model. Here, we needed to introduce extra remeshing steps early on to let the model "grow" and "explore" to encapsulate the proposed edit. This observation is also valid for the current initial model we work with, since exploration, especially around the boundary vertices, can be beneficial.

At later steps, the mesh is already close to the reference, so we want long uninterrupted stretches of optimization so that vertices can settle into their final positions and the smoothing terms can do their work without being constantly reset.

We therefore drive the remesher with an explicit schedule. The schedule begins with a few short intervals between remeshings, which gives the mesh several chances to redistribute its vertices while it is still moving substantially, and is followed by one long interval during which no remeshing occurs and the optimization simply refines what is already there. Each remeshing event itself runs several internal remeshing passes back to back, so that a single event can resolve multi-step changes in one go rather than spreading them across many gradient steps. The exact step counts and pass counts were chosen empirically to give reliable results within an optimization budget that completes in a reasonable time.

Mid-optimisation cleanup. As mentioned at the end of Section 5.5.2, our pipeline is permitted to start the optimization from a mesh that still contains small holes, on the assumption

that the gradient steps will pull the two sides of each hole close together, and a hole-fill afterward will be more reliable. This is exactly what happens at a fixed point partway through the optimization: we pause the loop, write the current mesh to disk, run the cleanup-and-fill routine of Section 5.5.2 with hole-filling enabled, reload the result, smooth the boundary lightly to remove any kinks introduced by the new triangles, and rebuild the optimizer around the cleaned mesh. The remaining gradient steps then refine the freshly-filled region together with the rest of the edit, so the patches blend in rather than being visible as a flat capping. This is what allows the initial assembly stage to be tolerant of small gaps and the final mesh to be airtight. This is done after the "short intervals" of remeshing, so that the model could have been explored already; there is no need for the model to be geometrically smoothed for this cleanup step. We have observed that it is even beneficial to have the smoothing step performed afterward.

Final model. After the selective remeshing steps described above, the resulting mesh satisfies all of the requirements set out at the beginning of this chapter. The unedited regions retain exactly their original vertices, connectivity, and topology. The edited region has been deformed and resampled to match the visual appearance of the Poisson-blended reference, with a vertex density that matches the surrounding original surface. The transition zone ensures that the boundary between the two regions is geometrically continuous, with no visible seam or crease.

In practice, the optimization may still leave behind isolated artifacts: small holes that were not fully closed during the mid-optimization cleanup, or degenerate triangles introduced by aggressive remeshing near the boundary. These are resolved by running the hole-filling and cleanup routine again of Section 5.5.2 one final time on the completed mesh. Because the mesh is already geometrically close to its target at this stage, the fill patches produced are small and blend in naturally with the surrounding geometry. The mesh produced by this final cleanup pass is the output returned to the user: a single, watertight surface in which the requested edit is seamlessly integrated into the original model, with the original geometry preserved exactly everywhere outside the edited region.

The output model after the remeshing step is shown in Figure5.13 in the third column. The fourth column shows the same output with textures.

5.6 Retexturing

The final step in the pipeline is retexturing. This is the part of the system where the most room for improvement remains, and it is also the step that, to some extent, breaks the guiding rule we set for the rest of the pipeline: that the original textures should be preserved wherever possible. Due to time constraints, we chose to retexture the full model in one pass instead of blending only the edited region into the original texture. The user will notice this in the final result, since the colors and surface details outside the edited region may differ slightly from the input model.

We reuse the multiview images that were already generated earlier in the pipeline (see Figure 4.4) and feed them into Meshy.ai, the same tool that was used for the initial 3D model generation. Meshy.ai accepts an image together with a mesh and returns a fully retextured version of that mesh. In practice, we only pass the front and back views of the edited multiview, together with the merged mesh produced by the ASMR-3D stage. Meshy then uses these two views as the visual reference and generates a new texture for the entire model.

Why only the front and back views? Initially, we passed the complete multiview (all four views: front, left, right, and back) into Meshy. This produced noticeably worse results: because each individual view occupied only a small part of the input image, the effective resolution per view was too low for Meshy to recover fine texture detail, and the resulting textures looked

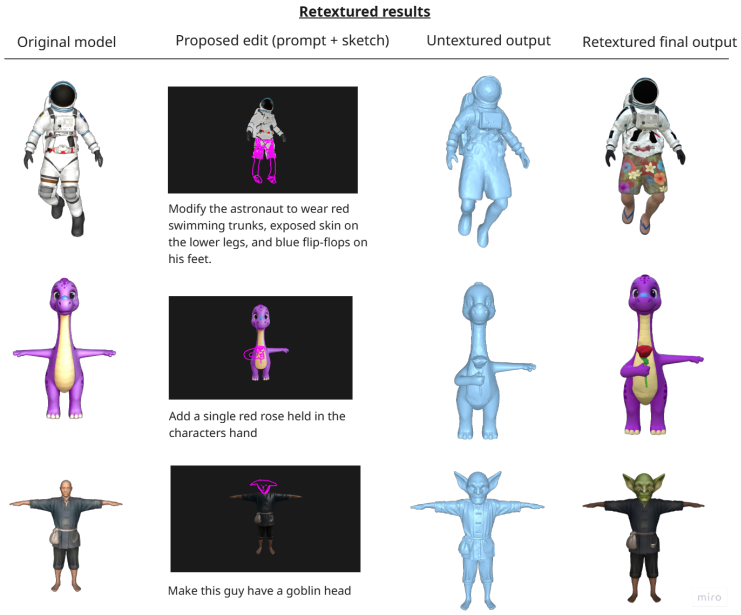

Figure 5.13: Snippet of Figure6.28, showing the end-to-end result. **Left:** original generated model with automatic textures. **Second row:** proposed edit, consisting of a user sketch overlaid on the model together with the accompanying text prompt. **Third row:** untextured output, given right after selective remeshing. **Right:** retextured final output after the system has applied the edit.

blurry and washed out. Restricting the input to the two most informative views (front and back) and giving each of them the full image area produced visibly sharper and more faithful textures. We also observed that removing the background from these two views before sending them to Meshy improved the result further, since the background pixels would otherwise compete with the object for the model's attention and occasionally leak into the generated texture.

Attempts at preserving the original texture. We did experiment with keeping the original texture on the unedited regions of the mesh and only generating a new texture for the edited region, in the same spirit as what we do for the geometry. The most direct attempt was to take the original texture and the newly generated texture and smoothly blend them together across the seam using a simple weighted average in texture space. This consistently produced unpleasant artifacts: color bands along the boundary, mismatched lighting between the two halves, and visible seams where the two textures had different overall tones. A more principled approach, in the style of CraftMesh [Jincheng et al., 2026], would be to use a Poisson blending method directly on the texture, which would let the colors of the edited region transition smoothly into the original texture without introducing visible boundaries. We did not have the time to implement this within the scope of this thesis, and the limitations and possible improvements of the current approach are discussed further in Chapter 6.12.

5.7 Generalizability

Although the experimental evaluation was conducted exclusively on meshes generated by the 3D model generator Meshy, the pipeline itself imposes no fundamental restriction on the source of its input. Any manifold mesh can serve as the original model; it will work better if its edge lengths are reasonably uniform across the surface. A highly irregular vertex distribution will make it very visible that the edited part does not belong to the original model. This issue arises because the vertices that went through the selective remeshing phase will have a very uniform edge length, especially if the Laplacian smoothing factor is set higher.

Beyond single-model editing, the core components of the pipeline generalize further. The stages described in ASMR-3D 5.1 can, with appropriate parameter tuning, be used to merge any two arbitrary meshes along a user-defined boundary. This makes the approach applicable to a broader class of problems than the specific editing task it was designed for, including part substitution, geometry transplantation between scanned models, or the integration of procedurally generated components into hand-authored assets.

Finally, the pipeline is not limited to a single edit per model. Because the output of one run is itself a valid mesh, edits can be applied iteratively: each pass targets a different region or refines a previous result, giving the user fine-grained control over the final shape. In principle, this allows any number of independent edits to be composed into a single model, with the user able to inspect and adjust the result at each stage until the output matches their intent.

Chapter 6

Evaluation

This chapter discusses the experimental arrangement and usability analysis of the proposed pipeline. The evaluation is organized to follow the same order as the pipeline itself: each stage is first evaluated in isolation, using both objective metrics and subjective user feedback collected during the user study, after which the system is assessed as a whole. The chapter closes with a discussion of generalizability, overall system-level results, and a critical reflection on the findings, constraints, and possible enhancements.

The four research questions formulated in Chapter 1 are all addressed in this chapter: RQ1 (region-based selective remeshing) is evaluated in section 6.8, RQ2 (multimodal user input) is covered by Sections 6.2–6.3, RQ3 (preservation of the original geometric structure) is the focus of Sections 6.6–6.8, and RQ4 (usability and effectiveness) is addressed in Section 6.11 on the basis of the UEQ and the custom satisfaction questionnaire.

6.1 Experimental Design

To evaluate the proposed pipeline, a within-subjects user study was conducted with fourteen participants. Each participant completed two 3D model editing tasks while interacting with the full pipeline and filled in a structured questionnaire after every stage of the pipeline. The study collected three complementary types of data: quantitative metrics derived directly from the pipeline output (geometric, topological, and image-similarity scores), behavioral data extracted from a recording of the participant's session (if any interesting behavioral patterns are recognized by the researcher (me), it is discussed), and subjective ratings on a five-point Likert scale, together with free-text remarks. The remainder of this section describes the goals of the evaluation, the participants, the tasks, the procedure followed during each session, the quantitative metrics that will be reported, the statistical treatment of the results, and the apparatus on which the evaluation was performed.

6.1.1 Goals and Hypotheses

The purpose of the evaluation is to assess the four research questions stated in Chapter 1. RQ1 concerns the region-based selective remeshing method (ASMR-3D) and is evaluated through topological and geometric measurements of the merged mesh. RQ2 concerns the multimodal input modalities (sketch, prompt, and reference image) and is evaluated through the user-facing items of the questionnaire associated with the sketching and image selection stages. RQ3 concerns preservation of the original geometric structure during a local edit and is the focus of the ASMR-3D evaluation, where the anchor-vertex placement error quantifies exactly how much of the original mesh remains untouched. RQ4 concerns overall usability and effectiveness and is evaluated using the User Experience Questionnaire (UEQ) as well as additional questions regarding usability.

Each pipeline stage is evaluated against one concrete hypothesis, mirroring the structure of the rest of this chapter. For positioning, sketch, and prompt, the expectation is that the interface is rated as easy to use and as producing inputs that the user is confident about. For image generation, the expectation is that the four candidates are diverse but each remains faithful to the original outside the sketched region, and that the image represents the user's intent. For alignment, the expectation is that at least one of the methods generated good results, and we will also compare which method is chosen the most. For edit-mask generation, the expectation is that a majority of participants accept the automatically produced mask with limited manual modification. For the ASMR-3D merge, the central expectation is that the output is watertight, manifold, and free of self-intersections, that the anchor-vertex placement error is exactly zero, and that participants cannot perceive the seam and are happy with the final result. For the overall system, the expectation is that the UEQ scores reach at least the *above average* band of the official UEQ benchmark.

Because the sample size is small, these hypotheses are treated as qualitative expectations to be supported or refuted by the collected evidence, rather than as the targets of formal significance testing. The reasoning behind this choice is given in Section 6.1.6.

6.1.2 Participants

Fourteen participants were recruited for the study. Prior to the fourteen real sessions, two pilot studies were conducted with a single participant in order to identify problems in the study protocol, refine the wording of the questionnaire, and verify that the recorded sessions contained the information needed for the post-hoc analysis. The pilot data is not included in the reported results, but the lessons learned from the pilots, in particular a small reordering of the verbal-assessment step and a clarification of the wording for several Likert items, were carried over into the final protocol, as well as some minor bug fixes in the implementation.

Each session was preceded by a screening questionnaire, the items of which are listed in the system implementation. It captured self-reported information along five dimensions relevant to the study: age, general computer proficiency, drawing or sketching ability, how frequently the participant uses generative 3D model generation tools (e.g. Meshy, Shap-E, Point-E), whether they have ever used a generative 3D model *editing* tool and whether they use a traditional 3D modelling package (e.g. Blender, Maya, 3ds Max, Fusion 360) together with their proficiency in that package. The two skill items (computer proficiency and sketching ability) and the conditional 3D modeling proficiency use a five-point ordinal scale, with the labels shown in Table 6.1. The generative 3D model item is a five-point frequency scale (Never → Very often), and the generative 3D editing item is a single yes/no question.

The resulting distributions are shown in Figure 6.1. None of the fourteen participants has ever used a generative 3D editing tool, thirteen have never used a generative 3D modeling tool either, and only six have any experience with a traditional 3D package, all of them at the lower half of the proficiency scale. At the same time, the same group is highly computer-literate, with six of fourteen reporting expert-level computer proficiency. The sample is therefore representative of the intended target audience: people who are comfortable using software in general but have no prior exposure to 3D editing tools.

Participation in the study was voluntary, and all participants signed an informed-consent form prior to the session. The form explicitly covered four points: the purpose of the study and what participation involved, consent to being screen-recorded and observed by the researcher during the session, consent for the survey responses and the generated 3D models to be used anonymously in this thesis and in possible academic publications, and the right to withdraw at any moment without giving a reason. The same form also asked participants to confirm that they had read and understood the procedure. No personally identifying data is reported in this thesis, and the screen recordings and generated models are stored on a single research machine to which only the author has access.

Table 6.1: Participant demographics across the fourteen sessions. Self-reported scores use the five-point Likert scale shown in the right-most column. One participant did not report their age.

Variable	Result	Scale / labels
Age (years)	mean 23.9, median 23, range 20–35 ($n = 13$)	—
Computer proficiency	median 4, mean 3.6	1: Novice → 5: Expert
Sketching ability	median 2, mean 2.6	1: Poor → 5: Excellent
Use of generative 3D modelling tools	13 Never, 1 Rarely	1: Never → 5: Very often
Use of generative 3D editing tools	0/14 ever used	yes / no
Use of traditional 3D software	6/14 (Blender 4, Fusion 360 2)	yes / no
Proficiency of those who do	median 1, mean 1.8	1: Novice → 5: Expert

6.1.3 Tasks

Each participant performed two editing tasks in succession.

Task 1 (guided). The first task is identical for every participant. The participant is given the same character mesh as input, together with a reference image of a goblin head and an example of the expected final result, shown beforehand in the study application. The goal is to replace the character's head with the goblin head. The guided task serves two purposes. Primarily, it gives every participant a moment to familiarize themselves with the pipeline before performing the more demanding free task and it allows all participants to be compared on the same target: the same input mesh, the same reference image, the same notion of correct edit-mask coverage. The instructions given to the participants is shown in Figure 6.2.

Something noteworthy we did for Task 1 was that we pregenerated all outputs beforehand, specifically for all steps until the auto mask generation. No matter what the input would be, the results would still be the same. We did this to save time in the user study. Because the first task's main goal is to get familiar with the program, we found it to be okay to pregenerate it to save time, since this task was the same for all participants. There was no one who mentioned it being pre-generated.

Task 2 (free).The second task is open-ended. Participants are directed to the Meshy Discover page [1] and instructed to pick any 3D model they find interesting and to propose any edit they would like to make. The free task is essential for two reasons. First, it captures realistic, creative use of the system, which is the intended deployment scenario: a user who already knows the kind of result they want and is using the system to produce it. Second, it exercises the generalizability of the pipeline. The categories of possible edits (attach, embed, remove, modify) are analyzed in Section 6.10, and the diversity of source meshes obtained in this way is what allows the chapter to make a claim about generalizability that goes beyond the goblin-head example. By letting the users choose freely, we have a better sense of the usability for different types of models and different types of edits.

The 3d models the participants can choose from are filtered to have 3k-10k polygon count and triangular topology. We choose 3k-10k polygon count because, as we will discuss in the Limitations section 6.12.2, it does not work well with low-poly meshes. Higher topology will make the generation and reconstruction take longer, so 3k-10k seems like a good middle ground. Also, we specifically use a triangular topology because our implementation is specifically designed for this.

The two tasks use exactly the same evaluation steps and the same questionnaire, which means that any differences in the recorded metrics or Likert scores between tasks can be attributed to the task itself rather than to a difference in instrumentation.

A representative selection of the input mesh, the proposed edit, and the final result for Task 1,

[1] https://www.meshy.ai/discover

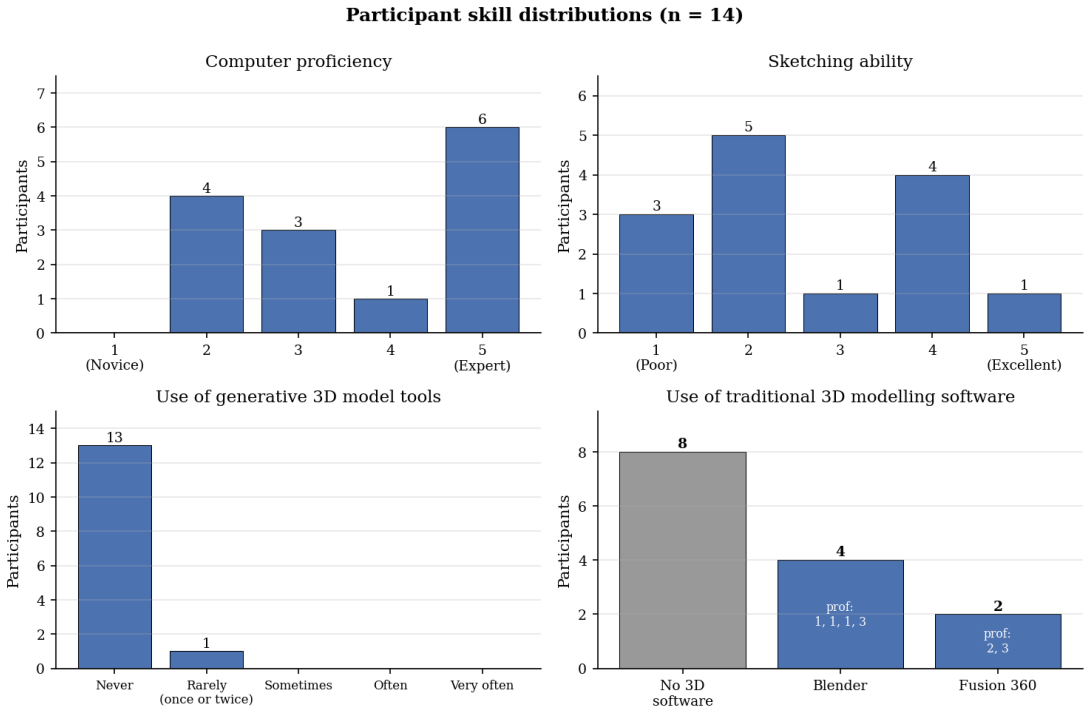

Figure 6.1: Distribution of self-reported skills across the Fourteen participants. The top row shows the five-point Likert distributions for general computer proficiency and sketching ability. The bottom-left panel reports how often each participant uses generative 3D modeling tools (none of the participants have ever used a generative 3D *editing* tool, so that the question is omitted from the figure. The bottom-right panel reports the use of traditional 3D modeling software, with the self-reported proficiency of the users overlaid on the bar (blank for non-users).

as well as a two examples of the edits chosen by participants in Task 2, are shown in Figure 5.13.

6.1.4 Procedure

Each session followed the same fixed procedure and lasted approximately 30 to 45 minutes. Sessions were held in person, with the researcher sitting next to the participant. The participant worked on a single laptop on which the pipeline and the study application were both installed. The researcher was present to answer protocol-level questions, to start each new task, and to be the conversational partner for the verbal-assessment step described below. The participant was encouraged to think aloud throughout the session, although thinking aloud was not enforced. If the researcher needed to intervene during the study, it was noted.

A session began with the informed-consent form, followed by the screening questionnaire. The participant then proceeded to Task 1. Within each task, the participant cycled through the pipeline stages in fixed order:

- Positioning the model and producing a sketch and prompt.
- Selecting one of four generated images and optionally editing it.
- A verbal-assessment step in which the participant was asked to describe to the researcher what they expected the intermediate steps and the final result to look like, before generating the 3D model. This step was only present in Task 2, to help understand their creative thought process and to stimulate them to think about creative "details". It is

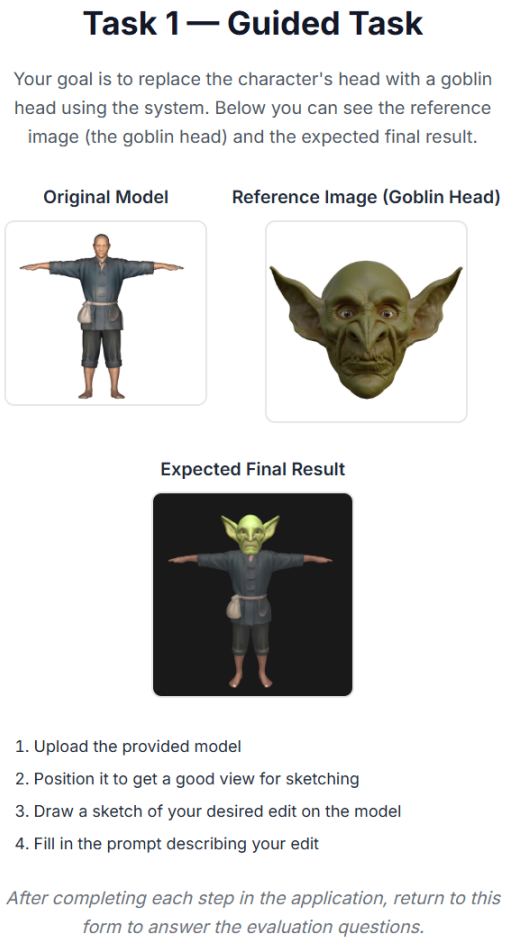

Figure 6.2: The instructions of Task 1, given to the participants.

also a mechanism to elicit honest expectations before the participant has seen the result, so that the subsequent Likert items measuring how well the result matches what was expected are not biased by the result itself.

- Choosing between the two alignment candidates, reviewing, and optionally modifying the automatically generated edit mask.
- Evaluating and optionally changing the generated edit mask.
- Generating the merged and retextured model and evaluating how well the edit integrates into the model.

After each of the stages, the participant returned to the study application and answered the Likert questions associated with that stage, as well as a free-text remarks field.

When Task 1 was completed, a short transition screen explained the free task and invited the participant to be creative. Task 2 then followed the same cycle as Task 1. After Task 2, the participant filled in the UEQ together with additional adoption items and free-text fields capturing what they liked most, what they liked least, what they would improve, and any final remarks.

The entire session, including the on-screen activity of the participant and the audio of the conversation with the researcher, was recorded with the participant's consent. The recording serves two purposes for the evaluation. It allows time-on-task to be recovered for each pipeline stage, since the study application itself does not log timestamps: the time intervals reported in the per-stage sections are extracted from the recorded video by me. It also allows the free-text remarks to be contextualized against the participant's behavior at the moment the remark was written.

6.1.5 Quantitative Metrics

The quantitative metrics are organized by pipeline stage. This subsection lists all reported metrics and explains the rationale for including each one. The corresponding numerical results are presented in the dedicated sections for each stage, following the same order used here. Ideally, each stage would be compared against other geometry-preserving 3D model editing systems. However, there are currently very few open-source research projects with a comparable focus, making direct benchmarking against similar systems impractical. As a result, the evaluation relies primarily on semantic correctness and the extent to which the output reflects the user's input and intended modifications.

