care, judgment, dexterity

CRAEFT

Craft-specific action simulations

Project Acronym	Craeft
Project Title	Craft Understanding, Education, Training, and Preservation for Posterity and Prosperity
Project Number	101094349
Deliverable Number	3.1
Deliverable Title	Craft-specific action simulations
Work Package	3
Authors	Xenophon Zabulis, Panagiotis Koutlemanis, Ioanna Demeridou, Nikolaos Partarakis, Peiman Fallahian, Nikolaos Nikolaou
Number of pages	74

This project has received funding from the European Commission, under the Horizon Europe research and innovation programme, Grant Agreement No 101094349.
http://www.craeft.eu/

Executive summary

Section 1 outlines the scope and objectives of this document, providing an overview of the content covered.

Section 2 reviews relevant literature in the field of craft-related simulations. The review begins with an exploration of techniques for simulating the interaction of light with objects, followed by a discussion on action simulations, particularly those that utilize physics to predict the outcomes of mechanical actions and those specific to traditional crafts.

Section 3 introduces and demonstrates a visualization toolbox built on the Mitsuba 3 physics-based renderer. This toolbox streamlines the process of scene composition by defining elements such as physical entities, lights, observation viewpoints, and virtual cameras in simple terms. Using this technology, we can realistically simulate the appearance of artefacts and materials based on their composition. Additionally, it enables the creation of presentation videos that simulate the appearance of objects in various environments and from different viewpoints. Integration with the physics-based simulators developed in Craeft allows for the realistic presentation of material-specific crafting actions.

Section 4 details a software interface to the PhysX physics-based simulator, which employs GPU parallelization to simulate physical phenomena in real-time. This interface manages the representation of physical entities within a simulated scene and utilizes the PhysX engine to execute prescribed actions. It also provides bindings for real-time manipulation of tools within the scene using a keyboard, mouse, VR, or haptic controller. Using this technology, we have developed a set of craft-specific simulators focused on creating solids by revolution, with specialized applications for pottery, woodturning, and glassblowing.

Section 5 presents our approach to achieving physically realistic and real-time simulations. Given the high computational demands of such simulations, real-time execution is typically infeasible. To address this, we propose an artificial intelligence (AI) approach that learns to approximate simulation results in real-time. This AI is trained on standard simulation data and their parameter descriptions, enabling it to approximate results even with limited training data. The approach is integrated with the PhysX physics engine, allowing for real-time approximation of original simulations.

Section 6 showcases the application of the tools we developed, resulting in simulators that produce digital and realistic objects by replicating the crafting processes and physical constraints associated with their creation. These simulators are intended for use in the Design Studio to create designs of objects that are physically realizable and that follow the production procedure that is needed to realise them. Their implementation is further demonstrated in Deliverable D5.1, “Craft Design Revisited,” with the technical aspects detailed in this document.

Section 7 concludes the deliverable with a summary of findings and provides directions for future work.

Document history

Date	Author	Affiliation	Comment
19/07/2024	Xenophon Zabulis	FORTH	Drafting
19/07/2024	Xenophon Zabulis	FORTH	Drafting
19/07/2024	Xenophon Zabulis	FORTH	Drafting

Abbreviations

AR	Augmented Reality
CAP	CRAEFT Authoring Platform
CH	Cultural Heritage
CHI	Cultural Heritage Institutions
FEM	Finite Elements Method
HCD	Human Centred Design
HCI	Heritage Computer Interaction
ICH	Intangible Cultural Heritage
IPR	Intellectual Property Rights
MET	Material Engagement Theory
MoCap	Motion Capture
NURB	Non-uniform rational B-spline modelling

Table of contents

1. Introduction

In recent years, the convergence of traditional craftsmanship with advanced computational simulations has unveiled new opportunities for preserving, exploring, and innovating within craft practices. As these technologies progress, they present unprecedented possibilities for simulating and visualising intricate interactions between materials, tools, and techniques, enriching our understanding and the creative potential within the craft domain.

This deliverable aims to contribute to the rapidly evolving field of craft simulations by providing tools that empower practitioners to push the boundaries of what is achievable in both digital and physical contexts. Our work not only preserves the essence of traditional craftsmanship but also lays the groundwork for future innovations in the craft industry.

The following sections detail our efforts in developing and refining a comprehensive suite of tools and methods designed to simulate craft-related activities with high fidelity. Our objective is to bridge the gap between traditional craftsmanship and modern digital tools, enabling artisans, designers, and researchers to explore the nuances of craft techniques within a virtual environment. We strive to offer tools that replicate not only the visual and physical properties of crafted objects but also predict the outcomes of specific crafting actions with remarkable precision.