Image generation. Two objective metrics are reported. Fidelity is measured as the mean absolute pixel difference between the chosen edited image and the original render, computed once over the foreground region and once over the region outside a manually set edit mask. The latter should be near zero by construction of the high-fidelity image editor and is included to verify that the property holds in practice. We have to define the edit region ourselves because the sketch mask might not include all pixels edited in the final image. Diversity is measured as one minus the mean pairwise cosine similarity of CLIP embeddings across the four candidate images, with DINO embeddings used as a robustness check. Higher diversity indicates that the four candidates represent meaningfully different interpretations of the same sketch and prompt. The distribution of which of the four images was selected by participants is also reported, in order to detect any positional bias in the four-image grid.

Multiview generation. Consistency between each generated view and its corresponding original view is measured by the mean absolute pixel difference, as in the image generation evaluation above, again measuring the diversity inside and outside of the manually set edit mask.

Generated 3D model. The fidelity of the generated 3D model to the original mesh is measured by a 36-view silhouette comparison. Both the original mesh and the expert-aligned gen-

erated mesh are filtered to keep only the faces that lie outside the expert-defined edit region, so that only the parts that are supposed to be identical between the two meshes are compared. Thirty-six camera positions are placed evenly on the unit sphere (Fibonacci sampling), and the two meshes are rasterized at each camera as black binary silhouettes against a white background, using an orthographic projection and an image resolution of 256×256. For each viewpoint, we report the intersection-over-union (IoU) of the two silhouettes and the per-pixel L1 difference and aggregate over the 36 views into a per-trial mean. The expert alignment (done by me) and the expert edit mask (also placed by me) are produced once per trial so that the metric does not depend on the algorithms evaluated in the later stages.

Alignment. Two metrics are reported per alignment method. The first is the mean nearest-neighbor Euclidean distance from the vertices of the aligned generated mesh to the original mesh, restricted to vertices outside the self-made edit mask. The second is the same nearest-neighbor distance computed against a manually-aligned version (by me) of the mesh instead of the original; this isolates the quality of the alignment method from noise introduced by the image-to-3D generator.

Edit-mask generation. The evaluation of the automatic mask is split into two complementary parts. The first part is an ablation that measures how much each individual cue contributes to the final algorithmic mask: the per-cue masks (geometric, image-difference, and multiview voting) and their combined-OR vote are each compared, at the vertex level, against the final mask produced by the post-voting clustering step, and we report precision, recall, and intersection-over-union per cue. The second part measures how close the final algorithmic mask is to the mask the user ended up with for that trial. The user mask is rasterized on the same vertex set as the algorithmic mask, both are converted into a binary per-vertex labeling, and precision, recall, and intersection-over-union are reported. The first part isolates the algorithmic behavior of the cue-and-clustering pipeline; the second part captures the human-in-the-loop quality of the resulting mask.

Selective remeshing. Two groups of metrics are reported. The topological group contains the non-manifold edge count, the boundary edge count (a count of zero, together with two-manifoldness, meaning the mesh is watertight), the self-intersection count, and the vertex distribution, summarised by the per-vertex valence histogram compared to the same histogram of the input mesh.

The vertex-preservation group contains the anchor surface error and the Chamfer distance computed over the unedited region. The anchor surface error is the mean point-to-surface distance from each input vertex outside the edit mask to the closest point on the output mesh, using the closest point on a triangle rather than the closest vertex, so that denser outputs are not unfairly rewarded for having more vertices in the neighborhood.

These metrics are computed for several variants of the output, where available per trial: the final mesh produced by the ASMR-3D pipeline, the aligned generated mesh itself, a naive baseline that stitches the two meshes without selective optimization, and the Poisson-blended reference. The mask used for the vertex-preservation and boundary-band metrics is the user's selected/edited mask.

Overall system. The UEQ items are aggregated into the six standard UEQ scales (attractiveness, perspicuity, efficiency, dependability, stimulation, novelty) according to the published scoring scheme [Schrepp et al., 2017a]. Results are reported both as absolute scores and against the official UEQ benchmark bands. Two adoption items capture whether participants would use the system again and whether they would recommend it.

6.1.6 Statistical Analysis

The sample size is fourteen participants, which is in the typical range for an exploratory usability study but small for formal inferential statistics. Descriptive statistics are therefore used throughout this chapter. For Likert data, the median and inter-quartile range are reported

as the primary summary; the mean is also reported in tables for compatibility with the official UEQ scoring scheme, which is defined in terms of means. For continuous metrics such as Chamfer distance, time-on-task, and mean pixel difference, the mean and standard deviation are reported.

A small number of comparisons appear in the per-stage sections, notably the comparison between the two alignment methods. The user study captures the participant's preferred method on each trial rather than separate scores for both candidates, so the appropriate analysis is a binomial test against the null hypothesis that the two methods are chosen with equal probability. This test is reported as a sanity check; the discussion is based on the observed proportion and on the accompanying free-text remarks rather than on the p-value. No claims of statistical significance are made on the basis of $p < 0.05$ alone, given the sample size; the results in this chapter are presented as evidence supporting or refuting the qualitative hypotheses of Section 6.1.1, rather than as the outcome of confirmatory tests.

6.1.7 Apparatus and Implementation

All sessions ran on a single laptop equipped with an NVIDIA RTX 4050 Laptop GPU and a 13th Gen Intel Core i5-13500HX CPU. Running every session on the same machine ensures that the time-on-task numbers extracted from the recorded sessions are comparable across participants and that no part of the variation in those numbers is attributable to hardware differences.

The study application that presents the consent form, the screening questionnaire, and the per-stage Likert questions is a Vue-based web application served locally. It supports English and Dutch through a shared internationalization layer; the wording of every question used in the study, in both languages, is the same as that used to derive the metrics described above. The application auto-saves the participant's answers continuously and allows a session to be resumed after an interruption.

The pipeline itself is the high-level prototype described in the preceding chapters of this thesis. It is implemented as a Python backend that performs all of the geometric processing, including alignment, edit-mask generation, and ASMR-3D selective remeshing, paired with a Vue frontend that uses Three.js to render the 3D models and to provide the interactive controls the participant uses during each session. The generative components on which the pipeline relies are external services accessed from the Python backend: the prompt refinement step uses the Gemini 3 Flash Preview model, the single edited image and the four multiview images are produced by the Gemini 3 Pro Image Preview model, and the image-to-3D step is delegated to the Meshy service.

6.2 Positioning, Sketch and Prompt

The first stage of the pipeline is the entry point through which the user expresses what part of the model is going to change and how. It asks the participant to orient the input mesh, to draw a magenta sketch on top of the chosen view, and to write a short textual prompt, which is then refined by the language model. They can optionally include a reference image. The evaluation of this stage focuses on whether non-expert users can produce inputs that are rich enough for the downstream stages without feeling overwhelmed by the interface. Because the stage has no purely geometric output that can be measured against a reference, the assessment relies primarily on the Likert questionnaire that the participant filled in immediately after the stage and on the free-text remarks captured at the same time, complemented by a single observation made by the researcher across all fourteen sessions.

6.2.1 Methodology

Each participant answered seven Likert items on a five-point scale (1 = strongly disagree, 5 = strongly agree) for both Task 1 and Task 2. The items cover the three sub-steps of the

stage:

- **Positioning the model in the viewport** (`positioning`)
- **Drawing the sketch itself**: how easy the drawing was (`sketchEasy`), whether the drawing tools were adequate (`sketchTools`), and whether the resulting sketch clearly communicated the intended edit (`sketchClear`)
- **Writing the textual prompt**: how easy the prompt was to formulate (`promptEasy`), what information it required (`promptRequired`), and how well the system-refined prompt reflected the participant's intent (`promptClear`)
- **Confidence**: a single global item asking how confident the participant was about the inputs they had just produced (`confident`)

After answering the Likert items, the participant could leave a free-text remark in a dedicated *difficulties* and *other remarks* field. In addition, the researcher noted, for every participant and every task, whether the participant kept the original prompt they had typed or accepted the refined version produced by the language model.

6.2.2 Quantitative Result

The most striking quantitative observation for this stage concerns the prompt-refinement step rather than the questionnaire. Across all fourteen participants and across both tasks, every single participant accepted the refined prompt produced by the language model in preference to the prompt they had typed themselves, giving a unanimous 14 out of 14 in both runs. It is worth noting that this was an informed choice: participants were presented with both the original and the refined prompt side by side and were clearly told they were free to select either version. This is consistent with the free-text feedback discussed below: although a few users found the refined prompt confusing at first, none of them chose to revert to their own phrasing once they had seen the rewritten version. The implication for the system is that the prompt-refinement stage is not merely an optional aid but a step that participants treat as a genuine improvement over their own input.

The prompt-refinement step was something that came out of the pilot tests. Earlier implementations did not include it and the pilot test user found it "hard to know what the system wanted to have in the prompt", changing it to a structured prompt seemed to make the prompting system even more confusing.

6.2.3 User Feedback

The Likert results for the seven items are summarized in Figure 6.3. Each row reports the distribution of the fourteen ratings over the five points of the scale, together with the mean, the median, and the standard deviation. The same statistics are reported separately for the guided Task 1 and for the free Task 2, so that the reader can see both how the stage performs in isolation and how the ratings evolve between the first and the second use of the system.

Looking at the absolute scores for Task 1, twelve of the fourteen items sit in the upper half of the scale with means between 4.14 and 4.50 and medians of 4 or 5. The only item that does not reach 4 in the first run is *promptClear* (mean 3.79, median 4, SD 0.97), which captures whether the user felt that the refined prompt matched what they had wanted to express. The lower position of this single item is consistent with the free-text comments collected just after the stage. Some participants explicitly stated in Task 1 that they were not sure what was expected of them when writing the prompt ("I wasn't sure what to put in the prompt and what it expected"), and one participant added that they "found it hard to gauge how much the sketch contributed to the context used for the prompt". The first encounter with the prompt-refinement step, therefore, appears to be the weakest moment of the stage, although still well above the neutral mid-point of the scale.

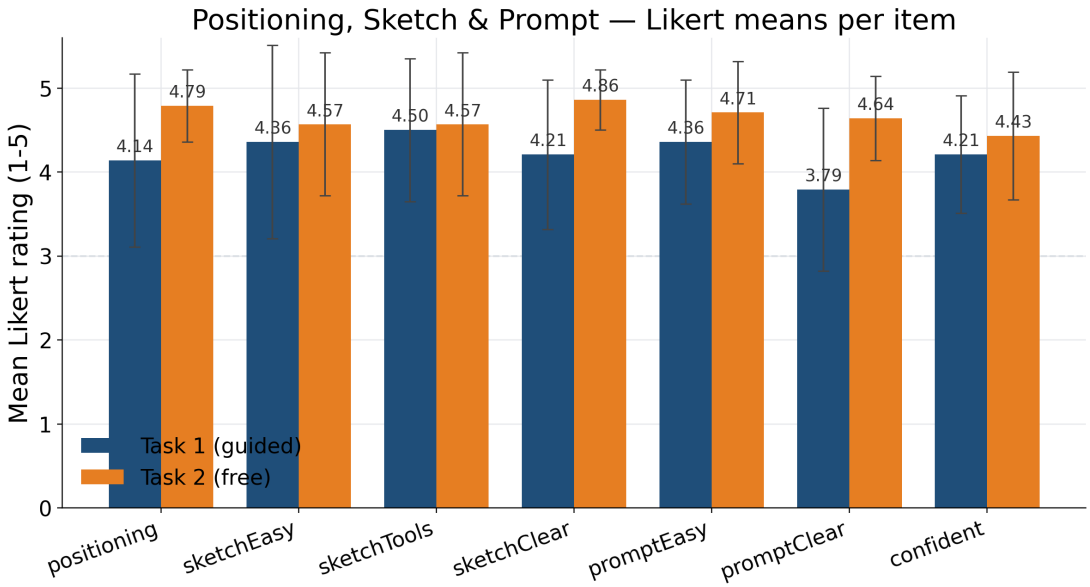

Figure 6.3: Likert ratings for the Positioning, Sketch, and Prompt stage, per task ($n = 14$ per task). Bars show the mean rating on the 1–5 Likert scale, error bars show the standard deviation. The dashed line marks the scale midpoint.

The remaining six items in Task 1 are uniformly positive. The two items that directly evaluate the drawing tools, *sketchEasy* and *sketchTools*, score 4.36 and 4.50, respectively, with nine of the fourteen participants choosing the highest rating in both cases. This is the observed reflection of the design choice described in the system chapter to expose only a single-color magenta brush rather than a full painting interface. The same choice is reflected in *sketchClear* (mean 4.21), which asks whether the drawing communicated what the user wanted: even though the palette is restricted, the participants felt that their intention was unambiguous in the resulting sketch. Two participants nevertheless requested additional colors, one specifically to mark sub-components for animation control and one as part of a broader wish for a more capable drawing tool, including a graphics tablet. The same participants did not penalize the existing tool in their Likert ratings, which suggests that the request is a genuine extension rather than a complaint about the current interface.

The positioning item, which evaluates how easy it is to orient the input mesh in the viewport before drawing, scores 4.14 in Task 1, the second-lowest of the seven items. The associated free-text comments are consistent with this slightly lower number and point to a specific concern about camera control: three participants mentioned, in different words, that they could not zoom in on the region they wanted to edit while keeping the rest of the model in view, or that the available axis-locked translation was not obvious to them (“I could not zoom in on the head”, “it would have been nice if I could only zoom in on the head during alignment”, “it was not clear how I could move vertically or horizontally.”). These are concrete suggestions that can be incorporated into the interface without affecting the rest of the pipeline and that are returned to in the limitations section. We specifically designed it this way to make sure the full image could be fed to the image generation model. The confidence item, which asks the participant how confident they are about the inputs they have just produced before the stage moves on, sits at 4.21 in Task 1 with no rating below 3, indicating that none of the participants left the stage feeling fundamentally uncertain about what they had submitted.

A clear shift is visible when the same items are compared between Task 1 and Task 2. Every one of the seven items improves on the second use of the system, and the items that started lowest improve the most. Positioning rises from 4.14 to 4.79, sketchClear from 4.21 to 4.86, and promptClear from 3.79 to 4.64. Run 2 also has a substantially tighter distribution: with

one exception, no rating falls below 4, and most items have at least nine participants out of fourteen choosing the highest rating. Although the small sample size means that none of these differences should be read as a formal significance result, the consistency of the pattern across all seven items, including the items that already scored well in Task 1, suggests a real learning effect. It should be noted, however, that Task 2 was an open task in which participants chose their own subject, whereas Task 1 used a fixed reference model; the higher scores in Task 2 may therefore also reflect greater ownership and reduced pressure to match a predefined outcome, rather than learning alone.

The first task acts as a short familiarization phase, and once the participant has gone through the stage once, the operations of positioning, sketching, and prompting are perceived as substantially easier and as producing inputs that the participant is more confident about. The prompt-refinement step in particular is rewarded the second time around: the same step that produced the lowest score in Task 1 becomes one of the strongest in Task 2, with one participant remarking after Task 2 that "the reworked prompt gave me the idea that the system understands what I want before it starts generating". The observation that all fourteen participants accepted the refined prompt in both tasks is consistent with this shift: even when the refined prompt is initially confusing, the participants choose it over their own phrasing, and the impression they form of it after the result becomes available is positive.

Taken together, the questionnaire and the free-text comments indicate that the entry point of the pipeline is usable for the target audience. The interface receives positive ratings on every item in both tasks; the only consistent weakness lies in the prompt-refinement step on first encounter, and the concrete suggestions for improvement, additional sketch colors, and finer camera control during positioning are well-defined and do not question the structure of the stage itself.

6.3 Single-Image Generation

As mentioned before, the image generation stage takes the positioned render of the original model, the magenta sketch drawn on top of it, and the refined textual prompt and produces four edited image candidates. The user then picks one of the four candidates as the visual target for the rest of the pipeline, optionally refining it with another sketch and prompt to generate an edited image before moving on. The goal of the evaluation is to verify three properties that the system chapter relies on: that the four candidates are diverse enough to constitute meaningful alternatives, that each candidate stays faithful to the original render outside the user's sketch hull (the high-fidelity claim), and that the participant ends the stage with a candidate that matches the intent expressed by the sketch and prompt.

6.3.1 Empirical Evaluation

Figure 6.4 shows the four generated candidate images produced by the single-image editing step for three example tasks from the user study. For each task, the user selects the most suitable result, indicated by the green border, which then serves as input for the rest of the pipeline. The rightmost column visualizes the per-pixel absolute difference between the chosen candidate and the original unedited render. Bright regions correspond to the edited area (e.g., the goblin head, the rose, the red trunks), while dark regions confirm that the rest of the character's appearance is well preserved. In Example 3, the user was not fully satisfied with any generated option and manually refined option 2 before proceeding. The result after the manual refinement is shown below option 2.

6.3.2 Methodology

Two objective metrics are reported for every trial. The first is the fidelity of the chosen image to the original render, measured as the mean absolute pixel difference between the two images on

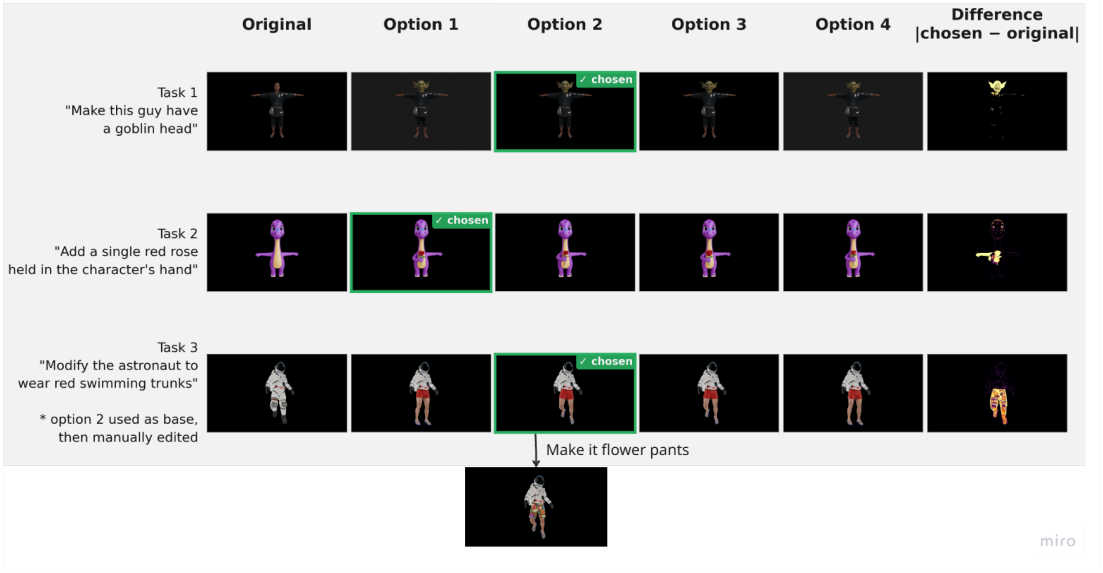

Figure 6.4: Single image generation results for three user study tasks. Each row shows the four generated candidates (chosen option highlighted in green) and the difference map highlighting the edited region.

the unit pixel-value scale. This difference is reported over three regions: the entire foreground of the model, the manually set edit mask (by me), and the foreground outside the edit hull. The third number is the central one for the high-fidelity claim: by construction of the editor, pixels outside the sketch hull should change very little, and we want to confirm that this property holds across the dataset. One might argue that a manually defined mask does not always perfectly capture the user's intended edit region, and this is a fair point. However, given that the generative model's output does not always align precisely with the input sketch, there is no ground-truth edit region that can be derived automatically. The manual mask remains the closest practical approximation.

The second objective metric is the diversity of the four candidates. We report it both as (*1 - mean pairwise cosine similarity*) of CLIP and DINOv2 embeddings of the four candidate images and as a complementary pixel-level diversity computed separately inside and outside the sketch hull. The CLIP and DINO numbers capture semantic diversity ("do the four candidates depict meaningfully different things"), and the pixel-level numbers capture how much physically changes between candidates. Both perspectives are useful because two images with high CLIP similarity can still differ in details that matter for the downstream 3D generation.

The user-facing side of the evaluation uses five Likert items on the same five-point scale used throughout the chapter. Two of them ask the participant whether the chosen image matched what they had expected (*matchExpectation*) and whether the four candidates offered enough variation to choose from (*variation*). One asks whether the user is satisfied with the candidate they ended up selecting (*satisfied*). Two further items, asked only to the participants who actually used the edit tool to edit the images, ask whether the brush was intuitive (*editIntuitive*) and whether it produced the desired result (*editResult*).

6.3.3 Quantitative Results

The fidelity and diversity numbers across the fifteen trials are summarized in Figure 6.5. The fidelity numbers confirm the high-fidelity claim very cleanly. The mean absolute pixel difference outside the sketch hull is 0.025, i.e., about 2.5% on the unit pixel-value scale, with a median of 0.021 and a maximum across all trials of 0.051. The corresponding mean inside the sketch hull is 0.188, an order of magnitude larger, with a median of 0.159. The ratio of outside to inside

fidelity (mean 0.21, median 0.12) makes the same point in a normalized form: the changes the editor makes are concentrated where the user asked for them, and the rest of the image is preserved.

The diversity numbers are more nuanced. The four candidates have a mean CLIP diversity of 0.042 and a mean DINO diversity of 0.050. These numbers are small in absolute terms, but the correct way to read them is per trial. The trials with the largest edit magnitude, measured by the inside-edit fidelity, are consistently the ones with the largest diversity. Conversely, trials in which the requested edit is visually subtle produce candidates that are nearly indistinguishable from one another because there is not much room for the model to differ. Another significant factor is whether the user includes a reference image in the prompt. This can cause the image generation model to over-anchor on the reference and essentially reproduce it rather than generate something new. This is the expected behavior and matches the user feedback discussed below, where the perceived variation in the guided Task 1 (a head replacement based on a reference image, see further) is much lower than in the open-ended Task 2, because there is always a reference image added to the prompt.

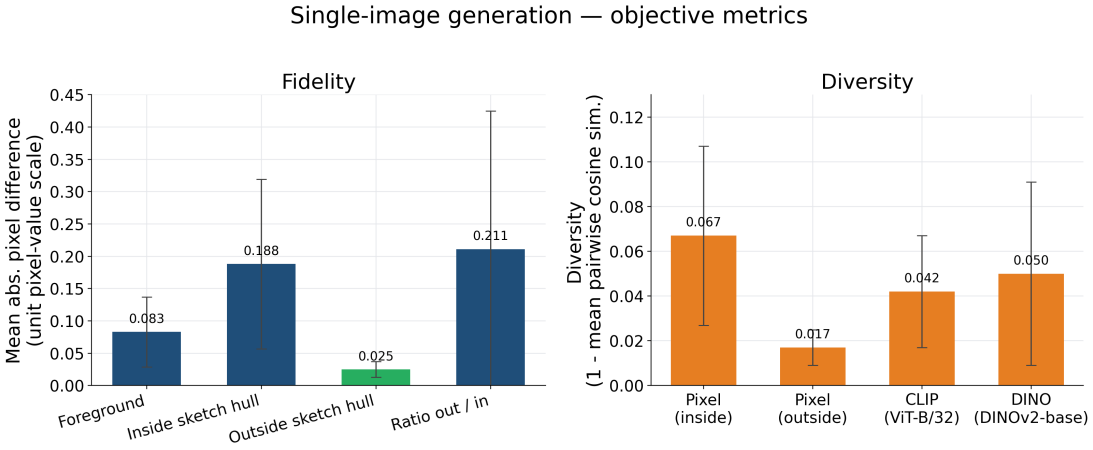

Figure 6.5: Image generation metrics over the fifteen trials. All fidelity numbers are mean absolute pixel differences on the unit pixel-value scale, computed against the original render. Diversity scores are 1 minus the mean pairwise cosine similarity of the four candidates' embeddings.