Section 2 critically reviews existing literature on craft-related simulations, focusing particularly on the interaction of light with objects and the simulation of mechanical actions within traditional crafts.

Section 3 introduces our visualisation toolbox, a powerful tool based on the Mitsuba 3 physics-based renderer. This toolbox simplifies scene composition and enables realistic simulations of materials and artefacts. It serves as an essential infrastructure tool, particularly within the Design Studio, facilitating material-specific and photorealistic visualisations. Additionally, the toolbox supports environment-specific visualisations, allowing prospective clients to envision how a particular artefact would appear in a specific setting, such as their home.

Section 4 discusses the integration of a software interface with the PhysX physics engine, enabling real-time simulation of physical phenomena and the development of craft-specific simulators. This middleware software, known as the action simulation toolbox, is crucial for Craeft because traditional scientific FEM simulations are computationally intensive and cannot be executed interactively in real time. Although PhysX does not simulate all the phenomena and material properties as comprehensively as a FEM multi-physics simulator, our toolbox implements three elementary types of actions according to the action classification taxonomy outlined in D2.1, "Action and Affordance Modelling," using the capabilities available in PhysX. Similar to the visualisation toolbox, the action simulation toolbox simplifies the definition of crafting scenes, including workspace, workpiece(s), and tools. It maintains a 3D representation of the scene and provides software bindings, enabling users to manipulate tools in real-time using various computer controllers, whether conventional (i.e., mouse, keyboard) or immersive (i.e., VR and haptic controllers). Moreover, the toolbox utilises PhysX to provide real-time 3D visualisations of the scene.

Section 5 presents an innovative approach to achieving real-time simulations through artificial intelligence, which approximates the results of computationally intensive simulations. Our method uses FEM simulation results, obtained after hours of computation, to train an AI network capable of approximating these results in real time. This approach is integrated with PhysX to exploit its built-in functions and fine-tune its parameters, allowing it to approximate the actions described in the training data for a given material. This technique is incorporated into the action toolbox, enabling these approximations to be used in real-time, interactive applications.

In Section 6, we apply the aforementioned tools to create craft and process-specific simulations that generate virtual artefacts using realistic manufacturing processes. This approach is vital for Craeft as it (a) ensures that users design physically realisable artefacts, and (b) provides a process description that can be translated into instructions for practitioners. Our initial applications of this approach include simulations for moulded and sculpted solids, stained glass windows, lamps, cane-working products, metal engravings, and glazed pottery.

Finally, Section 7 concludes the deliverable with a summary of our findings and outlines potential directions for future research and development.

2. Related work

We classify the reviewed literature into two classes. The first regards the interaction of light with objects and the way that this can be approximated and simulated in the computer. The second regards a review of action simulations, specifically those that use physics to predict the results of mechanical actions and those related to traditional crafts.

2.1. Appearance simulation

The work reviewed in this subsection relates to representing and simulating the geometrical and material properties of objects that are relevant to their appearance.

2.1.2. Digital representation of appearance

To date, CH research has focused on the visualization of existing artefacts that have been digitized (or, following Computer Vision jargon, "reconstructed"). This type of digitization is based on scanning and modelling technologies, which have become valuable tools in the photographic and 3D documentation of cultural heritage artefacts. High-resolution 3D laser scanning and photogrammetry are the most commonly used to capture precise geometrical details of artefacts as well as their photorealistic appearance. A recent and comprehensive review of 3D scanning technologies and approaches can be found in The Mingei Handbook on Heritage Craft representation and preservation, at Step 3 "Craft recording", in Section 3.2, "Digitisation of endurant assets".

In addition, other digitization methods focus on the material composition of artefacts and, in this case, employ imaging techniques that reveal properties imperceptible to humans, because they identify features that are visible outside the visible spectrum (i.e. IR and UV illumination). In [1], advances in multi-spectral and hyperspectral imaging for archaeology and art conservation are reviewed, informing on their use in the identification of the materials used and contributing to their conservation and preservation.

Despite these technologies, most of the visualization techniques focus on Lambertian, or "matte", surfaces. The rendering of shiny, transparent and translucent materials is rendered by explicitly specifying the 3D model regions that have this property. The simulation of the appearance of these materials follows some simple rules but does not simulate the interaction of light with individual types of materials.