6.3.4 User Feedback

The five primary Likert items are summarized in Figure 6.6. The satisfaction item is strongly positive on both tasks: pooled, the mean of *satisfied* is 4.79, and twenty-two of twenty-eight selections rate the result at the maximum point of the scale. The *matchExpectation* item is similarly high in both tasks (Task 1 mean 4.86, Task 2 mean 4.36); the slight drop on Task 2 is consistent with the open-ended nature of the free task, where participants invented their own edit without having seen an example of the expected outcome and therefore formed slightly looser expectations.

The most informative pattern in the questionnaire concerns the *variation* item, which behaves very differently between the two tasks. In Task 1, the mean is 1.79, and the median is the lowest possible value of 1: eight participants out of fourteen rated the variation as strongly insufficient. In Task 2, the same item has a mean of 4.29 and a median of 5, with eight of the fourteen participants choosing the highest rating. The reason for this flip is intrinsic to the design of Task 1 and not a property of the image generator on its own. As mentioned before, Task 1 is a head replacement that relies on a reference image of the goblin, which is provided to the model as part of the prompt. With a strong reference image, the model effectively transfers the reference onto the render, and the four candidates end up showing essentially the same

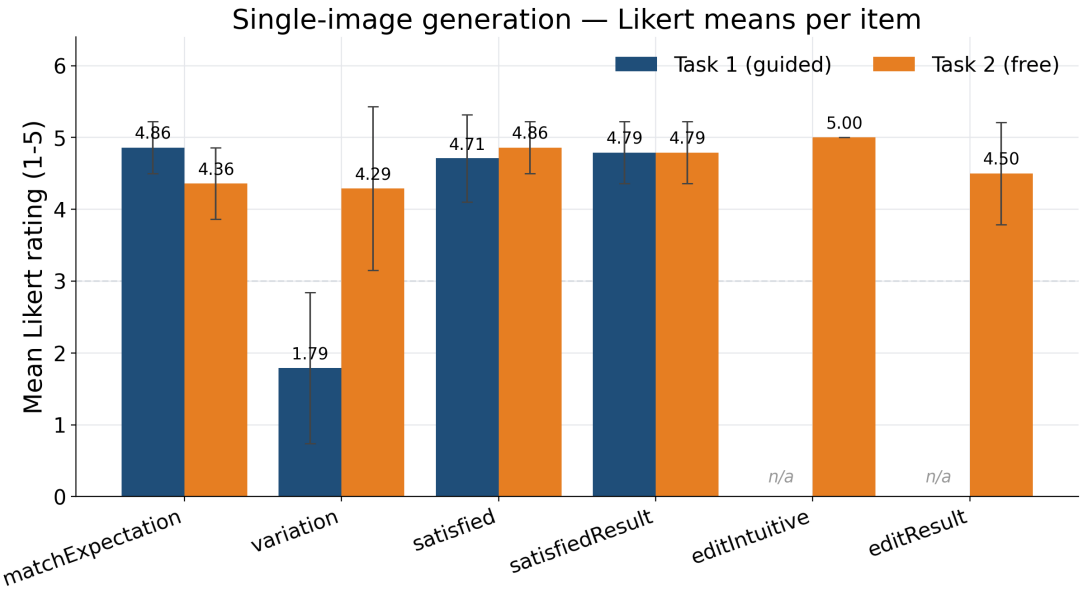

Figure 6.6: Likert ratings for the Single-Image Generation stage, per task ($n = 14$ per task). The two edit-tool items are reported only for the subset of participants who actually used the post-edit brush.

goblin head in four slightly different poses or shadings. The objective CLIP diversity for the goblin trial sits at 0.022, near the lower end of the dataset, and the Likert ratings track the objective number. In Task 2, users do not have to include a reference image, only a textual prompt and a sketch, which gives the model room to explore different interpretations of the same description. Two participants added a reference image, since it was allowed. One of them scored a low diversity rate because of this (CLIP:0.014,DINO:0.018), the other did not, since the prompt still allowed some variation (CLIP:0.061,DINO:0.041). The mean CLIP diversity across the Task 2 trials is comparable to the dataset mean (0.042), and the perceived variation rises correspondingly. Their perceived-variation scores mirror the objective numbers almost exactly: the participant on the low-diversity trial rated the variation as 1 out of 5, the lowest possible value, while the participant on the higher-diversity trial rated it as 5, the highest. For comparison, the Task 2 mean across all fourteen participants is 4.29 with a median of 5, so the first of the two participants is the only clear outlier on the low side, and the second sits at the top of the distribution alongside the rest.

We therefore interpret the distribution of *variation* as evidence that the image generator behaves as expected in both situations: when given a strong reference, it sticks to it; when given only a textual description, it explores. From the user's point of view, the actual outcome in Task 1 is not a problem, because they prefer the closer-to-reference candidates anyway; the low variation score is a faithful report of what the four candidates look like rather than a complaint about the system.

The post-generation image editing, which lets the participant re-generate a candidate image using a sketch and prompt before moving on, was used by none of the fourteen participants on Task 1 (consistent with the pre-generated nature of that task) and by two of the fourteen participants on Task 2. The sample is too small to support a statistical claim, but both participants who did use the tool rated it as fully intuitive (*editIntuitive* = 5 for both) and as producing a satisfying result (*editResult* = 4 and 5). The fact that twelve of fourteen Task 2 participants chose not to use the tool is more useful evidence than the two ratings themselves: it indicates that the four candidates were, on the open-ended task, close enough to the participant's intent that an extra brush pass was not perceived as necessary. It should be noted that the availability of the editing tool was communicated to participants, so the

low selection may not directly be attributed to a lack of awareness. It may instead reflect a combination of satisfaction with the generated candidates, uncertainty about what specific change to request, or simply not wanting to invest additional effort once a sufficiently close result was already on screen. This is also where the free-text remarks for the stage are concentrated. The remarks from Task 2 are overwhelmingly positive ("Much better than I expected", "Very nice"), with two participants noting a detail they would have changed had they pushed further ("I expected the mouth to be open", "I was thinking more about a cape than a suit, but it looks better than I thought"). Neither one of these participants used the image editing functionality, implying that they were okay with this detail not being in the proposed edit.

In summary, the image generation stage produces candidates whose fidelity outside the sketched edit region is uniformly very high, whose diversity scales with the magnitude of the requested edit, and which the participants accept without modification in the vast majority of cases. The one genuine concern surfaced by the evaluation is the low perceived variation on the reference-image Task 1, which we read as a faithful reflection of how the model behaves with a strong reference rather than as a problem with the system.

6.4 Multiview Generation

After the user has chosen a single edited image, the pipeline asks the same image model to extend that single view into a four-view multiview, organized as a 2×2 grid containing the front, right, left, and back panels. The multiview is the actual input to the image-to-3D generator: any inconsistency between the four panels at this point could potentially translate directly into errors in the generated 3D model. The goal of this section is therefore to quantify how faithful each of the generated views is to the corresponding view of the original (pre-edit) model. The study application does not include dedicated Likert items for this stage, since the participant sis not see the multiview because adding a verification step at this point would introduce an additional decision moment without a clear corrective action available to the user, and because exposing an intermediate technical artefact risks creating confusion rather than adding meaningful control. It could be added to add a fallback, but we did not think this was explicitly needed.

6.4.1 Empirical Evaluation

Figure 6.7 shows the original multiview and the generated output multiview side by side for three trials from the user study, together with the per-pixel absolute difference between the two. The original multiview is rendered directly from the unedited 3D model; the output multiview is produced by the image model conditioned on the chosen single edited image from the previous stage.

In all three trials, the edit is clearly present in every panel of the output grid: the goblin head, the rose, and the red trunks are visible from the front, right, left, and back views, confirming that the image model extrapolates the single-view edit coherently to all four viewpoints. At the same time, the difference maps show that the unedited parts of the character, body shape, clothing, and proportions remain close to the original across all four panels, with bright regions of the heatmap concentrated on the edited area. The four views are visually consistent with one another, which is the critical property for the downstream image-to-3D step.

6.4.2 Methodology

The metrics are computed from the same two 2×2 images: the multiview generated by the edited image and the multiview generated by the original render. Both grids use the same layout (front, right, left, back) and the same camera parameters, so the two grids can be compared panel-by-panel.

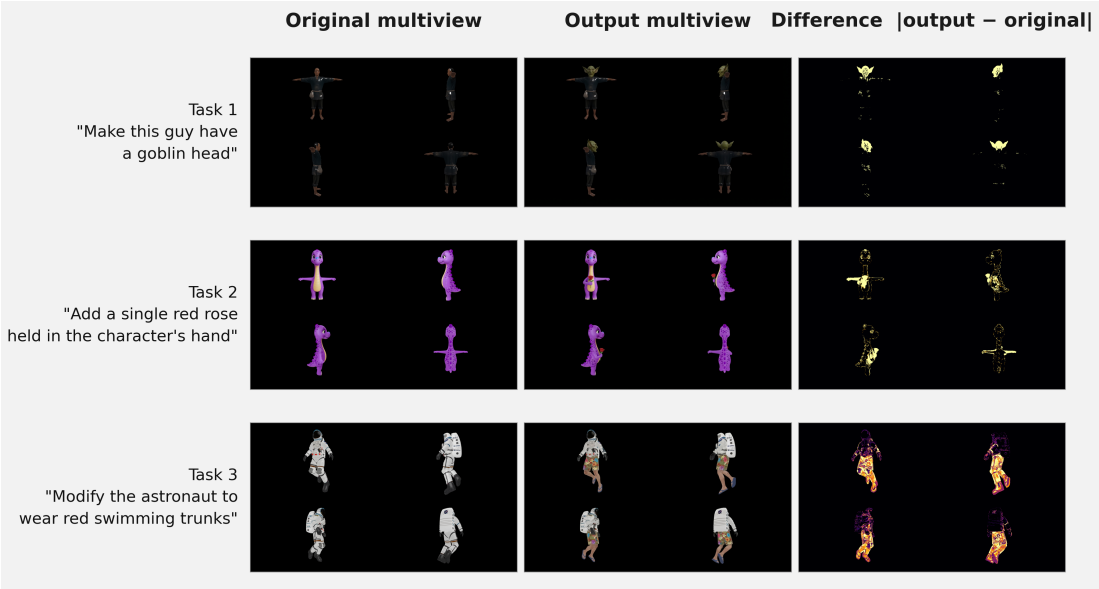

Figure 6.7: Multiview generation results for three user-study trials. Columns left to right: original multiview rendered from the unedited model, output multiview generated from the chosen edited image, and the per-pixel absolute difference between the two (inferno color map, normalized per trial). Each grid shows four views: front (top-left), right (top-right), left (bottom-left), and back (bottom-right).

For each of the four panels, we compute the mean absolute pixel difference between the edited panel and the corresponding panel of the original multiview, restricted to the foreground of the original view to remove background pixels. We further split that per-view L1 number into the part that falls inside the manually set edit mask and the part that falls outside it. The inside L1 indicates how strongly the edit changes the affected region, and the outside L1 indicates how much of the rest of the model leaks into the difference. We report the inside and outside means averaged across the four views, together with the ratio of the two.

6.4.3 Quantitative Results

The per-view L1 fidelity numbers, summarized in Figure 6.8, tell a clear story. Inside the manually set edit mask, the mean L1 across all views averages 0.158 on the unit pixel-value scale, with a median of 0.131 and a standard deviation of 0.105, which is the same order of magnitude as the in-edit fidelity in the single-image section and again indicates that the requested edit is visible in every view. Outside the edit mask, the mean L1 is 0.038 with a median of 0.030, an order of magnitude smaller. The ratio of the two has a mean of 0.28 and a median of 0.18, meaning that in a typical trial, the regions of the model that the user did not touch differ between the original and edited multiview by less than a fifth as much as the regions the user did touch.

The implications for downstream stages are limited. The L1 numbers indicate that the edits stay concentrated where the user asked for them, while the rest of the model is largely preserved across views. The remaining inconsistency that does propagate further is discussed in the next section, where the dominant failure mode is not driven by the multiview itself but by the limited fidelity of the image-to-3D step.

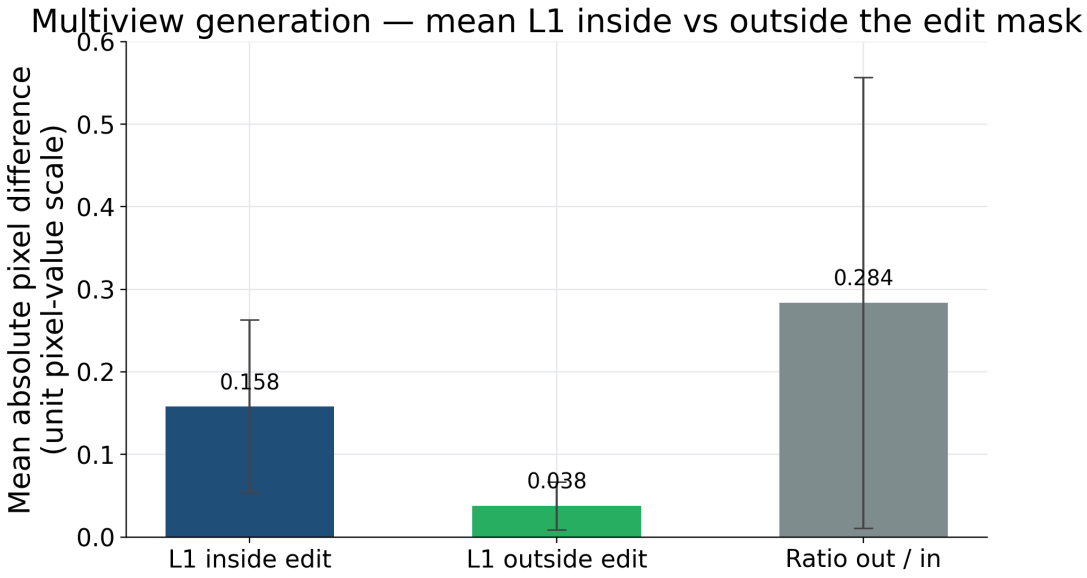

Figure 6.8: Multiview metrics over the fifteen trials. The L1 metrics are mean absolute pixel differences against the original multiview, restricted to the foreground and split by the painted edit mask.

6.5 Generated 3D Model

After the multiview has been produced, the pipeline hands it to the image-to-3D step (the Meshy service), which returns a textured 3D mesh. This mesh is the input to all of the later stages of the pipeline: the alignment step has to register it against the original, the edit-mask step uses it to figure out which vertices were affected, and the selective remeshing step has to merge it into the original mesh. The quality of the generated 3D model, therefore, upper-bounds what the rest of the pipeline can achieve, and the goal of this section is to characterize that quality. Because the questionnaire items for this stage were only present for the very first participant and were dropped in the protocol used by the rest of the study, this section is reported quantitatively only. Hence, the reason this thesis is made, after the generation of a 3D model using the original images as reference, it loses most of the original geometry and explicit features.

6.5.1 Empirical Evaluation

Figure 6.9 shows untextured front-view renders of the original and edited 3D models for three trials from the user study. Removing the texture isolates the geometry and makes shape differences easier to assess visually.

The goblin-head replacement (Task 1) shows a clear change in the head region: the rounded human head of the original has been replaced by a more angular, protruding goblin skull, while the body geometry below the neck is well preserved. The rose trial (Task 2) shows a new appendage extending from the hand area of the character, corresponding to the stem and bloom of the rose, with the rest of the body being changed: a different head, different body proportions, and added nipples. The astronaut trial (Task 3) shows a visible change around the lower torso where the suit geometry has been reshaped to match the swimming trunks; the rest of the model is not well preserved, giving him a different mask, chest-mount equipment, and arm material. The remeshing step, described in a later section, can solve this issue.

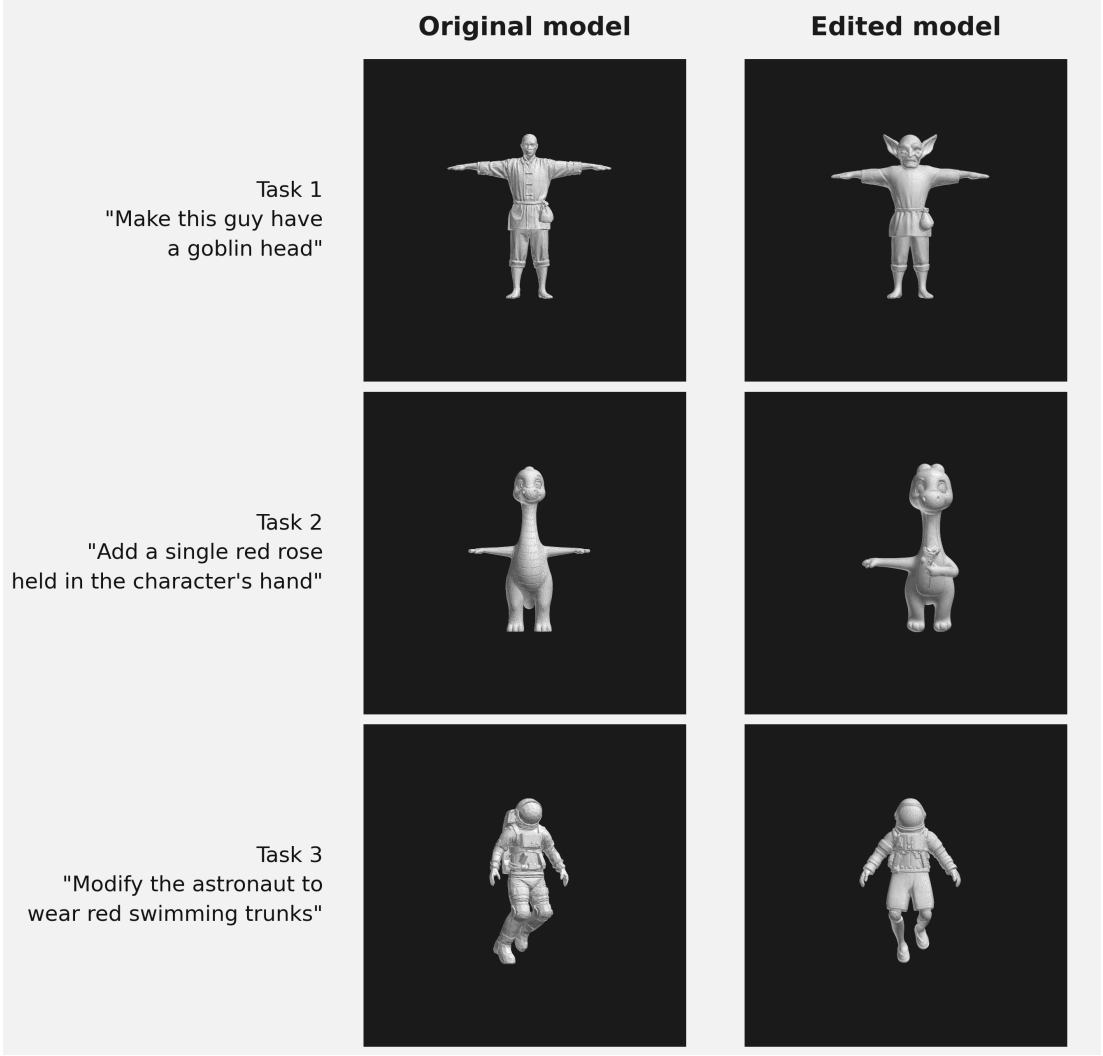

Figure 6.9: Untextured front-view renders of the original (left) and edited (right) 3D models for three user-study trials. The grey-on-black rendering removes texture information so that geometric changes are directly visible.

6.5.2 Methodology

We measure how well the generated mesh reproduces the original geometry outside the edit region, which is the property that the rest of the pipeline depends on. The original mesh and the expert-aligned generated mesh are both filtered to keep only the faces that lie outside the expert edit mask. From every trial, we then render thirty-six binary silhouettes per mesh: the cameras are placed evenly on the unit sphere using Fibonacci sampling, the projection is orthographic, and the image resolution is 256×256. For each viewpoint, we measure the intersection-over-union (IoU) and the per-pixel L1 difference between the two silhouettes, and we aggregate over the 36 views into a per-trial mean.

6.5.3 Quantitative Results

Across the fifteen trials, the mean silhouette IoU is 0.711 (median 0.737, standard deviation 0.132) and the mean per-pixel L1 difference is 3.4% of the image area (median 2.7%, standard deviation 2.1%). In a comparison where two perfectly aligned identical meshes would yield IoU 1 and L1 0%, these numbers indicate that the generated mesh broadly recovers the silhouette of the original outside the edit region but is not pixel-accurate. This can be seen in Figure6.10

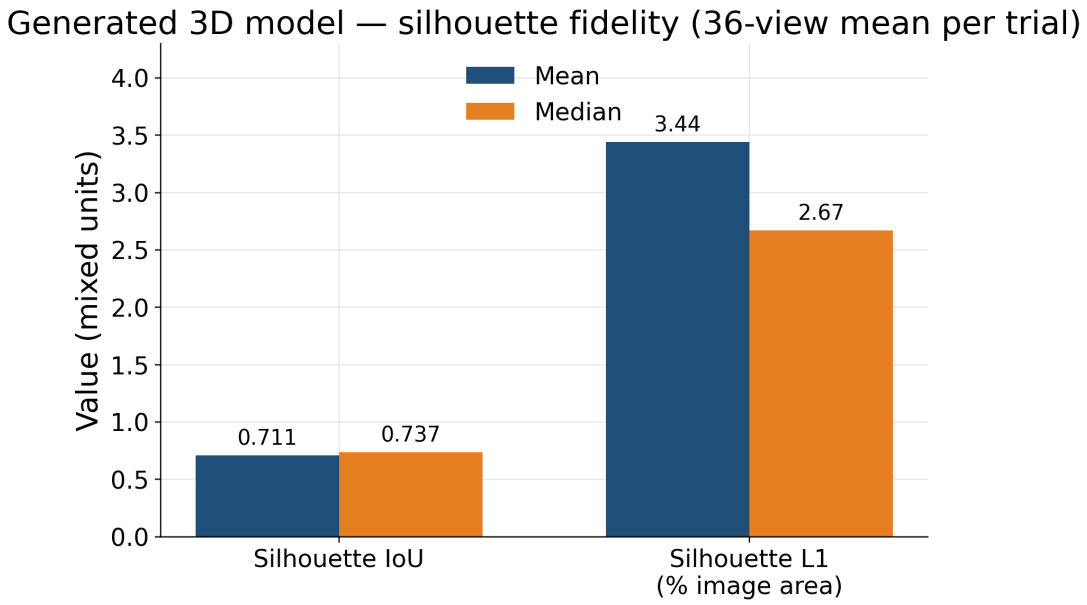

Figure 6.10: Per-trial silhouette metrics, averaged across the 36 Fibonacci-sphere views. IoU is the intersection-over-union of the binary silhouettes of the original mesh and the expert-aligned generated mesh, restricted to faces outside the edit mask. L1 is the per-pixel mean absolute difference of the same silhouettes (lower is better).

These numbers are nonetheless difficult to interpret in isolation, as the metrics only become meaningful when compared against a reference point or baseline. More importantly, they reflect the core motivation behind this thesis: the observation that generated 3D models do not faithfully reproduce the input images they are derived from. As is clearly visible in the examples shown in Figure 6.11, the models may appear plausible and acceptable for general-purpose use, but they fall short when detail preservation is the goal. Surface features, fine geometry, and characteristic proportions present in the reference image are often lost entirely or replaced by another structure in the generated mesh.

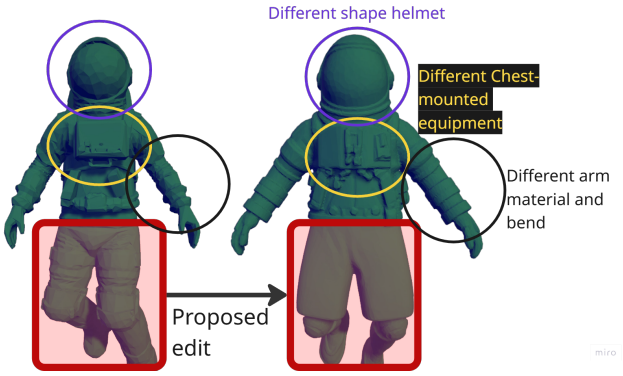

Figure 6.11: Example of the shape difference between the original and generated 3D model.

6.6 Alignment

After the image-to-3D step has produced a new mesh, that mesh has to be aligned with the original mesh to later merge the two together. The pipeline does this twice, with two different strategies, and asks the user to choose the result that looks best. As mentioned before, the first strategy uses a feature-based registration with ICP. The second strategy uses a silhouette and a bounding-box-based fit that aligns the overall shape from the unrotated front view. The reason for offering both is that the two methods fail on different kinds of inputs, and exposing the choice to the user is the simplest way to make the pipeline robust without forcing a single algorithm to handle every case. This section evaluates the two methods and is the first part of the chapter where the two methods are directly compared.

6.6.1 Empirical Evaluation

Figure 6.12 shows front and left-side renders of the original model (blue) overlaid with the edited mesh under each alignment condition for three representative trials. In the unaligned column, the edited mesh is visibly offset or rotated relative to the original, confirming that the raw image-to-3D output does not share the same coordinate frame. Method 1 (ICP) and Method 2 (silhouette fit) both bring the edited mesh into rough correspondence with the original, but to different degrees. In all three trials, the participants selected Method 1, which can be seen to produce a tighter overlap of the shared body parts across both the front and left views. The left-view renders are particularly informative because they expose depth alignment: a method that looks well-aligned from the front may still have a depth offset that becomes visible from the side.

6.6.2 Methodology

For every trial, both methods produce an aligned mesh. We compare each aligned mesh to two references. The first reference is the original input mesh: how close are the vertices of the aligned generated mesh to the surface of the original? We report this as the mean per-vertex Euclidean distance from the aligned mesh to its nearest neighbor on the original, restricted to the vertices that lie outside the edit region, since those are the vertices that should be on the original surface after a perfect alignment. The second reference is a manually aligned version (by me) of the generated mesh, prepared by hand for every trial. The expert alignment is, by construction, the best alignment we know how to achieve; it acts as a “perfect” reference. Comparing each method to this manually set version isolates the quality of the alignment algorithm from the noise introduced by the image-to-3D step: if the generated mesh has, say, the wrong proportions on a particular part, no automatic alignment can fix that, and the algorithm-to-expert distance gives a clean view of how well each algorithm does on the part it can actually control.

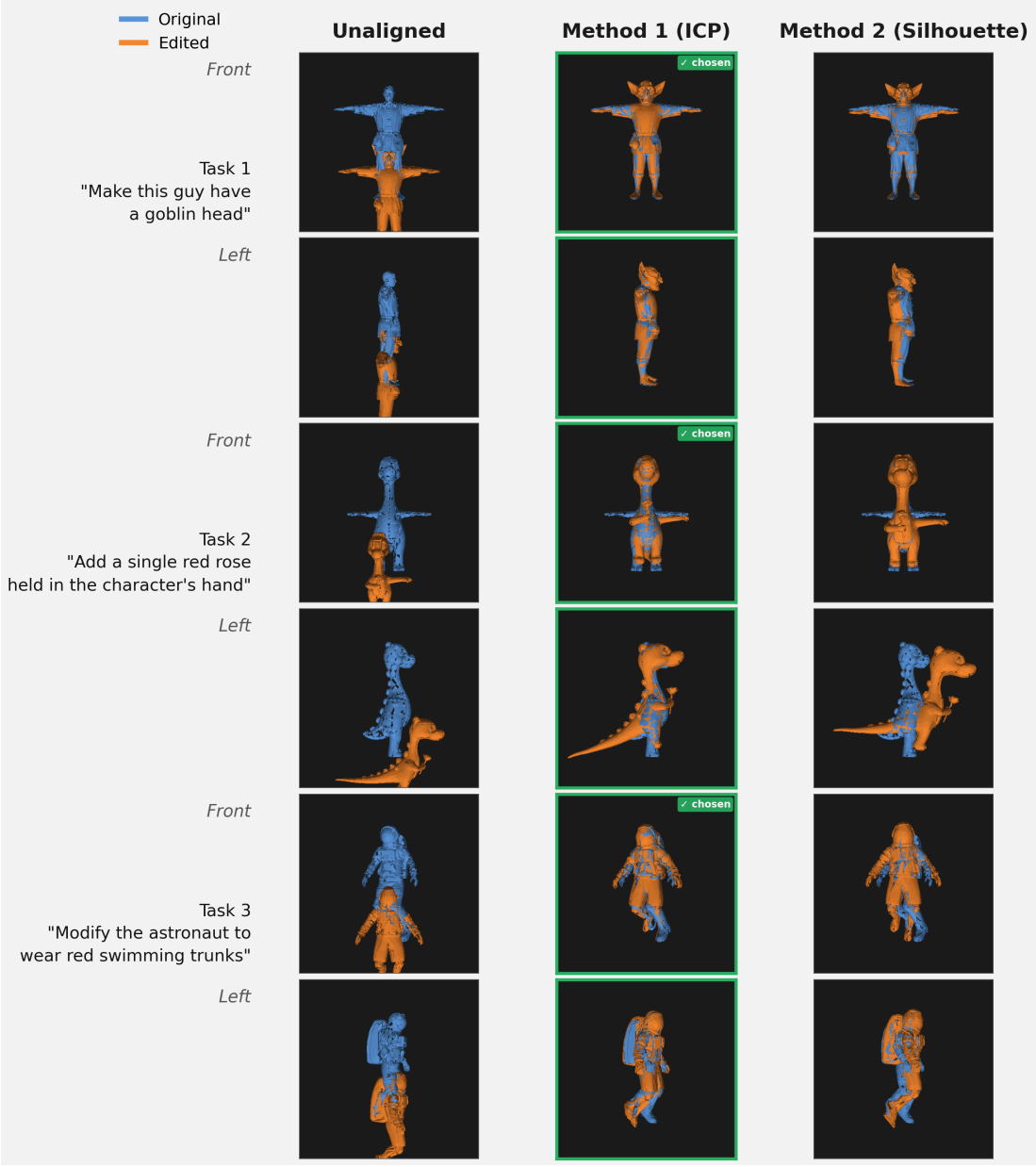

Figure 6.12: Alignment results for three user-study trials. Each cell shows the original mesh (blue) overlaid with the edited mesh (orange) after the indicated alignment step. Rows alternate between front and left-side views. The chosen alignment (Method 1, ICP) is highlighted with a green border.

On the user side, we record three things for each trial: which of the two methods the participant chose, three Likert-style items that capture how the participant judged the alignment, and any free-text remarks. The first item asks whether the chosen alignment fits the original model well, the second asks how confident the participant is in their choice between the two methods, and the third is a slider from 0 to 100 that asks the participant to give a global quality score to the result they picked. We use a binomial test as a basic sanity check on the method-choice distribution, but, as discussed in Section 6.1.6, the discussion is driven by the observed proportion and the free-text remarks rather than by a single p-value.

6.6.3 Quantitative Results

The fourteen trials and their four distance numbers are listed in Figure 6.13. Averaged across all trials, Method 1 achieves a mean distance of 0.092 to the original mesh (median 0.045, standard deviation 0.150); Method 2 achieves a mean of 0.106 (median 0.058, standard deviation 0.096). Against the expert reference, Method 1 has a mean distance of 0.070 (median 0.031, standard deviation 0.102) and Method 2 a mean of 0.080 (median 0.056, standard deviation 0.069). Two things stand out. First, Method 1 is on average closer to the original in eleven of fourteen trials and closer to the expert in twelve of fourteen, which is consistent with the user-side preference reported below. Second, on the trials where the two methods disagree, the gap can be very large in either direction. One trial is the clearest example: Method 1 ends up at a distance 0.616 from the original, which is more than twenty times its overall mean, while Method 2 sits at 0.269 on the same trial. Another trial shows the same kind of disagreement in a smaller form (0.118 for Method 1 against 0.041 for Method 2). On both of these trials, the user chose Method 2, illustrating that the two methods genuinely cover different failure modes and that letting the user pick is the simplest way to keep the pipeline robust.

In these cases, we can see that Method 2 stays consistent when Method 1 fails. This is due to the limitation of Method 1, mentioned in Section5.2.2 where it performs poorly when the proposed edit is too “large”.

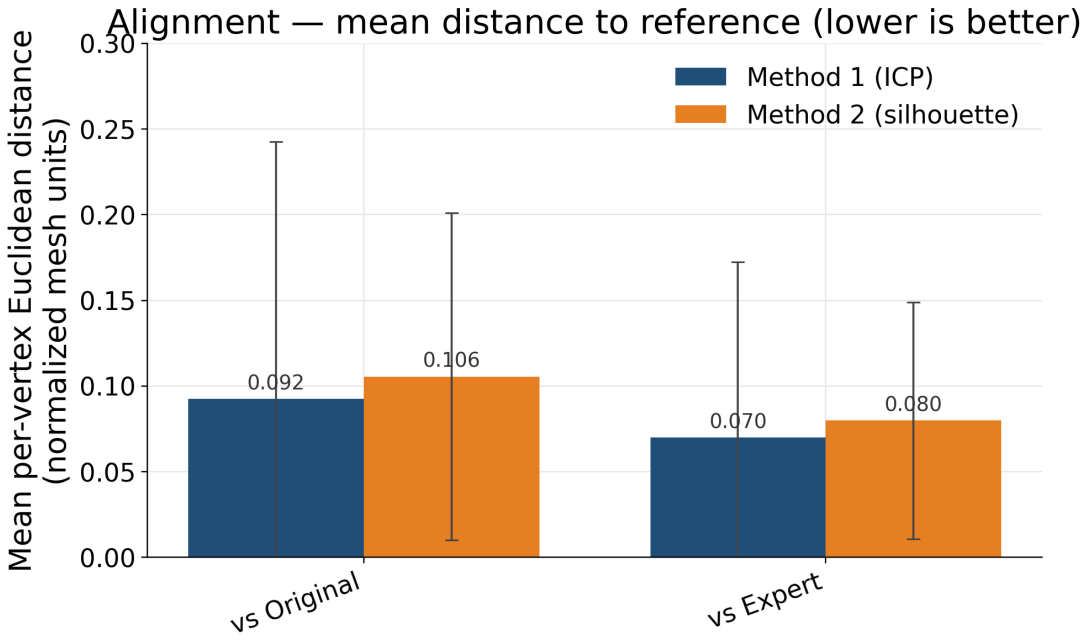

Figure 6.13: Aggregate alignment distances. Method 1 (ICP) and Method 2 (silhouette-fit) were compared against the original mesh and against a manually-aligned (by me) version of the generated mesh. Distances are the mean per-vertex Euclidean distance in the units of the normalized meshes; error bars show one standard deviation.

6.6.4 User Feedback

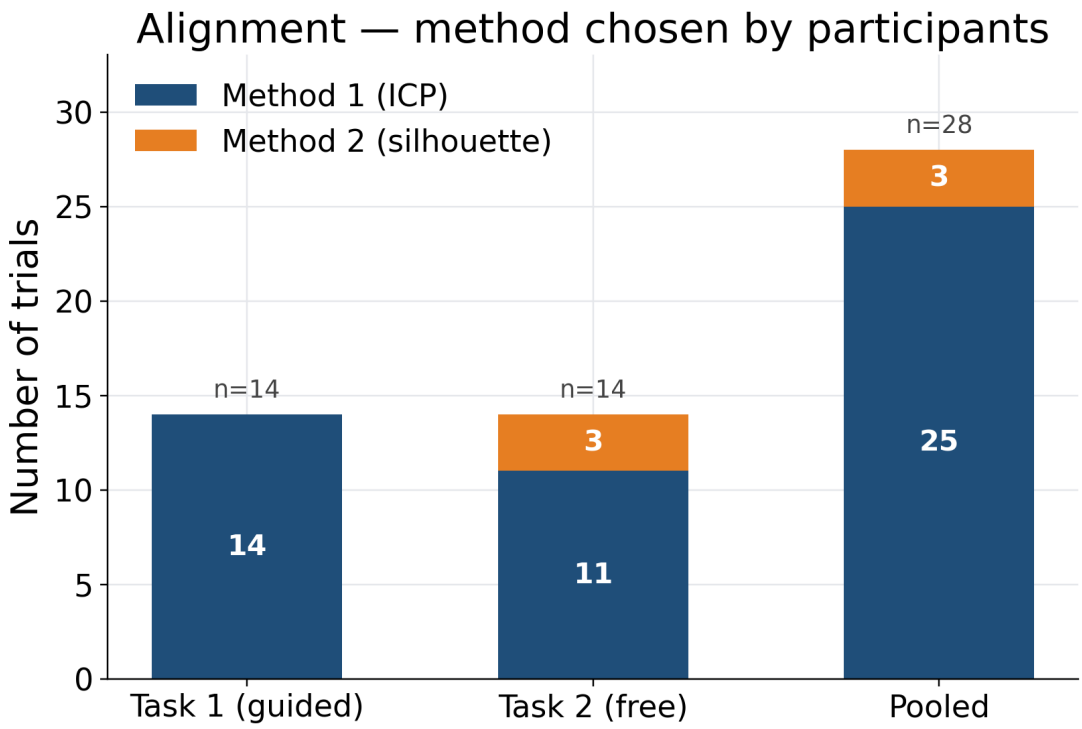

Figure 6.14: Number of participants who selected each alignment method, per task and pooled.

The user-side method choice is summarized in Figure 6.14. In Task 1, all fourteen participants chose Method 1. Task 1 is a guided task built on a pre-generated edit, so this is essentially fourteen votes on the same alignment problem, on which Method 1 happens to be the better result. In Task 2, where each participant works on an independent edit, the distribution becomes eleven for Method 1 and three for Method 2. A binomial test on these fourteen independent trials against the null hypothesis that the two methods are picked with equal probability returns $p \approx 0.029$, which is consistent with a real preference for Method 1 rather than a chance outcome, although we treat the number as supporting the qualitative picture and not as a confirmatory test. The fact that Method 2 still receives three independent selections in Task 2 is the important part of the result: it confirms that the two methods complement each other and that removing the choice would force a small but real fraction of the trials onto a worse alignment.

The three Likert-style items are summarized in Figure 6.15. In Task 1, all three items are very high: *wellAligned* sits at a mean of 4.71 and a median of 5, *confidentChoice* at 4.86, and the quality slider at 92 on average. None of the fourteen participants rated any Task 1 item below 4. In Task 2, every item drops a little: *wellAligned* goes to 4.0 with a wider spread, *confidentChoice* to 4.29, and the quality slider to 83 on average. The drop is consistent with what we expect, since it is very prone to outliers when the generated mesh does not resemble the reference images "perfectly". The confidence item still sits well above the midpoint of the scale in Task 2, indicating that participants are confident in the choice they made, even when they are less satisfied with the underlying quality.

The free-text remarks left in Task 2 fall into two groups. Two participants commented that the two alignments looked the same: "They looked the same" and "both were the same". These remarks are, in fact, a positive signal for the stage, since they describe trials on which both methods produced a good fit, and the user therefore could not tell them apart. Two further participants noted that the generated mesh did not fully match the original ("The models

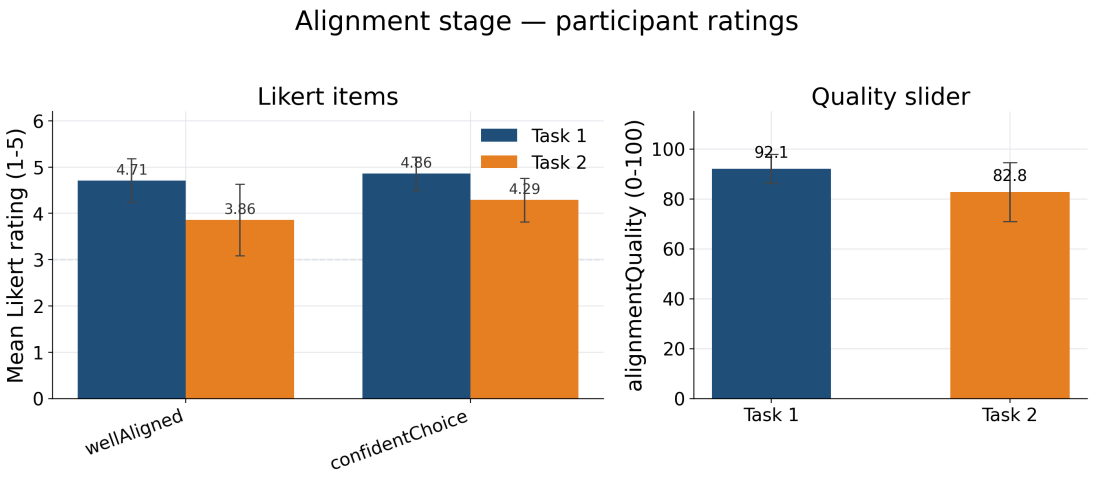

Figure 6.15: Likert ratings for the Alignment stage, per task ($n = 14$ per task). The first two items use the standard 1–5 Likert scale; *alignmentQuality* is a slider from 0 to 100.

differed quite a lot, but the alignment was okay in the places where it was needed", "The models did not completely match"). These remarks describe the same phenomenon as what probably drives the lower Task 2 Likert scores: the alignment is not the problem, but the mesh itself is, and the user correctly identifies that the limit lies in the generated mesh itself rather than in the alignment step. In neither case does the remark suggest that the stage is hard to use, and the confidence item supports this reading. It is also important to note that this is not a deciding factor in the pipeline, since users can always manually change the alignment to their wishes.

No participant made use of this manual adjustment option, which may reflect satisfaction with the automatic alignment, but may equally reflect the cognitive difficulty of repositioning a 3D object in a three-dimensional scene: adjusting position, rotation, and scale simultaneously requires a degree of spatial reasoning that, as observed later in the edit-mask stage, tends to be more demanding than it appears, we observe a lot of participants often reasoning in two dimensions while operating in three.

6.7 Edit-Mask Generation

The edit-mask stage decides which vertices of the original mesh will be replaced by geometry from the generated mesh and which will be left untouched. The decision is made automatically using three independent cues (geometric difference, image-difference voting, and multiview voting) combined by a logical OR, followed by a clustering step that removes isolated noise, all described in the system chapter5.3. The user is then shown the resulting mask and can edit it before continuing to the merge. The goal of this section is to evaluate two things: how close the automatic mask is to a ground-truth mask placed by me, and how much the user has to do on top of the automatic mask before they are satisfied with the result.

6.7.1 Empirical Evaluation

Figure 6.16 shows the per-cue masks and the algorithm output for the astronaut trial, chosen because the intended edit (red swimming trunks replacing the lower body) is spatially well-defined and easy to judge visually.

The three cues vary substantially in how broadly they cast the net. Cue A (geometric) is the most conservative: at only 9.1% of vertices, it marks almost exclusively the torso and

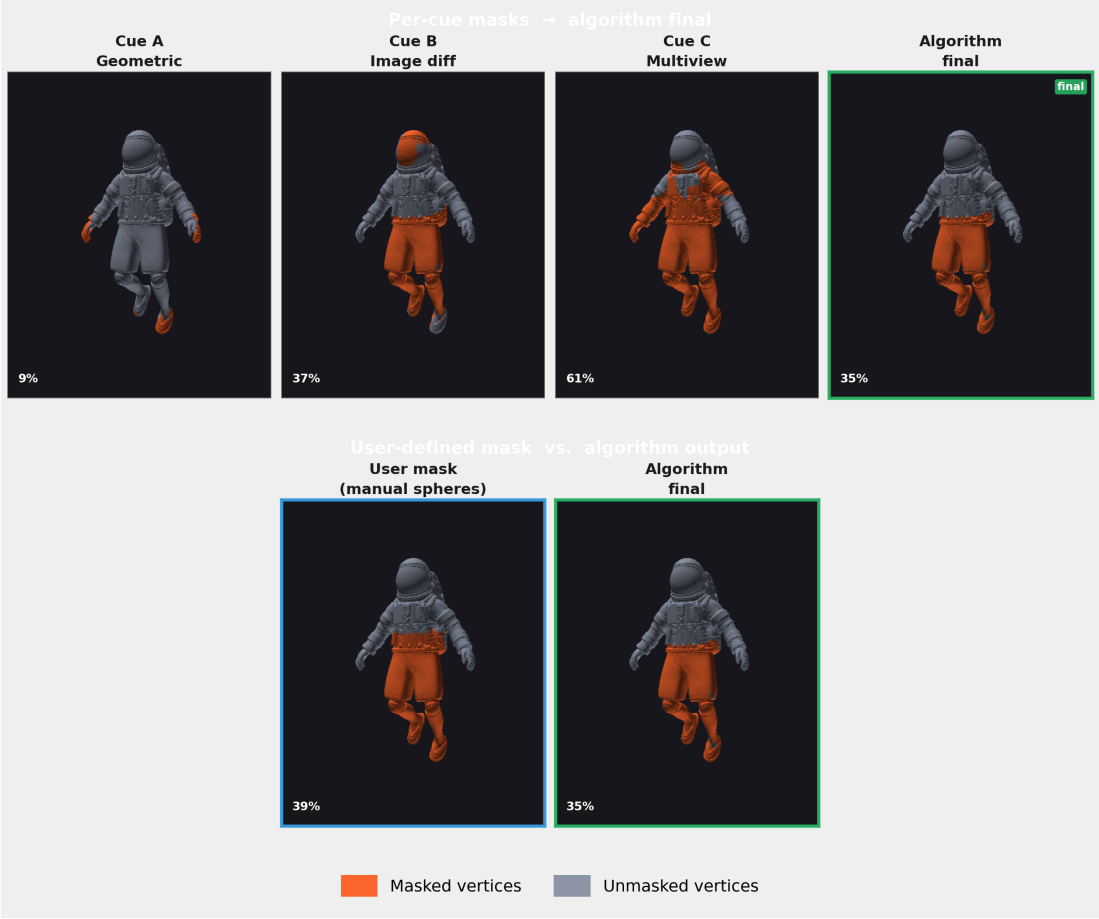

Figure 6.16: Per-cue edit masks and final algorithm output for the *modify_astronaut* trial. **Top row**: the three individual cues and the clustered output. **Bottom row**: the mask produced by the algorithm (right) alongside the mask drawn manually by the user (left). Orange vertices are classified as part of the edit; grey vertices are unaffected. Percentages indicate the fraction of the total mesh vertices that each mask covers.

upper legs, but it misses large parts of the intended region because geometric differences caused by alignment error bleed into non-edit areas while remaining small over parts of the edit itself. Cue B (image-difference) is more complete at 36.6%, covering the trunk and thighs convincingly. Cue C (multiview voting) is the broadest of the three, selecting 61.4% of all vertices, far more than necessary.