During the last few years, Neural Radiance Fields (NeRFs) [5, 6] were introduced as a method for synthesizing novel views of complex 3D scenes from a sparse set of 2D images. They leverage deep learning to model the volumetric scene representation by encoding the scene into a neural network, which can then be queried to render new views. This approach has gained significant attention due to its ability to produce highly realistic images and its applications in various fields, including cultural heritage visualization. NeRFs can capture the appearance of challenging materials, however, they cannot be used to render scenes with different properties (i.e., illumination and décor) to the scene in which they were imaged.

2.1.1. Photorealistic rendering

The visualization of cultural heritage artefacts has seen significant advancements in recent years, driven by technological innovations and interdisciplinary research. With the support of 3D graphics and immersive presentation technologies (i.e., AR, VR) interactive and educational experiences have been produced, making cultural heritage more accessible to broader audiences. Real-time rendering techniques are essential for interactive applications such as VR and AR. These techniques leverage the parallel processing capabilities of GPUs to render scenes efficiently [2].

Photorealistic rendering aims to create images that are indistinguishable from real photographs. This technique is valuable for producing accurate representations of artefacts. In [3], the principles of global illumination, which model the complex interplay of light in a scene to achieve photorealism are presented. In [4], image-based lighting techniques are employed using photographs of real-world lighting conditions to illuminate 3D models, enhancing the realism of renderings.

In terms of material-specific appearance simulation, a technical approach is proposed that is based on advanced, physically-based rendering software infrastructure called “Mitsuba 3” [16], which has been for research and educational purposes. This infrastructure simulates the interaction of light with materials in a highly accurate manner, enabling the creation of photorealistic images.

2.2 Activity simulation

Interactive, physics-based simulations have become vital tools for the study and preservation of traditional crafts and mechanical operations. These simulations employ computational modelling of the physical behaviour of materials and mechanisms, aiming to provide immersive and educational experiences.

2.2.1. Physics-based simulation

Physics-based simulations focus on the physical behaviour of mechanical systems—including structures, materials, and dynamics—and have been pivotal in various fields of engineering and science. The Finite Element Method (FEM), is foundational to many mechanical simulations and was popularized by Clough and others in the 1950s [30, 31]. FEM allows the simulation of complex mechanical systems by breaking them down into smaller, simpler parts known as finite elements. Initially, FEM was applied primarily to civil engineering problems, such as analysing the stress and strain in bridges and buildings [32]. The aerospace and automobile industry played a crucial role in driving the early adoption of mechanical simulations [33].

The commercialization of FEM software made these tools more widely accessible to engineers across various industries. Notable commercial FEM software included ANSYS [34, 37], NASTRAN [35], and Abaqus [36]. In parallel, Computational Fluid Dynamics (CFD) emerged as a critical tool for simulating fluid flows around mechanical structures, such as aircraft wings and automotive bodies, with software suites like FLUENT and CFX providing instrumental in advancing this field [38, 39].

Traditionally, generic and interactive simulations are based on “physics engines” [10]. Physics engines are software libraries which simulate the physical behaviour of objects under prescribed dynamic and kinematic configurations. In their majority, they simulate rigid objects, their potential collisions, and their behaviour thereafter [11]. Nevertheless, if real-time performance is required the structural complexity of involved models has to be low (i.e., their 3D meshes should be comprised of few polygons.

Realistic simulations of actions upon materials are found in the field of scientific simulation. Finite Element Analysis (FEA) [87, 88] is a numerical technique that utilizes the Finite Element Method (FEM) to simulate and analyse the behaviour of physical systems. FEA is the *de facto* standard in state-of-the-art physical simulation. The idea behind the FEA is to divide complex physical systems into smaller, simpler, and very local (or finite) elements. The behaviour of each element is predicted by mathematical equations that describe the physical laws governing an action.

Although widely adopted in modern mechanics and engineering, scientific simulation has been not widely applied in the domain of crafts. In [89], the formation of knots is studied using FEM. Mechanical models for fibres are proposed in [90] that account for elongation, bending and torsion forces, and the frictional contacts between them. In [91], the metalworking processing is studied to understand the quenching process and results of a computer simulation based on metallo-thermo-mechanics are presented to know how the temperature, metallic structure and stress/distortion vary in the process.

2.2.2. Game engines based on physics simulation

The integration of physics simulation into game engines has significantly advanced over the past few decades, evolving from basic collision detection to sophisticated simulations of rigid bodies, soft bodies, fluids, and other physical phenomena. In the early stages of game engine development, physics simulations were rudimentary and limited to basic collision detection. An example of this is id Tech 1, which featured simple collision handling without realistic physics [40]. More advanced physics simulations enabled interactions between game objects, especially in the context of rigid body dynamics [41].