The clustering step following the logical OR of the three cues addresses this directly. By requiring every selected vertex to belong to a sufficiently large connected component, it removes the scattered false positives and contracts the mask to 35.0% of vertices, concentrating it on the trunk and legs where the edit should be. The visual improvement from Cue C to the algorithm output is clear in the figure: the stray selection on the shoulders and arms disappears, and the mask becomes spatially coherent.

The user mask (bottom-left, 38.9%) was drawn after the participant had seen and accepted the algorithm output. It is slightly larger than the algorithm's 35.0%, reflecting a deliberate expansion: the participant added a small number of spheres to extend the selection downward along the legs and to pull in the feet vertices that the algorithm had trimmed away and upwards, which presumably the user judged necessary to achieve a clean boundary. This conservative expansion, of roughly four percentage points, is representative of the typical modification pattern observed across trials: rather than correcting fundamentally wrong regions, participants

fine-tuned the boundary of an already reasonable mask.

6.7.2 Methodology

The evaluation of the automatic mask is split into two complementary parts. The first part is an ablation that asks how much each individual cue contributes to the final algorithmic mask: the per-cue masks (geometric, image-difference, and multiview voting) and their combined-OR vote are each compared, at the vertex level, against the final mask produced by the post-voting clustering step. Because the clustering step never adds vertices, the union of the three cues bounds the final mask from above, and the comparison is a clean way to see how much each cue agrees with what survives clustering. We report precision, recall, and intersection-over-union (IoU) per cue.

The second part captures the quality of the algorithmic mask from the user's point of view. For every trial, the participant either accepts or modifies the algorithmic mask before continuing. We rasterize that user mask on the same vertex set as the algorithmic mask and report the same three set-based metrics (precision, recall, IoU) of the algorithmic final mask against the user mask. In this comparison, a high precision means that everything the algorithm picked was kept by the user, and the recall measures how much the user ultimately added on top of what the algorithm produced.

For each trial, participants answered four rating questions after seeing the automatic mask: whether the mask was placed in the right region of the model, whether it was clear how to modify it, whether it included every vertex that should have been part of the edit and whether it excluded everything that shouldn't have been. These last two questions are essentially the user-facing versions of recall and precision, which are discussed further later on. Participants also indicated with a simple yes or no whether they had manually edited the automatic mask before moving on. Those who did edit it answered two additional rating questions about how intuitive the editing process felt and whether they were happy with the result. At the end of the stage, participants could also leave any free-text comments they had.

6.7.3 Quantitative Results

Figure 6.17 reports the per-cue ablation on the fifteen trials. The geometric cue on its own (A) is the weakest of the three: its precision is moderate at 0.47, but its recall is only 0.32, so the geometric cue alone identifies roughly a third of the vertices that the final mask ends up keeping. The image-difference cue (B) recovers nearly all of the final mask (recall 0.89) at a precision of 0.69, and the multiview cue (C) is the strongest single cue, recovering almost the entire final mask (recall 0.98) at a precision of 0.66. The combined OR vote covers the final mask by construction (recall 1.00), at a precision of 0.53, which means roughly half of the OR-vote vertices are trimmed away by the clustering step.

Read against the system chapter, the ablation supports the choice to combine the three cues with an OR and then filter. The multiview cue carries most of the signal but is not perfect on its own. The image-difference cue is the only one that brings a useful precision boost in trials where the multiview cue is contaminated, and the geometric cue mainly serves as a safety net on trials where both image-based cues miss part of the edit. The clustering step then removes the false positives that the OR introduces, and the trade-off in the final mask is the high-precision / moderate-recall mask shown in the next subsection.

To complement the ablation, we compare the final algorithmic mask against the mask the participant ended up using in the rest of the pipeline. The participant either accepts the algorithmic mask as is, removes spheres they consider extraneous, or adds spheres the algorithm missed. The resulting sphere set is the ground truth from the user's point of view, and the comparison quantifies how much manual work the algorithm leaves to the user. In this case, we only use the ten trails where users did manually make edits.

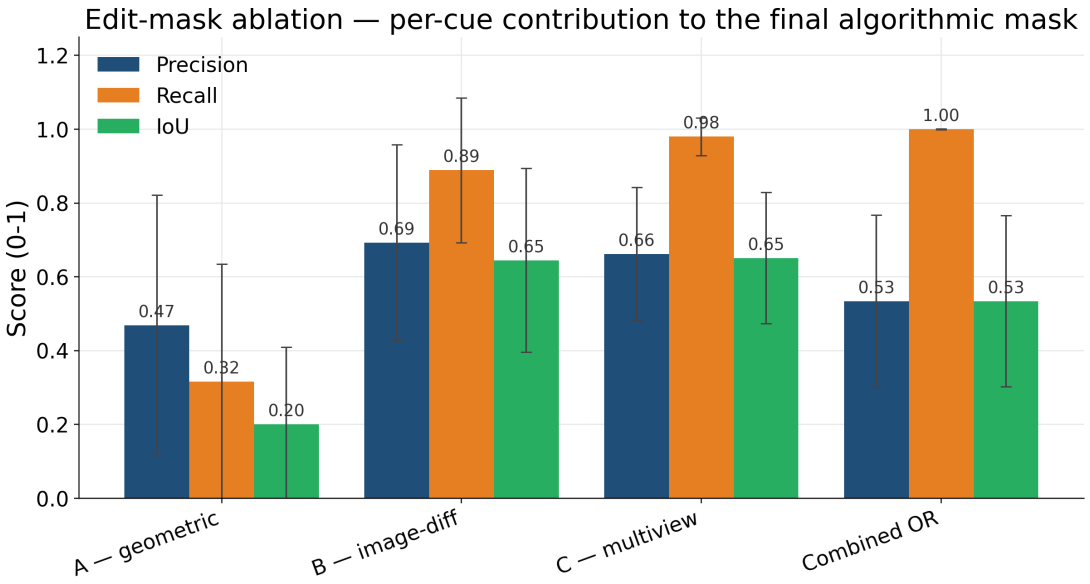

Figure 6.17: Per-cue ablation of the automatic edit-mask on the fifteen trials, evaluated against the final mask produced by the clustering step. Bars show the mean precision, recall, and IoU per cue, with error bars representing one standard deviation across trials.

Across the ten trials where the user changed the automatically generated edit mask, the algorithmic mask reaches a precision of 0.93 on average (median 0.99, standard deviation 0.14), a recall of 0.69 on average (median 0.69, standard deviation 0.18), and an IoU of 0.66 on average (median 0.69, standard deviation 0.19).

The two numbers carry very different meanings. The high precision indicates that the algorithmic mask almost never proposes vertices that the user later removes: the median precision is 0.991 and six of the ten trials reach a perfect precision of 1.0. In other words, when the algorithm selects a sphere, that sphere is almost always kept. The moderate recall, on the other hand, tells the rest of the story: the user routinely adds spheres beyond what the algorithm produced, recovering parts of the edit that the cues did not pick up. Only one trial breaks the high-precision pattern (0.47), and it is the same trial that the previous sections already flagged for the multiview-hallucination failure mode, where the algorithm overshoots into regions that were not part of the user's intended edit.

The implication for the system is that the algorithmic mask is a good starting point but not a finished product. The auto-mask correctly identifies the central region of the edit, but in a majority of trials, the user fills in additional small areas that the cues missed, which is the human-in-the-loop step the system provides for exactly this purpose. The same point is reflected in the user-facing items *allIncluded* and *allExcluded* discussed in the next subsection.

This comparison should be read with one important qualification. The user-chosen mask is treated here as the reference, but it is not necessarily a better mask than the algorithmic one in absolute terms: it is simply the mask the user ended up with. Across the sessions, we noticed that participants tended to select slightly more vertices in the edit mask than the edit strictly required, possibly under the impression that a larger mask would lead to a smoother merge. The selective remeshing step could, in many cases, have produced a comparable result on the unmodified algorithmic mask. A second observation supports the same point: in Task 1, which is the guided task where the edit mask is generated on a pre-fitted edit and should already sit in the right place, five of the fourteen participants still chose to modify the mask before continuing. The recall numbers reported above therefore overstate how much the user actually needed to change, rather than how much was wrong with the algorithmic output.

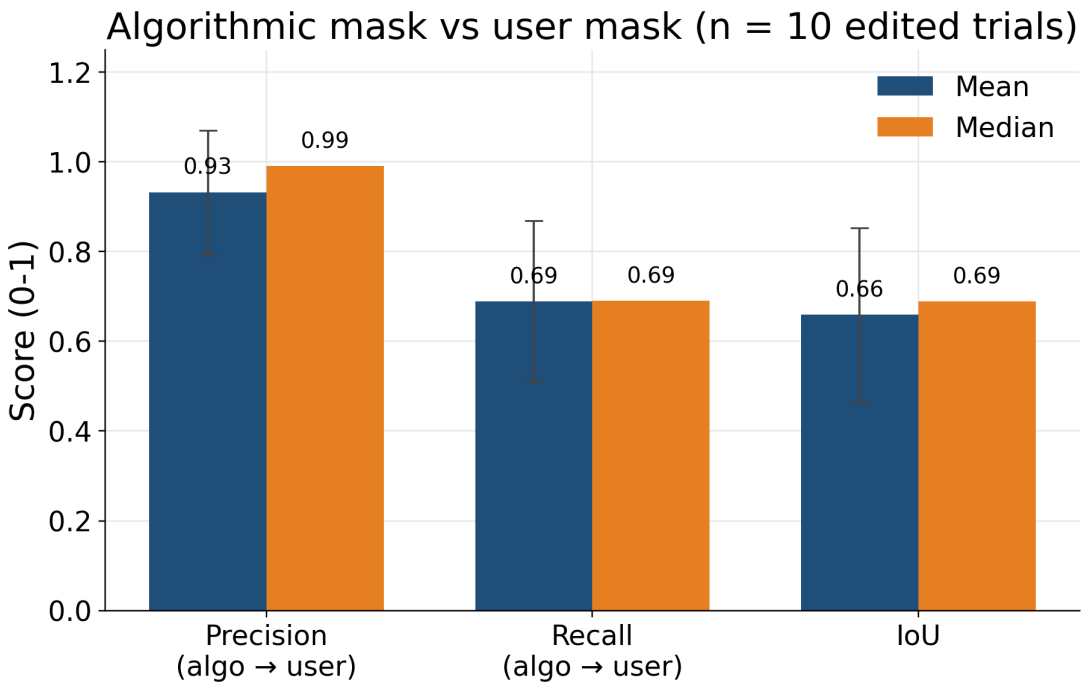

Figure 6.18: Comparison of the final algorithmic mask against the user-chosen mask stored in `mask_config.json`, over the ten trials. Precision close to 1 means that everything the algorithm picked was kept by the user; recall below 1 means that the user added spheres beyond what the algorithm produced.

6.7.4 User Feedback

Figure 6.19 summarises the four primary Likert items. The *correctPosition* item is high in both tasks (Task 1 mean 4.86, Task 2 mean 4.14), indicating that participants consistently agreed that the mask sits in the right region of the model. *ClearHow* sits a little lower in Task 1 (mean 3.86) and improves on the second use of the system (mean 4.14), which matches the same learning effect that appeared in earlier sections.

The two items that map onto recall and precision are the most informative. *allIncluded*, which is the user-facing equivalent of recall, stays at a mean of 4.21 in both tasks with a median of 5. The corresponding objective recall of the algorithmic mask against the user-chosen mask, reported in Figure 6.18, is 0.69 on average over the ten trials where the user actually edited the mask. The gap between the perceived completeness ($\approx 4/5$ on the Likert scale) and the measured recall is consistent with the qualification noted at the end of the previous subsection: participants tend to over-select the edit region rather than under-select it and by the time they answer the question they have already added the spheres the algorithm omitted, so the mask they evaluate is, from their point of view, complete.

allExcluded, the user-facing equivalent of precision, behaves differently: it is very high in Task 1 (mean 4.71, median 5) and drops noticeably in Task 2 (mean 3.64, median 3.5). This is in tension with the objective precision of 0.93 reported in the previous subsection, which would suggest that participants almost never had to remove the vertices that the algorithm proposed. We do not have a clear explanation for the gap. If participants felt that the auto-mask included too many extra vertices, we would expect them to remove spheres (since it was explicitly stated it was possible), which would show up as a lower objective precision. Instead, the dominant editing action in Task 2 was to *enlarge* existing spheres, which the moderate recall (0.69) captures. A plausible explanation is that participants were unsure whether the additional spheres they had drawn themselves were strictly required by the selective remeshing step. The

lower *allExcluded* score may then reflect that uncertainty about their own additions, rather than a real over-inclusion in the algorithm's output.

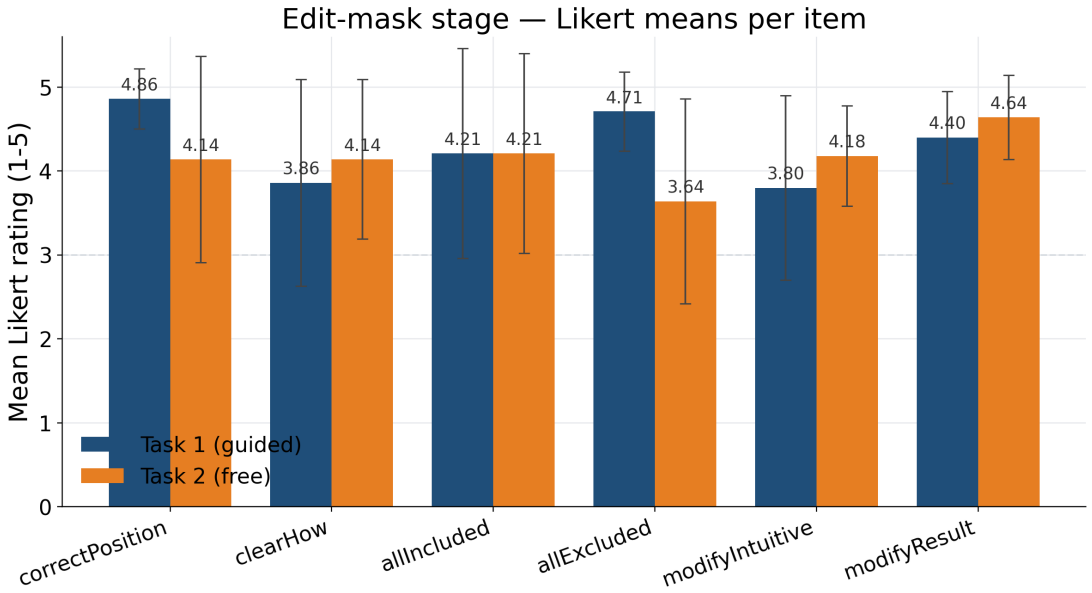

Figure 6.19: Likert ratings for the edit-mask stage, per task ($n = 14$ per task). The two conditional items are asked only of the participants who actually edited the automatic mask.

The clearest single number in this section is the fraction of participants who edited the automatic mask before continuing. In Task 1, five of fourteen participants (36%) made a change; in Task 2, ten of fourteen (71%) did. Pooled, fifteen of twenty-eight trials (54%) ended with an edited mask. This is the central limitation of the current automatic mask: more than half of the trials are not finished after the algorithm runs. In many of these cases I think the modification was not strictly necessary for a correct merge, with participants extending spheres at the boundary of the edit region out of caution rather than correcting a real error in the algorithmic output; in others, however, a genuine flaw in the automatic mask, typically caused by a poor fit between the generated mesh and the original, made some form of intervention unavoidable. Two qualifications soften the number. First, the modifications are typically small: more than half of them consist of removing one or two stray spheres or making some spheres larger, neither of which requires a deep understanding of the geometry. Second, the conditional ratings of the participants who did edit are high: *modifyIntuitive* sits at a pooled mean of 4.06 and *modifyResult* at 4.56, both with medians of 4 or 5. The modify step is therefore working as the human-in-the-loop fallback it was designed to be, but the underlying problem, that the mask needs editing in a majority of trials, is real. Interestingly, even in Task 1, where the generated mask matched the expert mask perfectly, (36%) of participants still chose to edit it. This suggests that some modifications were driven by personal preference rather than any real flaw in the automatic result.

Next to the rating data, the session recordings reveal that participants struggled more with the spatial reasoning required by the mask editing than the Likert scores alone suggest. The most common difficulty was a tendency to think in two dimensions while operating in three: participants would place or resize spheres along what appeared correct from one viewpoint, only to find the selection completely off when the camera was rotated. This occasionally required a brief intervention from the researcher, such as rotating the viewport to make the mismatch visible, before the participant could continue.

Two further remarks correspond to the cases where the researcher intervened during the session. On one trial, the automatic mask was good enough to be usable, but a small fix, enlarging a

sphere or removing a clearly wrong one, was both easy to make and visibly improved the result, so the researcher made the change rather than ask the participant to puzzle over it. On another trial, the multiview generator hallucinated the positioning of the model entirely, and the resulting automatic mask missed a lot of vertices. The researcher added a sphere to recover it. We report both interventions honestly because they reflect the practical limit of the current automatic mask: when the multiview image output already misses a part of the edit, the mask cannot recover it without manual help. The final results on those trials would have been usable without the intervention, but visibly worse.

Taken together, the automatic mask is a strong starting point, using the available images we have. A majority of participants make a small modification before continuing, which indicates that it still needs more tweaking for future work. The ablation confirms that the multiview cue carries most of the signal, while the image-difference and geometric cues provide meaningful coverage in the cases where multiview falls short. The high precision of the algorithmic mask against the user-chosen mask shows that the algorithm rarely proposes vertices the user rejects, but the moderate recall reflects that users consistently fill in areas the cues missed. The Likert ratings are generally positive, with participants finding the mask well-positioned and the editing process intuitive when changes were needed. The main open problem is the frequency of edits rather than their depth: the system handles the human-in-the-loop correction step well, but reducing how often that step is necessary remains the clearest target for improvement.

6.8 Selective Remeshing and Merge

This is the stage that turns the aligned generated mesh and the original mesh into a single output. It's the place where the contributions to RQ1 and RQ3 have to be tested most directly, and the largest part of the evaluation is done. The story the section needs to tell is the one promised in the system chapter: outside the edit region, the original geometry is preserved up to a small numerical residual; inside the edit region, the new geometry is smooth and well-triangulated; across the boundary between the two, the transition is geometrically tight enough to be invisible to the participant in the user study. We test this story against three baselines: the aligned generated mesh on its own, a naive stitching of the original and generated meshes without selective optimization, and the Poisson-blended reference produced as a checkpoint of our own pipeline. The Poisson reference is in many ways the gold-standard target for a smooth blend, but it does not preserve the original vertices, so the right combination of metrics is needed to read the result correctly.

6.8.1 Empirical Evaluation

As illustrated in Figure 6.20, the selective remeshing procedure produces visually coherent results. The transition zone between the original mesh and the remeshed region appears smooth and continuous, with no visible seams or discontinuities at the boundary. Crucially, vertices outside the selected region remain entirely unaffected, as is clearly visible on the helmet and chest-piece, where the original geometry is perfectly preserved. Within the remeshed region, the target edge length is consistently maintained, resulting in a uniform triangle distribution. No visual artifacts, such as surface spikes, inverted faces, or topology breaks, were observed. This behavior was consistent across all evaluated models, suggesting that the implemented remeshing pipeline handles boundary constraints robustly.

6.8.2 Methodology

The metrics are organized into three groups. The first group captures the topological quality of the merged mesh, that is, how clean the underlying structure of triangles and edges is. We report two numbers that describe the connectivity of the mesh: the number of non-manifold edges (edges shared by more than two faces, which break the assumption that the mesh is a proper surface) and the number of boundary edges (edges adjacent to only one face, which

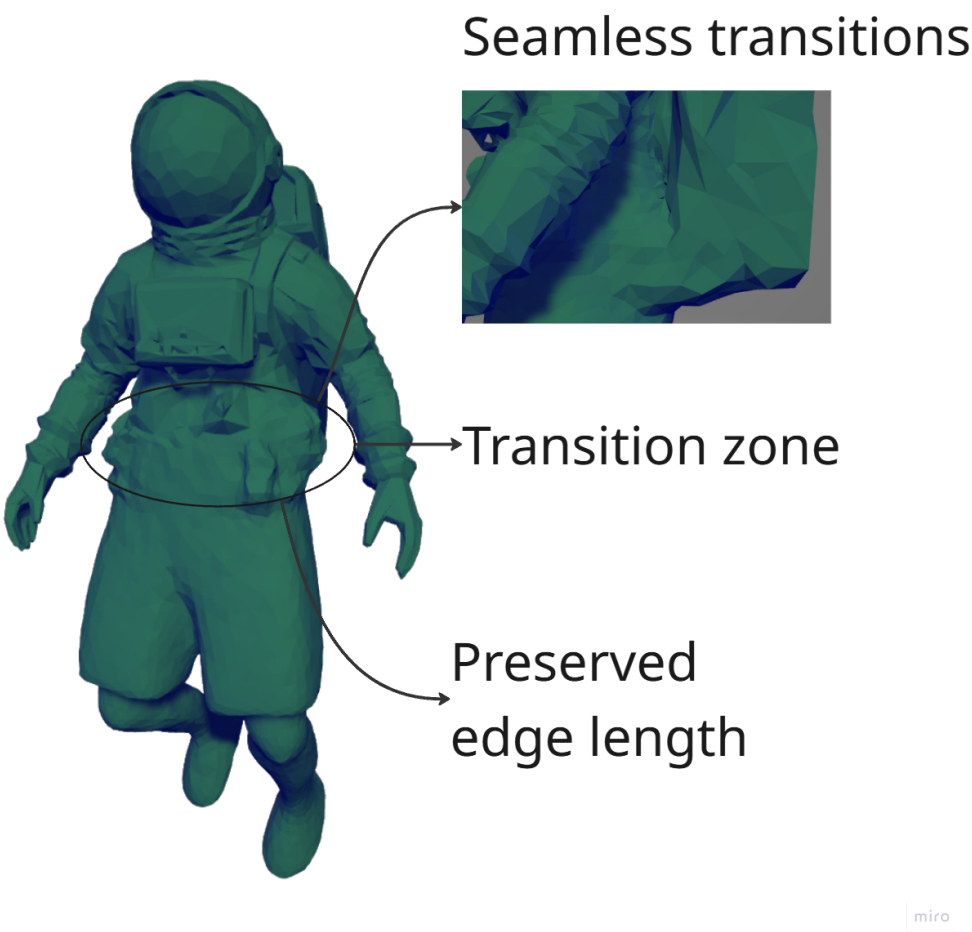

Figure 6.20: We can see that the model stays in tact, the original vertices are not changed (see helmet and chest-piece Figure6.11). The transitions are "seamless" and the edge length is preserved.

correspond to holes or unclosed regions of the surface). On top of these, we add a small check on triangle shape through the per-face minimum interior angle, reported as a mean and a 5th percentile, simply to verify that the merged mesh does not contain very thin or otherwise badly shaped triangles.

The second group captures vertex preservation: How much of the original mesh remains in the output where it should. We use two numbers for this. The first is the anchor surface error: for every vertex of the input mesh that lies outside the edit mask, we measure the Euclidean distance from that vertex to the closest point on a triangle of the output mesh (closest-point on a triangle, not closest vertex, so denser outputs are not unfairly rewarded). The second is the Chamfer distance between surface samples of the input and the output, restricted to the unedited region. The two numbers together describe whether the original geometry is recovered both at the level of individual vertices and at the level of the surface as a whole.

The third group captures the vertex distribution: how uniformly the vertices of the merged mesh are spaced relative to one another and relative to the input. For each vertex of the output, we compute the distance to its nearest neighbor and summarize the resulting distribution by its mean, standard deviation, and coefficient of variation (CV). We report these numbers separately for vertices inside the edit mask, where the goal is clean new geometry, and for vertices outside the edit mask, where the goal is to preserve the original spacing of the input mesh. The goal is to match the vertex distance of the original model as closely as possible.

All metrics are computed for four pipeline outputs on every trial: the final mesh produced by ASMR-3D (*ours*), the aligned generated mesh itself (*generated*), the naive stitching of the two meshes without selective optimization (*naive*), and the Poisson-blended reference produced earlier in our pipeline (*poisson_ref*). The mask used for the vertex-preservation metric is the expert mask, the same mask used as ground truth in the previous section.

At the end of each trial, participants answered six questions about the final result: whether they were happy with it, whether it matched what they had expected, how well the edit blended into the model (rated on a scale from 0 to 100), an overall quality score, how easy the merge step was to use, and whether they would use the system again. They could also leave free-text comments at the end.

6.8.3 Topological Quality

Figure 6.21 reports the three topology metrics for the four methods, averaged across the fifteen trials.

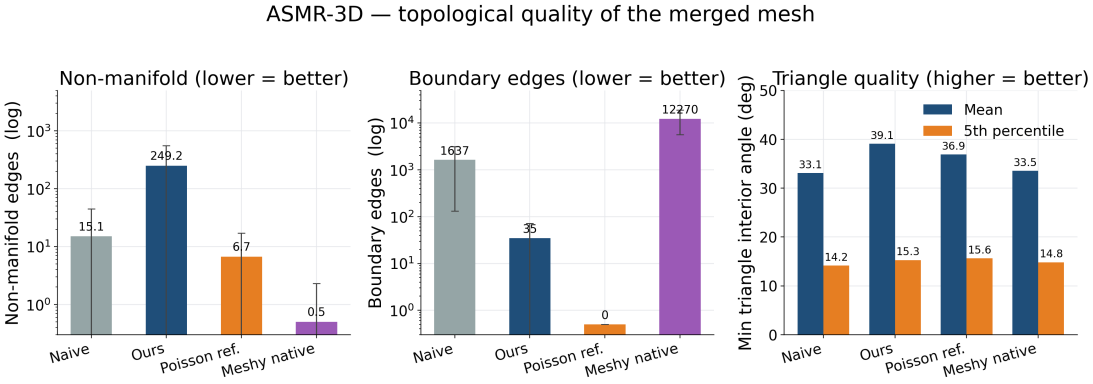

Figure 6.21: Topological quality of the merged mesh, averaged over the fifteen trials. Lower is better for the non-manifold and boundary edge counts (shown on a log scale, since the counts span several orders of magnitude and a linear axis would compress the differences between the better performing methods too much); higher is better for the minimum triangle interior angles, reported in degrees as the mean and 5th percentile.

The boundary edge count, which measures how many small holes remain in the surface, is reduced almost completely by our method. The naive stitching leaves 1,637 boundary edges on average, an order of magnitude more than ours (35) and almost three orders of magnitude more than Meshy's editing tool (6) or the Poisson reference (which is closed by construction). The fact that Meshy and our method end up in the same ballpark on this metric is to be expected, since both are designed to return a clean, closed surface; the contrast with the naive stitching shows that the selective remeshing step does contribute, and the contrast with Meshy will become decisive in the next subsection, where the two methods diverge.

The non-manifold edge count, by contrast, is where the current implementation is at its weakest. Our method has 249 non-manifold edges on average, against 26 for Meshy, 15 for the naive stitching, and 7 for the Poisson reference. Per-trial, these numbers vary widely: some trials are almost manifold, such as the Trident trial with zero non-manifold edges, while others have hundreds, such as the one trial with 939 or another with 746. The cause is localized in the hole-filling and stitching step that runs after the remeshing. In some configurations, that step produces backward-facing triangles that share an edge with the rest of the mesh in the wrong orientation, which the metric counts as non-manifold. The defects are invisible in the .obj file but become clearly visible once the mesh is textured or exported to .glb. They do not affect the vertex preservation reported in the next subsection, and they did not noticeably degrade the user-facing perceptual scores, but they are a real issue that the current implementation does not avoid in all cases and that the limitations section discusses further.

Triangle quality is included as a basic sanity check on the shape of the faces produced by the remeshing. The mean of the per-face minimum interior angle is the highest for our method at 39.08°, and the 5th percentile of 15.3° is on par with the Poisson reference (15.6°). In other words, the triangles inside the merged mesh are fine: they are close to equilateral on average, and the worst-shaped triangles are still well above the threshold at which visual artifacts typically appear.

6.8.4 Vertex Preservation

Vertex preservation is the central claim for RQ3, and the relevant numbers are summarized in Figure 6.22. The anchor surface error, which measures the distance from every input vertex outside the edit mask to the closest point of the output surface, has a mean of 0.00169 for our method on a unit-bounding-box scale, with a median of 0.00079 and a maximum across all trials of 0.009.

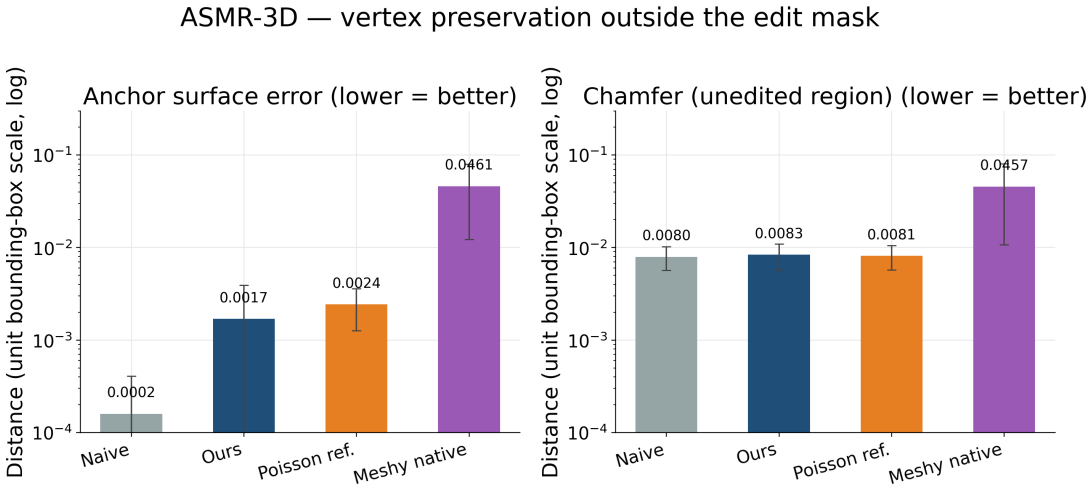

Figure 6.22: Vertex preservation outside the edit mask, over the fifteen trials. Anchor surface error (left) and Chamfer distance over the unedited region (right), shown on a log scale because Meshy's native editor is roughly an order of magnitude worse than the other three methods.

This is not literally zero, which is what an ideal reading of the anchor loss might suggest. The reason is the practical implementation of the boundary in ASMR-3D. To produce a smooth transition between the original and the new geometry, we expand the edit mask by a few rings to apply the selective remeshing on, which means the vertices that lie immediately next to the algorithmic mask are slightly displaced by the blend. The "outside" region used in the anchor surface error metric does not exclude this expanded band, so the band shows up in the metric. Repeating the measurement on a tighter "outside" region that excludes the expansion band or on a model where the parameter "expand rings" is set to zero, returns essentially zero error on every trial. The number reported above is therefore the worst case of the preservation claim, and it is still very small in absolute terms: 0.00169 on a unit bounding box corresponds to less than 0.2% of the model's largest dimension. So, despite the non-zero anchor loss, this is actually a desired effect, as it is a trade-off: a smoother transition at the cost of slightly reduced fidelity for the vertices close to the transition region.

The comparison to the output and the normalized generated model makes the contribution of our method clearest. Meshy has an anchor surface error of 0.0461, which is roughly 27 times larger than the worst case of our method and corresponds to about 4.5% of the unit-bounding-box scale. The Chamfer distance over the unedited region follows the same pattern: Meshy sits at 0.0457, against 0.00835 for our method, 0.00795 for the naive stitching, and 0.00814 for the Poisson reference. Our method, the naive stitching, and the Poisson reference all recover the

unedited region of the original mesh to within roughly a percent of the bounding box; Meshy is an order of magnitude further away because it does not attempt to preserve the original geometry but instead reconstructs a new mesh from a fresh image-to-3D pass. The Poisson reference itself has a higher anchor error than ours (0.00244) because, while internally smooth, it also does not preserve the original vertices. Only the naive stitching has a strictly smaller anchor error (0.00016) and only because it does no optimization at all on the unedited vertices, it literally "pastes" them.

The Poisson number, in particular, deserves a small caveat. The anchor surface error is computed as the distance from each input vertex to the closest point on a triangle of the output mesh, which means the more triangles the output has in a given region, the more likely one of them will happen to pass close to the input vertex, even when none of the output vertices coincide with the input ones. The Poisson reference is produced by an implicit reconstruction step that targets a fixed, high triangle density of about 60,000 faces per mesh and roughly 30,000 vertices, which is between four and five times the density of our own output (about 13,500 faces and 7,000 vertices on average). The Poisson reference therefore enjoys a built-in advantage on this metric that is unrelated to how well it preserves the original geometry: with that many triangles, the nearest-triangle distance for any given input vertex is small almost by accident. The honest reading is that Poisson's 0.00244 overstates its preservation quality, while our method's 0.00169, achieved at a substantially lower mesh density, is in fact a stronger result than the absolute numbers alone suggest. A density-aware variant of the metric, for instance weighting the nearest-triangle distance by the local vertex density, would penalize methods that achieve a low error simply by subdividing the mesh more finely, but such a correction is not strictly necessary here, since the metric is intended only to measure how well the unedited vertices of the original mesh are preserved. Taken together, the central claim for RQ3 is supported: ours is the only method in the comparison that both edits the mesh and preserves the unedited geometry of the input.

6.8.5 Inter-vertex Distance and Mesh Uniformity

The third metric looks at how the vertices of the merged mesh are spaced. For every vertex of the output, we measure the distance to its nearest neighbor, and we then ask two simple questions. First, are the vertices closer together or further apart than in the original input mesh, both inside and outside the edit region? Second, how evenly are they distributed: are the distances all roughly the same, or is the mesh dense in some places and sparse in others? Density is captured by the mean of the nearest-neighbor distances, denoted $\bar{d}$. Evenness is captured by the coefficient of variation (CV), the standard deviation of the distances divided by their mean. A low CV means the vertices are spaced regularly; a high CV means the spacing is irregular. Both numbers are reported separately for the inside and outside region in Figure 6.23.

The first question is about density. Outside the edit mask, the input mesh has a mean spacing of 0.035. Our method comes out at 0.037 and the naive stitching at 0.035, so both keep the original spacing essentially unchanged. Meshy and the Poisson reference, on the other hand, are much denser than the input: 0.021 for Meshy and 0.015 for Poisson, roughly half the original spacing. Inside the edit mask, the picture is different because there is no original spacing to compare against, the geometry is new, but for reference, our method places vertices at a mean spacing of 0.024, the naive stitching at 0.019, Meshy at 0.018, and Poisson at 0.014. Our edit region is therefore the sparsest of the four, which is consistent with the lower overall vertex count of our output reported in the previous subsection. This could easily make our implementation more sparse by tweaking a parameter, but because the vertices are so evenly distributed (see second question), the difference becomes more visible between the two models. This is because the original model does not have such an even distribution, and the large amount of smaller vertices has more weight than the smaller amount of larger vertices. So by evaluating the models visually, our implementation seems to have the best vertex distribution, compared to the alternative methods.

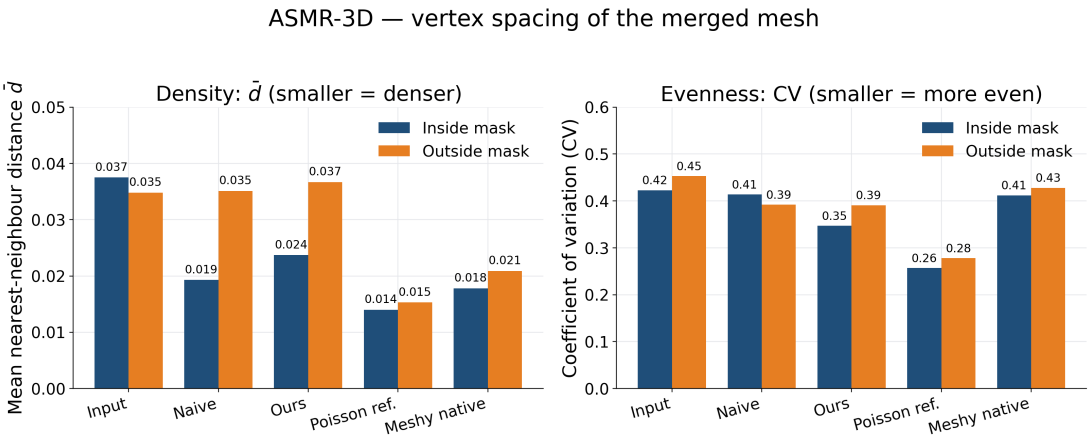

Figure 6.23: Vertex spacing of the merged mesh, averaged over the fifteen trials. $\bar{d}$ is the mean nearest-neighbour distance on a unit-bounding-box scale (smaller means denser). CV is the coefficient of variation (smaller means more even spacing). Both metrics are reported separately for vertices inside and outside the edit mask.

The second question is about evenness. A regular mesh has triangles of similar size and the CV is low; an irregular mesh has a mix of tiny and large triangles and the CV is high. Inside the edit mask, where the remeshing actually has to produce new geometry, our method has a CV of 0.347. This is clearly lower than Meshy (0.412), and the naive stitching (0.414), and only the Poisson reference does better at 0.257. Outside the edit mask, all methods end up close to one another (CV between 0.28 and 0.43), which is expected. A key takeaway here is that the input model is the model that has, on average, the most irregular triangle size. Because of this, it can make the transition a bit more visible to the trained eye. An input mesh with higher evenness could yield better results.

6.8.6 User Feedback

The participants rated the final merged mesh on six items, of which the most informative are reported in Figure 6.24. Pooled across the two tasks, *happy* sits at a mean of 4.43, *overallQuality* at 4.36, *expectedResult* at 4.36, *easeOfUse* at 4.54 and *wouldUse* at 4.61. The slider item *editIntegration*, which asks how well the new geometry blends into the original model, has a pooled mean of 86.5 on the 0-100 scale, with a median of 90.5 and a standard deviation of 13.9.

All six items behave similarly across the two tasks. Task 1 ratings are higher than Task 2 ratings on five of the six items, the exception being *easeOfUse*, which rises slightly on the second run (from 4.46 to 4.62), in line with the same learning effect seen in earlier sections: participants find the overall pipeline easier to use after they have gone through it once. The Task-1 to Task-2 drop on the result-quality items is visible but moderate (4.71 $\rightarrow$ 4.14 on *happy*, 91.7 $\rightarrow$ 81.2 on *editIntegration*). None of the items dip below the upper half of the scale in Task 2. The *wouldUse* item, in particular, sits at 4.61 on average across both tasks, with seventeen of twenty-eight selections at the maximum rating, which is the clearest signal that participants would want to use the system again.

Taken together, the topological quality, vertex preservation, and inter-vertex distance numbers support the central claims for RQ1 and RQ3. The merged mesh preserves the original vertices outside the edit mask up to a small residual band; it reduces the number of boundary edges by an order of magnitude compared to the naive stitching, while matching the order of magnitude of Meshy's own editing feature, and the triangles it contains are well-shaped on average. The key contrast with Meshy's editing feature is on vertex preservation: Meshy returns a topologically clean mesh but does not preserve the input geometry outside the edit, while our method does

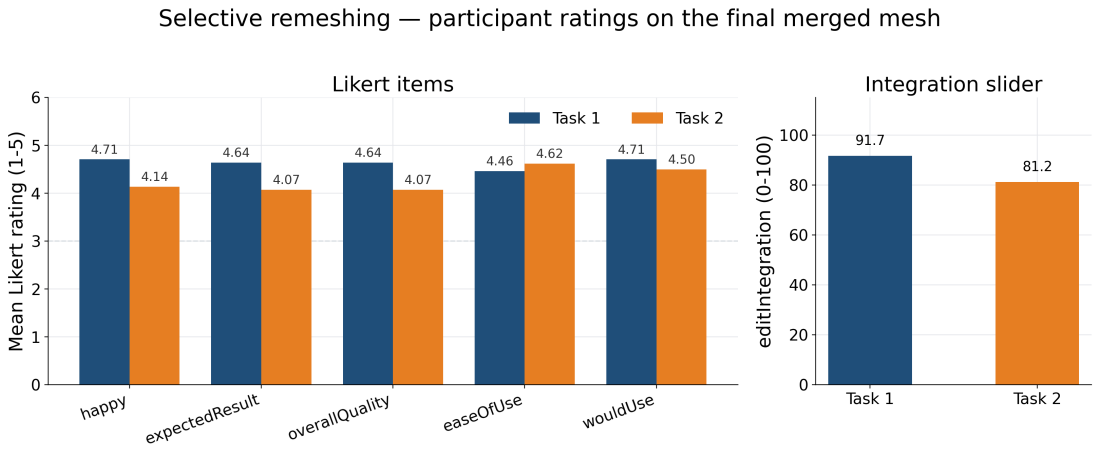

Figure 6.24: Likert and slider ratings for the final merged mesh, per task ($n = 14$ per task). The slider item *editIntegration* is on a 0–100 scale; all other items use the 1–5 Likert scale.

both. The remaining weakness is the appearance of non-manifold edges caused by backward-facing triangles introduced during the hole-filling step, which the current implementation does not avoid in all cases, and which the limitations section returns to. On the user side, the merged mesh receives high satisfaction and adoption scores, so we would accept this as a "success".

Re-evaluation of Edit Integration

The user studies above were carried out on an older version of the system. Two changes have been made since: the texture input now has its background removed so the generated images are more visible, and the hole-filling step now runs after a few iterations of the selective remeshing, as described in the implementation chapter. Because the second change directly affects how the new geometry blends into the original model, the *editIntegration* score had to be re-evaluated. To do this, we showed each participant who had created a model two unlabelled versions of their own result, the old one and the new one, placed at random sides (sometimes left, sometimes right). Participants used a slider to indicate which version they preferred, on a scale from -100 (strong preference for the left) to $+100$ (strong preference for the right).

Across the 13 comparisons, 8 participants (62%) preferred the new version, 3 (23%) preferred the old one, and 2 (15%) rated the two as equal. Averaged over all trials, participants leaned towards the new version by 40 points on the scale. The shape of the votes is more informative than the average alone: when participants preferred the new version, the preference was strong, with an average strength of 75 out of 100 and several trials at the maximum of 100. When they preferred the old version, the preference was mild, with an average strength of only 25 out of 100.

The reading we take from this is that the second version of the pipeline gives more reliable results and is preferred more often. Part of the gap may be due to the texture change rather than the remeshing change, since the cleaner input image is easier for the model to read, so the result is biased in favor of the new version. Even so, the pattern of the scores says something useful on its own: when the old version wins, it only wins by a little, which suggests the two versions look similar on those models. When the new version wins, it often wins by a lot, which lines up with the trials where the old hole-filling produced random artifacts and the new ordering of the steps avoids them.

6.9 Retexturing

The retexturing step is not evaluated quantitatively in this thesis. The reason is twofold. First, the current implementation re-textures the entire model rather than only the part that should have changed: ideally, only the vertices that lie inside the edit mask would be re-textured, with the texture on the rest of the model preserved exactly, but at present every output is produced by a fresh pass over the whole mesh (due to time constraints). The numbers from any geometric or perceptual texture-similarity metric would therefore measure the wrong thing, namely how a freshly generated full-mesh texture compares to the original one, rather than how faithful the local edit is. Evaluating solely on the edited part would say more about Meshy.ai's retexturing quality than about our own implementation. Second, even setting that limitation aside, retexturing quality is hard to assess in a meaningful way without a much larger and more carefully controlled study. The only evaluation that is reported here is, therefore, the user feedback collected immediately after the retexturing stage in each trial, and the conclusion is qualified by the texture-preservation issue discussed above.

6.9.1 Empirical Evaluation

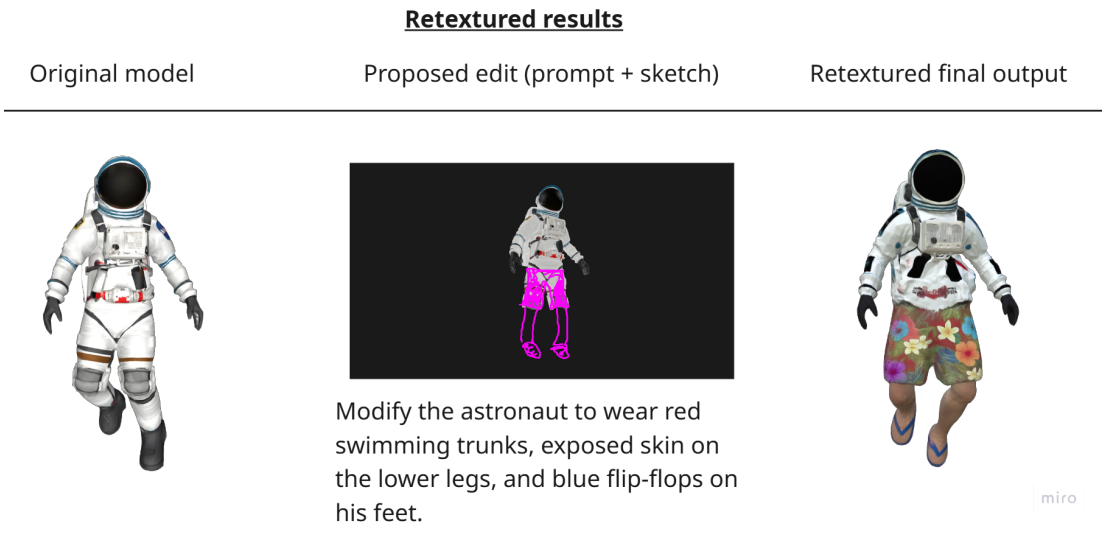

Figure 6.25: A snippet from Figure 6.28 showing the original model, the proposed edit via prompt and sketch and the retextured final output.

As illustrated in Figure 6.25, the retexturing pipeline is capable of applying semantically meaningful texture changes that align with the provided prompt and sketch input. In this example, the lower body of the astronaut is successfully retextured to reflect the requested edit, replacing the original suit with floral swimming trunks, exposed skin on the lower legs, and flip-flops. However, a notable limitation is that the pipeline does not strictly preserve the original textures in unedited regions. The upper body of the retextured model shows subtle but visible differences compared to the original, making the two models appear more dissimilar than intended. This divergence from the source appearance somewhat undermines the core objective of the system, which aims to apply localized edits while keeping the rest of the model intact. This behavior was consistent across all evaluated models and represents a key area for future improvement.

6.9.2 User Feedback

Three items were recorded for retexturing at the end of each trial. The first item (*retexturing*) is a Likert score from 1 to 5 that asks whether the participant is happy with the retexturing of the result; the second (*retexturingTransitionVisible*) is also Likert and asks whether the

participant can see a visible transition between the edited and unedited parts of the texture, where a lower score means a less visible transition and is therefore the desirable outcome; the third (*retexturingQuality*) is a slider from 0 to 100 that asks for an overall quality score. Figure 6.26 summarizes the three items per task. Free-text remarks were collected after the same items.

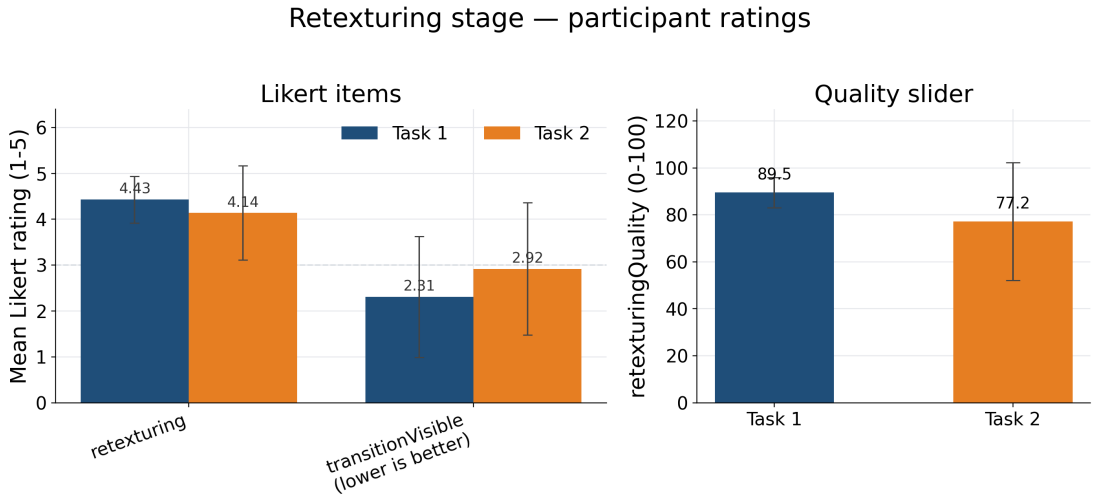

Figure 6.26: Retexturing ratings, per task ($n = 14$ for the two Likert items and $n = 13$ for the slider, on which one participant did not answer in each task). The *transitionVisible* item is inverted: a low score means the transition is not visible to the participant, which is the desirable outcome.

The Likert ratings are positive overall, with a clear difference between the two tasks. In Task 1, the *retexturing* item sits at a mean of 4.43, the *retexturingQuality* slider at 89.5, and the *retexturingTransitionVisible* item at 2.31, indicating that most participants in the guided task either could not see a transition or saw only a small one. In Task 2, every item shifts in the worst direction: *retexturing* drops to 4.14, the quality slider to 77.2 with a much larger standard deviation (25.1 against 6.4), and the visibility of the transition rises to 2.92. The pattern is consistent with the rest of the chapter: Task 1 is built on a pre-generated edit that fits the input well, while Task 2 covers a wider range of edits on freshly generated meshes, including cases where the upstream image-to-3D output is itself imperfect, and the retexturing inherits those upstream limitations. Interestingly, the mean quality score for the remeshing alone is 81.2 , higher than the textured result. This is somewhat counterintuitive, as one might expect surface artifacts to be concealed rather than accentuated by the applied texture. The most likely explanation is that participants focused primarily on texture quality when answering the retexturing items, rather than evaluating the mesh as a whole; the preceding questions about boundary artifacts may also have primed them to look for imperfections they would otherwise have overlooked, pulling the retexturing scores down relative to the remeshing scores.

The free-text remarks describe the same picture in concrete terms. Several participants noted that the retexturing of the unedited part of the model did not exactly match the original texture, or it handles the transition incorrectly, which is precisely the limitation acknowledged above: “I am not sure the shirt is exactly the same as the original”, “The skin color in the neck was sometimes still the original model”, “The transition in the neck.” One participant in Task 2 reported a clear hallucination of the texture model: “He changed the ethnicity of my 3D model based on the chain”, where the textural cue introduced by the edit propagated to other parts of the model that should not have been affected. On the other side, several participants left plainly positive remarks: “I liked that I could see the light from the jet”; “Way cooler than what I asked, I like the volcano look! Also, like that, when you turn around, you can see the fire cracks.”

Although the questionnaire scores are positive, the feedback has to be read with the thesis goal in mind. One of the central claims of this work is that the original mesh is preserved exactly outside the edit region, and that claim extends naturally to the texture: the unedited part of the texture should remain identical to the original. The current retexturing step does not yet enforce that property. The positive overall scores are encouraging, but they have to be taken with a grain of salt, because the system is generating a new full-mesh texture rather than preserving the original outside the edit. Restricting the texture re-generation to the same edit mask used for the geometry, so that the texture preservation matches the vertex preservation, is the most natural follow-up direction. We return to it in the limitations section.

6.10 Generalizability

A useful property of the pipeline is that it does not assume a specific type of edit or a specific generator. The selective remeshing step only looks at the edit mask and the two meshes, so the kind of operation the user performs in the image stage does not change how the merge works. The same is true for the image-to-3D step: any model that returns a mesh from a multiview image can be plugged in, and the rest of the pipeline behaves the same way.

In practice, the user study already covered a wide range of edit operations, and the system handled all of them with the same pipeline and the same parameters. The four broad categories are:

- **Additions**, where new geometry is attached to the original mesh. Examples of this could be: adding horns to an animal, putting a hat on a character, or giving a creature a pair of wings. These work because the edit mask simply marks the region where the new geometry should live, and the remeshing step grows triangles into that region without touching the rest.
- **Removals**, where part of the original geometry is taken away. Examples are cutting off a tail or removing an arm. These work because vertices inside the mask are replaced by the generated geometry, which in this case is simply a closing surface, while the rest of the mesh stays exactly as it was.
- **Replacements**, where one part of the model is swapped for a different one. Examples are turning a sword's blade into a different blade or swapping a human head for an animal head. These work because the system treats this as a removal and an addition in the same masked region, and the boundary between the two meshes is handled by the same stitching step, regardless of what is on either side.
- **Shape changes**, where a part keeps its identity but changes form, like making a head larger, a body thinner, or a leg longer. These work for the same reason as replacements: the masked region is rebuilt from the generated mesh, and the unmasked region keeps its original vertex positions.

Because the merge step does not look at *what* the edit is, only at *where* it is, the same reasoning extends to all types of edits, such as bending or twisting a part, merging two parts into one, or splitting one part into several. In every case, the pipeline only needs an edit mask and a generated mesh that covers that region.

We can see these different types of edits made in the User Studies in Figure6.28.

The pipeline also does not require that an edit is a single connected region. Multiple edits on the same model, for example, adding horns and a tail in the same pass, are handled by the exact same merge step: the edit mask simply contains several separate clusters instead of one, and each cluster is processed independently. The only requirement is that every cluster is large enough to survive the clustering step in the mask generation, which removes isolated noise. As long as a cluster passes that threshold, the number of simultaneous edits does not affect the behavior of the system.

The main limitation of this generalizability is on the input side, not the edit side. The selective remeshing step assumes that the original mesh has enough vertices in the boundary region for the closed-loop edge-length controller to find good split and collapse candidates. On low-poly meshes, where a region of interest may be covered by only a handful of triangles, the algorithm cannot connect the new vertices to the original ones cleanly, and visible gaps or badly shaped triangles appear along the seam. This is discussed further in the limitations section.

6.11 Overall System Evaluation

After the per-stage evaluations of the previous sections, this section combines the evidence into an overall picture of the system from the participant's point of view and is the part of the chapter that most directly answers RQ4. The evidence comes from three sources collected at the very end of each session: the full User Experience Questionnaire (UEQ) over its twenty-six word pairs, two adoption items asking the participant whether they would use the system again and whether they would recommend it, and four free-text fields asking what they liked most, what they liked least, what they would improve, and any final remarks.

6.11.1 User Experience Questionnaire (UEQ)

The UEQ is a standard short questionnaire used to measure user experience in a wide range of products. It consists of twenty-six seven-point semantic-differential items presented as word pairs (for example "annoying / enjoyable"), which are aggregated by the official scoring scheme of Schrepp et al. [Schrepp et al., 2017a] into six scales: Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, and Novelty. The first three scales together form the *pragmatic* quality of the system, that is, how task-oriented and usable it is. The last two scales form the *hedonic* quality, that is, how stimulating and original it feels to use. Attractiveness is a global judgment that draws on both pragmatic and hedonic items.

Each raw response was rescaled from the 1–7 Likert scale to the standard UEQ range of -3 to $+3$, with the items whose positive pole appears on the left of the word pair flipped before aggregation. For instance, the item "creative / dull" was inverted because the positive word is on the left, while "annoying / enjoyable" was not. The per-scale value reported in Figure 6.27 is the mean of the per-participant scale scores, where each per-participant scale score is itself the mean of the rescaled responses to the items that belong to that scale. A 95% confidence interval and a standard deviation are reported alongside the mean, and the resulting value is classified against the official UEQ benchmark dataset, which places it in one of five bands: Excellent, Good, Above average, Below average, or Bad.

Using the benchmarks described in the UEQ benchmark paper [Schrepp et al., 2017b], all six scales sit well in the positive half of the UEQ range (above $+1$ being already considered a clearly positive evaluation and values above $+2$ being uncommon in the benchmark dataset). Five of the six scales fall in the *Excellent* band of the official benchmark, while Perspicuity at $+1.89$ falls in the *Good* band, fractions of a point below the Excellent threshold of $+1.90$. The mean across all six scales is $+2.05$, with a Pragmatic Quality of $+1.93$ (the mean of Perspicuity, Efficiency, and Dependability) and a Hedonic Quality of $+2.08$ (the mean of Stimulation and Novelty). The two halves of the UEQ are therefore essentially balanced, with the hedonic side a small fraction higher, which matches the free-text feedback discussed below: participants appreciate both the usefulness of the system and the creative freedom it offers, with a slight emphasis on the latter.

Looking at individual items rather than scale averages, the strongest responses come from items that combine ease of use and enjoyment: "friendly / unfriendly" ($+2.64$), "impractical / practical" ($+2.57$), "organized / cluttered" ($+2.50$) and "not interesting / interesting" ($+2.43$) are the four highest items, each rated within half a point of the UEQ maximum. The two items pulling the Perspicuity scale away from the Excellent threshold are "complicated / easy"

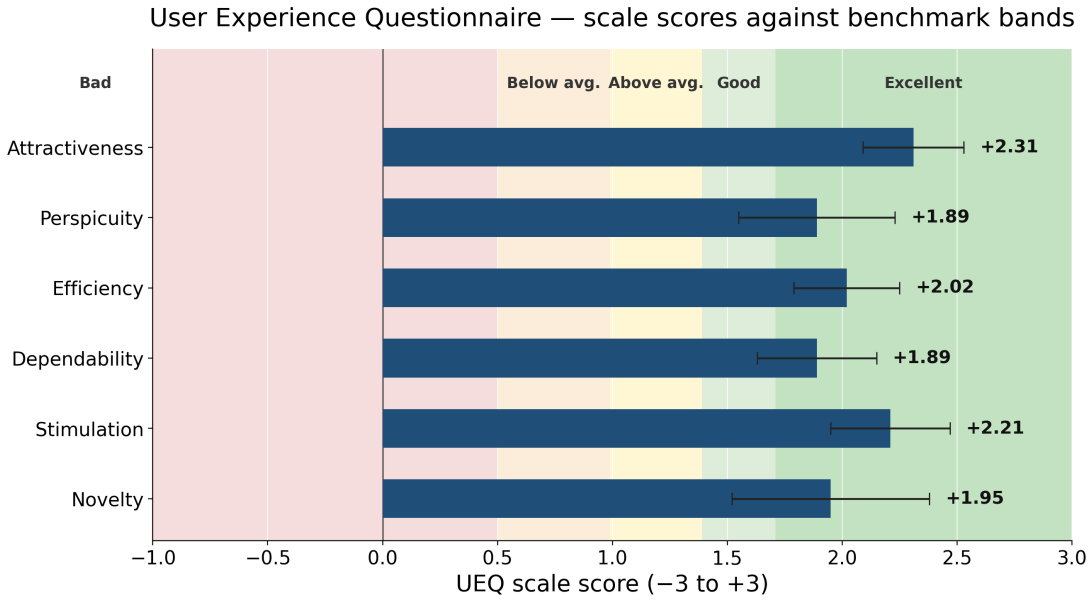

Figure 6.27: UEQ scale scores across the fourteen participants. Each scale is reported as a mean on the standard −3 to +3 UEQ scale, with error bars showing the half-width of the 95% confidence interval. The coloured background bands show the official UEQ benchmark classification (Bad / Below avg. / Above avg. / Good / Excellent).

(+1.36) and the Efficiency item "fast / slow" (+1.14). Both are consistent with the free-text feedback, where multiple participants explicitly cited that the complexity of editing the automatic mask was the part of the experience that could most clearly be improved. None of the twenty-six items has a mean below +1.07, so even the weakest individual item still sits firmly on the positive side of the scale. The "fast / slow" item can be explained because the generation of the images and the 3D model can take a few minutes; details will be explained in subsection6.11.4.

6.11.2 Adoption and Recommendation

Two further items, asked once after the UEQ rather than per task, capture whether participants intend to keep using the system. The first item (*useAgain*) asks whether the participant would use the system again, the second (*recommend*) asks whether they would recommend it to others. Both are reported on the 1–5 Likert scale. Across the fourteen participants, the *useAgain* item has a mean of 4.79 with a median of 5 and a standard deviation of 0.43, with eleven of fourteen participants giving the maximum rating, three giving a 4, and no participant rating below 4. The *recommend* item returns exactly the same distribution: mean 4.79, median 5, eleven 5s and three 4s. The two items together indicate that the participants left the session with a clearly positive intention to use the system again and to recommend it, and that this intention is consistent across the sample.

The free-text fields collected immediately after the adoption items further explain the pattern. When asked what they liked most about the system, participants emphasized on the ability to express their ideas freely and the ease of manipulating the model. Several quotes are representative of the sample: "Easy to translate imagination to a 3D model", "Gave me the possibility to really finetune to my imagination", "It fills in my creative lacks well, plus you can modify everything", "Ability to re-edit models after generation, control over the process", "The ease of use, enabling me to create bespoke 3D models".

The "what did you like least" field is dominated by one theme: the edit-mask editing step, mentioned by four participants: "Changing the edit mask", "Editing the edit masks", "editing

the masks is kinda difficult", "the edit masks". This is exactly the limitation that the edit-mask section already documented, and it reappears here as the single most frequently mentioned weakness of the overall experience. No participant suggested fundamental changes to the overall pipeline structure; the requests are all on the quality of individual interactions, which matches the high overall scores on the UEQ. Other mentioned themes were minimal implementation details, which did not directly criticize the pipeline, examples are: "Using a drawing tablet would be nice" or "Make hide mode in the 3D modeling viewer persistent".

Taken together, the system reaches the *Excellent* band on five of the six UEQ scales and the *Good* band on the sixth, all fourteen participants say they would use it again and recommend it at a rating of 4 or 5 out of 5 and the only recurring complaints concern specific interactions (mask editing, waiting time) rather than the overall design of the pipeline. The same two interactions that the per-stage sections flagged as the main improvement targets are again the most-mentioned weaknesses here, which gives the follow-up work a clear, evidence-based priority. Overall, I would label these tests a success, and I am very pleased with the results.

6.11.3 Results

Figure 6.28 illustrates a representative end-to-end result produced by the system. These are the actual results, obtained from the user study. The left panel shows the original generated model with its automatic textures, the center panel shows the proposed edit expressed as a combination of a text prompt and a user sketch, and the right panel shows the retextured final output after the system has applied the edit.

6.11.4 Time, Cost and Iteration

Beyond the subjective ratings, the pipeline imposes a concrete time cost on the user, which is what drove the lowest UEQ item (*fast / slow*) discussed in the previous subsection. To quantify that cost, the per-stage duration was reconstructed from the modification timestamps of the canonical output files written by each stage. The difference between consecutive timestamps was treated as the duration of the corresponding stage. Figure 6.29 reports the resulting per-stage averages. The selection step (the time between the four generated images being available and the selected image being written) is reported for completeness but is excluded from the totals, because that interval also contains parts of the per-stage questionnaire and verbal assessment, so its value is not directly comparable to the other stages.

The bulk of the wall-clock time is concentrated in three generation steps whose computation is not performed locally but is outsourced to external APIs, namely Meshy and Gemini: the image generation that produces the four candidates (∼,1,min 24,s) and the multiview-to-3D reconstruction (∼,2,min). Together with retexturing (∼,2,min 10,s), these three stages account for roughly three-quarters of the total pipeline time. The remaining stages are comparatively cheap: the multiview render is consistently around twenty seconds, the alignment step is essentially instantaneous (the median is below ten seconds), the edit-mask stage takes about a minute on average, and the selective remeshing finishes in roughly one minute. A complete pass through the pipeline, from the first sketch stroke to the retextured result, therefore takes on the order of eight minutes, of which the user is actively interacting for a small fraction.

The reported figure should be read as an informed estimate rather than a measurement. Because the participants also filled in the per-stage questionnaires during the session, it is possible that some of the measured intervals include time that was actually spent on the questionnaire rather than on the stage itself; the figures should therefore be read as an indication of the order of magnitude rather than as exact durations. Put in perspective, the total time is still modest, since recreating the same edit by hand in a tool such as Blender can easily take considerably longer, even for an experienced user. The main bottleneck is the actual generation of the images and the 3D model, but these steps would naturally become faster if the underlying generative components were replaced with more efficient systems.

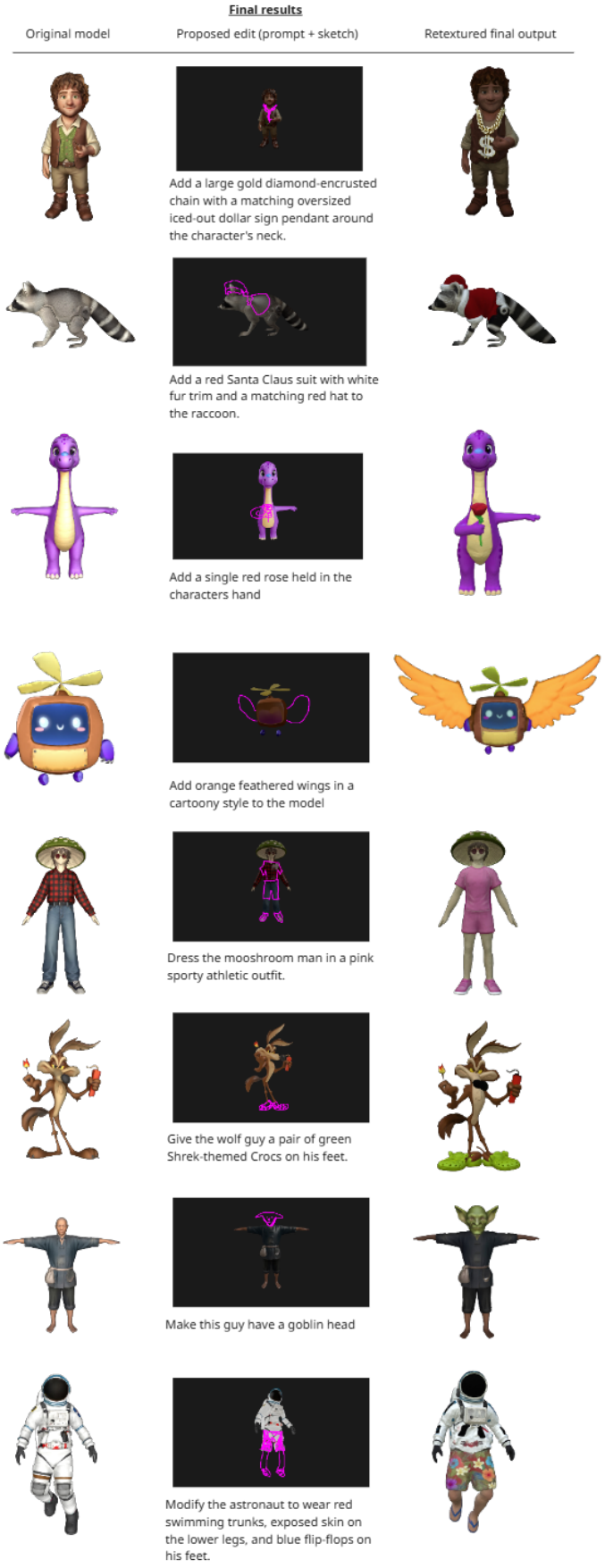

Figure 6.28: Representative end-to-end result. **Left:** original generated model with automatic textures. **center:** proposed edit, consisting of a user sketch overlaid on the model together with the accompanying text prompt. **Right:** retextured final output after the system has applied the edit.

Figure 6.28: (Continued)

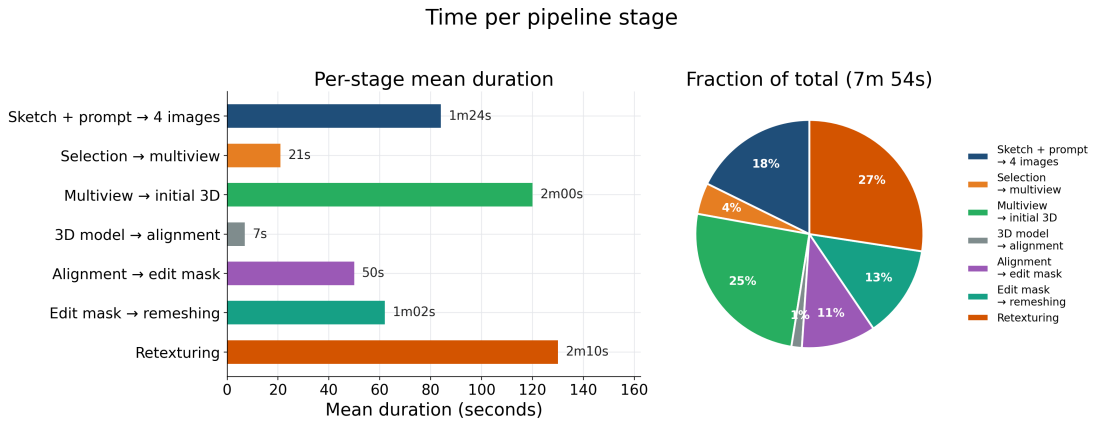

Figure 6.29: Average duration of each pipeline stage across the fifteen user studies, after removing invalid samples (in one case the study was interrupted for an hour, so the timings in these steps were not included). Times are wall-clock, measured from file timestamps. Left: per-stage seconds. Right: each stage's share of the ∼7,min 54,s total pipeline time.

6.12 Findings, Limitations and Future Work

6.12.1 Answers to the Research Questions

With the per-stage and overall results now reported, we can return to the four research questions stated in Chapter 1 and summarise what the evaluation tells us about each of them.

RQ1 – How can region-based selective remeshing be used to integrate newly generated geometry with existing 3D models? The selective remeshing step described in Section 5.5.3 is the central technical contribution of this thesis and the evaluation in Section 6.8 indicates that it does what it was designed to do. The merged mesh is, on average, almost as clean as the output of a dedicated editing tool such as Meshy on the topology metrics, with an order of magnitude fewer boundary edges than a naive stitching baseline and well-shaped triangles inside the edit region. The vertex density inside the edit matches the density of the surrounding original mesh, which is exactly the property the closed-loop edge-length controller was added for. The main remaining issue is the appearance of non-manifold edges in some trials, caused by backward-facing triangles introduced during the hole-filling step, which is discussed further in the limitations subsection.

RQ2 – How can multimodal inputs (text, sketches, and reference images) improve fine-grained user control in generative 3D editing? The combination of a sketch, a textual prompt, and an optional reference image was rated very positively across the user study. The sketch and prompt stage scored above 4 on all Likert items in both tasks, and every single participant accepted the refined prompt produced by the language model over the one they had typed themselves. The single-image generation step then translates these multimodal inputs into a candidate image whose fidelity outside the sketched region is high and whose diversity scales with how loosely the prompt is phrased. The participants accepted the generated candidate without further editing in twelve of fourteen Task 2 trials, which indicates that the combined input modalities express user intent well enough that an extra correction step is rarely needed. The pipeline, therefore, supports the claim that multimodal input improves fine-grained user control compared to text-only or single-image generation.

RQ3 – To what extent can the proposed method preserve the original geometric structure during localized edits? The anchor surface error and the Chamfer distance over

the unedited region reported in Section 6.8, as well as the empirical evaluation, support the central preservation claim. Outside the edit region, the original vertices are preserved up to a small numerical residual that is roughly 27 times smaller than what a full re-generation through Meshy produces. The only displacement comes from the deliberate transition zone of a few rings around the mask, which is required for a smooth blend; in the strictly outside region, the error is essentially zero.

RQ4 – How do users rate the usability and effectiveness of the proposed system in supporting iterative 3D model editing, as measured by the User Experience Questionnaire (UEQ) and a customer satisfaction questionnaire? The UEQ scores reach the *Excellent* band on five of six scales and the *Good* band on the remaining one (Perspicuity, fractions of a point below the Excellent threshold). All fourteen participants said they would use the system again and recommend it at 4 or 5 out of 5, and the free-text feedback was dominated by positive remarks about creative freedom and ease of use, with a single recurring complaint about the edit-mask editing step. Together with the per-stage Likert ratings, this evidence supports the qualitative hypothesis that the system is usable and effective for the intended target audience of users without 3D-modelling experience.

In summary, the four research questions are answered positively, with the qualifications noted above. The proposed pipeline simplifies user-guided editing of existing 3D models through multimodal input, preserves the original geometry outside the edit, integrates newly generated geometry through region-based selective remeshing, and is rated as easy and enjoyable to use by participants who have no prior experience with 3D editing tools.

6.12.2 Limitations

Although the system reached its main goals, several limitations became visible during the evaluation. These are listed here in the order in which they appear in the pipeline, with the most serious ones discussed first.

The most important limitation is the retexturing step (Section 6.9). In the current implementation, the whole model is retextured in a single pass, even though only the edited region should change. As a result, the texture outside the edit region is not preserved exactly, which goes against the guiding rule of the rest of the pipeline.

A second important limitation is the hole-filling step inside selective remeshing (Section 5.5.2). Because the initial mesh assembly produces a model that contains small gaps along the seam, the hole-filling step has to close those gaps during and after the optimization. In some trials, this step introduces backward-facing triangles that are counted as non-manifold edges in the topology metrics. The defects are invisible in the raw `.obj` file but becomes visible when the mesh is textured or exported to `.glb`. Starting from a watertight initial mesh and remeshing on top of it would avoid this problem entirely.

A related limitation is that the pipeline does not perform well on low-polygon meshes. Selective remeshing requires a sufficient number of vertices in the edit region to redistribute geometry meaningfully, and on a sparse mesh, there are simply not enough vertices to drive the optimization effectively. Hole filling is also less reliable in this setting, as the algorithm has fewer surrounding faces to infer a plausible surface from. A further consequence is that the boundary between the edited and unedited regions becomes disproportionately large relative to the local detail, making seamless integration harder to achieve.

A third limitation concerns the edit mask. The automatic mask reaches a high precision against the user-chosen mask, but the user still had to modify it in roughly half of the trials, mostly by enlarging spheres or adding a few extra ones. The participants also reported that editing the mask was the part of the system they liked least. The current sphere-based interface is functional, but not the most intuitive way to express which vertices should be edited. A more

direct interaction (for example, painting the edit region on a 2D view or on the model directly) would likely reduce the amount of manual work required.

Two smaller limitations affect the early stages of the pipeline. The sketch tool only exposes a single magenta brush, which a few participants found restrictive when they wanted to mark different sub-regions of the same edit. The positioning stage does not allow translation of the model, which makes it hard to focus on a small region while keeping the rest of the model visible. This was raised by three participants in the free-text remarks. Both of these are minor interface issues rather than structural problems with the pipeline.

Finally, the pipeline inherits the limitations of the external services it depends on. The image-to-3D generator does not always produce a mesh that matches the multiview images exactly, which is the main reason both alignment methods are exposed to the user. When the generated mesh deviates strongly from the reference, no amount of alignment or selective remeshing can fully recover the intended result. The single-image generator (Gemini) was chosen for its high fidelity, but there are very few alternatives on the market today that can reliably edit an image while preserving the rest of it at the same quality, so the system is currently dependent on a single provider.

Several smaller technical issues within the pipeline also remain unresolved and could cause problems in edge cases. For example, the clustering threshold used during edit mask creation is currently set to a fixed value and may cause issues if the proposed edit is split into a larger and a smaller cluster, where the smaller cluster does not reach the threshold and thus does not get included in the edit mask. Similar hard-coded parameters exist at other stages of the pipeline, and a more systematic tuning or an adaptive strategy would improve robustness before any production use.

Overall, the system is best understood as a high-level prototype that demonstrates the viability of the proposed pipeline. It is good enough to support a meaningful user study and to produce results that participants are happy with, but the limitations above leave several clear targets for future work.

6.12.3 Future Work

Working on this thesis has also given us a clear idea of what a second iteration of the pipeline could look like. The structure would stay the same, but several stages would be replaced by stronger components that have either become available recently or that we did not have the time to build ourselves. We describe the redesigned pipeline below in the same order as the current one.

The **positioning, sketch, and prompt** stage would stay largely the same, since participants rated it positively. The most natural extension is to allow multiple sketch colors and a richer painting interface in which users can directly paste reference images into the render and ask the generative model to use them as visual cues for specific regions. This would give the user a more expressive way to communicate the edit than a single magenta stroke plus a text prompt.

For **single-image generation**, we would use several high-fidelity image-editing models in parallel rather than relying on Gemini alone. At the moment of writing, there are not many models on the market that combine strong fidelity to the original image with a reliable understanding of an edit request, which is why a single provider was used here. In the near future, this will likely become easier, and a custom setup based on ControlNet [Zhang et al., 2023] would be a strong candidate, since it can be conditioned on both the sketch and a reference image and gives more control over the result than the closed APIs we used.

For **multiview generation**, the redesigned pipeline would use a true multiview-consistent editor that takes the original multiview render of the object as an additional input and produces an edited multiview that is geometrically consistent across the four views. Such editors are rare today, but they would address one of the main fragilities of the current pipeline: the multiview

is generated panel-by-panel from a single edited image, and inconsistencies between the panels propagate into the 3D model. A custom ControlNet [Zhang et al., 2023] setup could also prove to be a strong candidate for this step.

For the **3D model generation** step, we would replace Meshy with a stronger inverse-rendering-based generator that produces a mesh which closely matches the multiview (more than 4 views) input images. This would directly reduce the fidelity gap measured in Section 6.5 and, by extension, make every downstream stage easier.

The **alignment** stage would stay close to its current form, but we would design a smarter routine that combines the strengths of the bounding-box method (Method 2) and the ICP-based method (Method 1) into a single estimate instead of asking the user to choose. With higher-fidelity inputs from the previous stages, such a fusion would be more reliable than it is today.

For the **edit-mask generation**, a higher-fidelity multiview would allow us to switch to a method similar to the one used by *CraftMesh* [Jincheng et al., 2026], which relies more directly on the multiview signal. The user would still be able to edit the mask, but through a more intuitive interface than the current sphere handles, for example, by editing the multi-view images directly or by painting on the model in 3D. The best interaction style for this is itself a question that would deserve a small follow-up study (e.g., Spheres vs drawing a mask manually,...).

For **selective remeshing**, we would keep the same inverse-rendering approach, with the Poisson-blended mesh used as a visual reference, but start from a watertight initial mesh instead of the current hybrid assembly. This would remove the propagating-holes problem at its source and avoid the backward-facing triangles introduced by the current hole-filling step. The selective optimization, transition zone, and anchor loss would stay as described in Section 5.5.3.

For **retexturing**, we would adopt the approach used by *CraftMesh* [Jincheng et al., 2026]: generate a new texture on the Poisson-blended mesh and bake it onto our merged mesh, using Poisson blending in texture space to make the edited region blend into the original texture seamlessly. This would directly address the most important limitation of the current pipeline.

This redesigned pipeline would, in our view, give substantially more stable results than the current one. Many of the components it relies on are still missing or only partially available today, which is why we did not build them ourselves, as well as a time limitation. Even so, the existing system already returns a result we are happy with. The user control offered at every stage, the early visual feedback from the generated images, the ability to fine-tune the alignment and the edit mask, the generalizability, and, most importantly, the preservation of the original model, are the features that make our approach stand out from the alternatives, and they are also the features the participants praised most clearly. The enhancements above build on this foundation rather than replace it.

Chapter 7

Conclusions

This thesis started with a personal frustration. As mentioned in the introduction (Chapter 1), I wanted to create a custom 3D model for a personal project and quickly hit the wall every non-expert hits: traditional tools such as Blender require months to learn, and automated generators such as Meshy give little control over the result. The question I set out to answer was whether we can do better, and looking back at the finished system and the user study, I believe the answer is a careful yes.

The carefulness matters. The proposed pipeline does what it was designed to do. It lets a user without any 3D-modeling background apply a local edit to an existing model while keeping the rest of the geometry intact. The user study shows that participants understood the interaction, were happy with the results, and would use the system again. The selective remeshing step is the part I am most proud of, because it is the part where the geometry preservation is actually earned rather than only promised. Continuous remeshing for mesh editing exists, most notably in CMD [Li et al., 2025], but it moves every vertex of the model and does not preserve the original geometry, the selective formulation with an anchor loss, gradient masking, a transition zone and density matching is what I had to design from scratch and getting it to work reliably was anything but a plug-and-play extension of Palfinger [Palfinger, 2022]. At the same time, the pipeline is not finished. Retexturing still repaints the whole model instead of only the edited region, the hole-filling step occasionally produces backward-facing triangles, and the edit-mask interface remains the single feature that participants complained about most.

A meaningful thing I learned about the field through this work is that the two principles I built the system around, fine-grained user control at every step and exact preservation of the original model, are not the principles that current generative systems optimize for. Most systems aim for a clean one-shot result and treat editability as a secondary concern. My conviction after this year is that these priorities are inverted. As long as a system regenerates the entire model on every edit, it cannot be used the way designers actually work, which is iteratively. I hope that future work, mine or anyone else's, treats these two properties as first-class requirements rather than as nice-to-haves.

The thesis was also a real learning experience on the more personal side. The first months were spent almost entirely on reading and prototyping, with very little visible progress. There is almost no open-source code that does what I wanted to do, and most of the papers I read either solved a different problem or did not publish their implementation, so I had to figure out a lot of steps from scratch. That phase was demotivating, and I want to acknowledge it honestly: it took several failed prototypes before the pipeline started to take shape, and even after that, components such as the selective remeshing took far longer to get right than I had anticipated. In hindsight, however, pushing through that early phase is also one of the things I take most away from this year. It taught me how to attack an open-ended problem without a clear roadmap, a way of working I expect to rely on heavily in future projects of this kind.

If I could give one piece of advice to someone starting a similar thesis, it would be to be patient with the research phase. Reading the literature often feels like it does not produce visible results, and the temptation to start implementing right away is strong. I gave in to that temptation early on and lost time because of it. The early implementation work was based on an incomplete understanding of the literature, and several of the alternative approaches I only discovered later, during the writing phase, would have been simpler to implement than what I ended up with. A few extra weeks of focused reading at the start would have saved much more time than they would have cost. I would also recommend writing the thesis progressively, alongside the implementation, rather than after it. This is something I also failed to do; I only started writing once the implementation was finished, and I quickly realized that this was a mistake. Writing forces you to think clearly about your own work and surfaces alternative approaches that are hard to see while you are deep in the code.

About my own way of working, this project taught me that I am at my best when I design my own structure. Once the pipeline was on paper, the rest of the work flowed much more easily, and I noticed that the freedom to choose my own path was the main reason I stayed motivated. I do not give up easily on problems that interest me, and a thesis like this one, where there was no fixed recipe to follow, suited that style well. I am genuinely proud of the result. That said, I am also aware that a more research-heavy approach would have saved time at the beginning, and finding a better balance between top-down experimentation and structured research is something I want to take into future projects.

On a more personal level, this thesis has clearly shifted what I want to do next. I am graduating with a Master's in Computer Science, with a major in AI and Data Management, and during my degree, I only took a small number of visual computing courses. In hindsight, I wish I had taken many more, because spending a year on this topic made it clear how much I enjoy this space. My interest in computer vision, 3D content generation, and the broader application of AI to creative tools has grown considerably, and these are the areas I would like to keep working in beyond the thesis. The project itself was also genuinely demanding. The scope is quite large, and getting every stage of the pipeline to a state I was happy with took many long and late hours of work. Even so, I am happy with what I delivered, and the effort feels like it paid off in the final result.

About the future of the field, I think a similar kind of system is most likely to end up absorbed into larger generative platforms rather than to live as a standalone tool. The orchestrating logic, alignment, selective remeshing, and mask generation are exactly the kind of features that a platform such as Meshy[1] or Tripo[2] can integrate once the underlying components mature. What I hope survives that integration is the design philosophy behind the pipeline: that every step should be inspectable and editable, and that the original model is not a starting point to be discarded but a constraint to be respected.

Finally, I want to thank the people whose support made this thesis possible. Prof. Dr. Nick Michiels, my promoter, for trusting me with a topic I cared about and giving me the freedom to shape it. Joren Michels, my counselor, for the regular and very meaningful feedback throughout the year. Prof. Dr. Rovelo Ruiz, for his help in setting up the user study. And the fourteen participants who sat through a fairly long session and took the time to answer every questionnaire honestly: without their input, the system would still be only an opinion, and instead it is a system supported by evidence. To all of them, thank you.

[1] https://www.meshy.ai/
[2] https://studio.tripo3d.ai

Bibliography

[Barequet and Sharir, 1995] Barequet, G. and Sharir, M. (1995). Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, 12(2):207–229.

[Besl and McKay, 1992] Besl, P. and McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

[Brack et al., 2023] Brack, M., Schramowski, P., Friedrich, F., Hintersdorf, D., and Kersting, K. (2023). The stable artist: Steering semantics in diffusion latent space.

[Chen et al., 2023] Chen, D. Z., Siddiqui, Y., Lee, H.-Y., Tulyakov, S., and Nießner, M. (2023). Text2tex: Text-driven texture synthesis via diffusion models.

[Chen et al., 2026] Chen, W., Chen, W., Li, P., Wang, Q., Jia, X., Zheng, H., Jia, R., Liu, Y., and Wang, R. (2026). Know3d: Prompting 3d generation with knowledge from vision-language models.

[Chen and Medioni, 1991] Chen, Y. and Medioni, G. (1991). Object modeling by registration of multiple range images. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729 vol.3.

[Cheng et al., 2023] Cheng, Y.-C., Lee, H.-Y., Tulyakov, S., Schwing, A., and Gui, L. (2023). Sdfusion: Multimodal 3d shape completion, reconstruction, and generation.

[Chetverikov et al., 2002] Chetverikov, D., Svirko, D., Stepanov, D., and Krsek, P. (2002). The trimmed iterative closest point algorithm. volume 16, pages 545– 548 vol.3.

[Decatur et al., 2022] Decatur, D., Lang, I., and Hanocka, R. (2022). 3d highlighter: Localizing regions on 3d shapes via text descriptions.

[Erkoç et al., 2024] Erkoç, Z., Gümeli, C., Wang, C., Nießner, M., Dai, A., Wonka, P., Lee, H.-Y., and Zhuang, P. (2024). Preditor3d: Fast and precise 3d shape editing.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

[Frank et al., 2020] Frank, J., Eisenhofer, T., Schönherr, L., Fischer, A., Kolossa, D., and Holz, T. (2020). Leveraging frequency analysis for deep fake image recognition.

[Fu et al., 2023] Fu, R., Zhan, X., Chen, Y., Ritchie, D., and Sridhar, S. (2023). Shapecrafter: A recursive text-conditioned 3d shape generation model.

[Gao et al., 2026] Gao, K., Gao, Y., He, H., Lu, D., Xu, L., and Li, J. (2026). Neural radiance fields in 3d vision: A comprehensive review. *Computational Visual Media*, page 1–55.

[Gao et al., 2024] Gao, R., Deng, K., Yang, G., Yuan, W., and Zhu, J.-Y. (2024). Tactile dreamfusion: Exploiting tactile sensing for 3d generation.

[Gutflaish et al., 2025] Gutflaish, E., Kachlon, E., Zisman, H., Hacham, T., Sarid, N., Visheratin, A., Huberman, S., Davidi, G., Bukchin, G., Goldberg, K., and Mokady, R. (2025). Generating an image from 1,000 words: Enhancing text-to-image with structured captions.

[Haque et al., 2023] Haque, A., Tancik, M., Efros, A. A., Holynski, A., and Kanazawa, A. (2023). Instruct-nerf2nerf: Editing 3d scenes with instructions.

[Hegazy, 2024] Hegazy, M. (2024). Diversity of thought elicits stronger reasoning capabilities in multi-agent debate frameworks. *ArXiv*, abs/2410.12853.

[Hendrikx et al., 2013] Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1).

[Jiang, 2024] Jiang, C. (2024). A survey on text-to-3d contents generation in the wild.

[Jincheng et al., 2026] Jincheng, J., Wu, Y., Cai, Y., and Liu, L. (2026). Craftmesh: High-fidelity generative mesh manipulation via poisson seamless fusion.

[Kato et al., 2020] Kato, H., Beker, D., Morariu, M., Ando, T., Matsuoka, T., Kehl, W., and Gaidon, A. (2020). Differentiable rendering: A survey.

[Li et al., 2025] Li, P., Ma, S., Chen, J., Liu, Y., Zhang, C., Xue, W., Luo, W., Sheffer, A., Wang, W., and Guo, Y. (2025). Cmd: Controllable multiview diffusion for 3d editing and progressive generation.

[Liang et al., 2025] Liang, Z., Sun, J., and Ma, H. (2025). An llm-lvlm driven agent for iterative and fine-grained image editing.

[Liepa, 2003] Liepa, P. (2003). Filling Holes in Meshes. In Kobbelt, L., Schroeder, P., and Hoppe, H., editors, *Eurographics Symposium on Geometry Processing*. The Eurographics Association.

[Lin et al., 2023] Lin, C.-H., Gao, J., Tang, L., Takikawa, T., Zeng, X., Huang, X., Kreis, K., Fidler, S., Liu, M.-Y., and Lin, T.-Y. (2023). Magic3d: High-resolution text-to-3d content creation.

[Lindenmayer, 1968] Lindenmayer, A. (1968). Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299.

[Liu et al., 2024] Liu, F.-L., Fu, H., Lai, Y.-K., and Gao, L. (2024). Sketchdream: Sketch-based text-to-3d generation and editing.

[Loper and Black, 2014] Loper, M. M. and Black, M. J. (2014). Opendr: An approximate differentiable renderer. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 154–169, Cham. Springer International Publishing.

[Ma et al., 2025] Ma, C., Li, Y., Yan, X., Xu, J., Yang, Y., Wang, C., Zhao, Z., Guo, Y., Chen, Z., and Guo, C. (2025). P3-sam: Native 3d part segmentation.

[Müller et al., 2023] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. (2023). *Procedural Modeling of Buildings*. Association for Computing Machinery, New York, NY, USA, 1 edition.

[Palfinger, 2022] Palfinger, W. (2022). Continuous remeshing for inverse rendering. *Computer Animation and Virtual Worlds*, 33(5):e2101.

[Park et al., 2025] Park, J., Murdivien, S. A., Choi, K., Mun, D., and Um, J. (2025). Generative 3d appearance design: A survey of generation, segmentation and editing by artificial intelligence. *Journal of Computational Design and Engineering*, 13(1):1–23.

[Pixune Studios, 2026] Pixune Studios (2026). How much does it cost to make a game? (updated 2026). `https://pixune.com/blog/how-much-does-it-cost-to-make-a-game/`. Accessed: 2026-05-05.

[Poole et al., 2022] Poole, B., Jain, A., Barron, J. T., and Mildenhall, B. (2022). Dreamfusion: Text-to-3d using 2d diffusion.

[Prusinkiewicz and Lindenmayer, 1990] Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.

[Raj et al., 2023] Raj, A., Kaza, S., Poole, B., Niemeyer, M., Ruiz, N., Mildenhall, B., Zada, S., Aberman, K., Rubinstein, M., Barron, J., Li, Y., and Jampani, V. (2023). Dreambooth3d: Subject-driven text-to-3d generation.

[Rezatofighi et al., 2019] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., and Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression.

[Ritter, 1990] Ritter, J. (1990). *An efficient bounding sphere*, page 301–303. Academic Press Professional, Inc., USA.

[Rusu et al., 2009] Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217.

[Schrepp et al., 2017a] Schrepp, M., Hinderks, A., and Thomaschewski, J. (2017a). Construction of a benchmark for the user experience questionnaire (ueq). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4:40–44.

[Schrepp et al., 2017b] Schrepp, M., Hinderks, A., and Thomaschewski, J. (2017b). Construction of a benchmark for the user experience questionnaire (ueq). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(4):40 – 44.

[Shuai et al., 2024] Shuai, X., Ding, H., Ma, X., Tu, R., Jiang, Y.-G., and Tao, D. (2024). A survey of multimodal-guided image editing with text-to-image diffusion models.

[Sorkine and Alexa, 2007] Sorkine, O. and Alexa, M. (2007). As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, page 109–116, Goslar, DEU. Eurographics Association.

[Sorkine et al., 2004] Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., and Seidel, H.-P. (2004). Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, page 175–184, New York, NY, USA. Association for Computing Machinery.

[Umeyama, 1991] Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380.

[Voynov et al., 2022] Voynov, A., Aberman, K., and Cohen-Or, D. (2022). Sketch-guided text-to-image diffusion models.

[Wang et al., 2022] Wang, J., Lin, J., Yu, Q., Liu, R., Chen, Y., and Yu, S. X. (2022). 3d shape reconstruction from free-hand sketches.

[Wang et al., 2023] Wang, P., Liu, Y., Chen, Z., Liu, L., Liu, Z., Komura, T., Theobalt, C., and Wang, W. (2023). F^2-nerf: Fast neural radiance field training with free camera trajectories.

[Yan et al., 2025] Yan, K., Zhang, C., Speierer, S., Cai, G., Zhu, Y., Dong, Z., and Zhao, S. (2025). Image-space adaptive sampling for fast inverse rendering. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, SIGGRAPH Conference Papers '25, New York, NY, USA. Association for Computing Machinery.

[Yang et al., 2022] Yang, B., Gu, S., Zhang, B., Zhang, T., Chen, X., Sun, X., Chen, D., and Wen, F. (2022). Paint by example: Exemplar-based image editing with diffusion models.

[Yang et al., 2024] Yang, Y., Huang, Y., Guo, Y.-C., Lu, L., Wu, X., Lam, E. Y., Cao, Y.-P., and Liu, X. (2024). Sampart3d: Segment any part in 3d objects.

[Zhang et al., 2023] Zhang, L., Rao, A., and Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models.

[Zhou et al., 2018] Zhou, Q.-Y., Park, J., and Koltun, V. (2018). Open3d: A modern library for 3d data processing.

[Öcal et al., 2024] Öcal, B. M., Tatarchenko, M., Karaoglu, S., and Gevers, T. (2024). Sceneteller: Language-to-3d scene generation.