As physics engines matured, they were increasingly integrated into mainstream game engines like Unreal Engine and Unity. These engines not only supported rigid body dynamics but also expanded to more complex simulations, including cloth, fluid dynamics, and destructible environments [42]. The Unreal and Unity 3D engines were among the first to incorporate middleware physics engines like PhysX [43].

The application of physics simulations in game engines has expanded beyond traditional gaming into areas such as VR and AR, where physics-based interactions are essential for creating immersive and engaging experiences [44]. Additionally, game engines like Unreal Engine are increasingly used in virtual production for film and animation, where real-time physics simulations enable dynamic scene creation [45]. Furthermore, these engines are employed in serious games and simulations for engineering, architecture, and medical training, where accurate physics simulations are necessary for realistic virtual environments [46].

Balancing realism with performance remains a significant challenge, particularly on hardware-limited platforms like consoles and mobile devices. Techniques such as Level of Detail (LOD) for physics, hardware acceleration, and multi-threading are continually optimized to address these challenges [47]. Additionally, the demand for real-time simulations is growing, especially in VR/AR and multiplayer online games, necessitating more efficient and scalable physics simulations. Future developments in game engines may also involve the integration of AI to dynamically adjust physics simulations, thereby creating more adaptive and responsive environments [48].

2.2.3. Craft-specific simulations

Traditional crafts involve complex interactions between materials and tools, often requiring a deep understanding of physical properties and manual skills. Interactive, physics-based simulations can accurately replicate these processes, offering new ways to preserve and teach these crafts.

2.2.3.1. Presentation of crafting actions

In [54], a mobile Augmented Reality (AR) system that superimposes 3D craft objects in space is proposed. The system triggers the virtual demonstration of their usage using a multi-touch surface. In [55], a Head Mounted Display is used to present traditional craft objects with high presence and absorption. In [56], an AR system augments a given physical object with audio and visual digital assets, relevant to its making. A user study is conducted to bring insight into methods of combining virtual and physical materials, to present a narrative located in the decoration of the object.

Virtual Reality (VR) and handheld controllers are used for more realistic virtual handling and interaction with the presented 3D craft objects. In [57], VR demonstrations for two crafts are provided, for the production of two Greek traditional alcoholic beverages. The demonstrations employ pre-recorded interactions with tools and machines, which in VR can be viewed from any viewpoint. In [58], visitors are immersed in a VR environment where they can perform some indicative woodworking tasks, in the context of introducing the usage of dovetailing carpentry tools. The simulation of potential interactions is pre-recorded and shown as animations.

2.2.3.2. Textiles

Textile weaving simulations regard the patterns and the interweaving of threads. In [7], a system for the interactive design of woven structures is proposed, enabling the simulation of the weaving process and the visualization of the resulting fabric in real-time.

A broad range of studies exist on the mechanical characterisation of textiles (see [92-95] for reviews). Several pertinent works also focus on how textiles are to deform and distort when worn, e.g. [96, 97]. The most relevant work to the purposes of this work is TexGen [99], open-source software for modelling the geometry of textile structures, as well as including textile mechanics, permeability and composite mechanical behaviour. In the computations pertinent to the manufacturing of textiles, we use the TexGen simulator to model the 3D structure of fabrics.

Works that predict the visual appearance of crafted artefacts are found in the textile industry. Prominent examples are WeaveIt [78], Weaving Design Software [79], ArahneWeave [80], pixeLoom [81], WeavePoint [82], and WIF Visualizer [83]. The 3D Knitting Simulation [84] was developed for flat and circular knitting technology. Given the fabric design, the simulator creates realistic visualisations for Jacquard Raschel, Multibar Lace, and warp-knit fabrics. More relevant to handcrafted textiles a physics-based heuristic model is used in [85] to predict the visual results of painting on fabric, using thin-brush dyeing. The simulator focuses on modelling 2D fluid simulation on fabrics to reduce computational burden. The dyeing algorithm is based on an ink-wash painting algorithm [86].

2.2.3.3. Robotic re-enactment

Automated manufacturing of crafting products was initiated in the Industrial Revolution and textile manufacturing. Today robotic automation is the norm in manufacturing industries [59] for precisely predefined tasks. In this subsection, work that attends to the recreation of human crafting motion is reviewed.

Mimicking human freehand motion using robots has been mainly studied for carving tasks. As robotic motion has fewer and different degrees of freedom than humans, for a robot to achieve the same tool movement as a human, it is required to convert human kinematics for the available robotic embodiment. In [60], human movements are analysed into their principal components and then encoded to robot kinematic instructions. In [61-63], these components were approximated through machine learning.

Taking an inverse approach, other studies focused on achieving the same result as human actions. Studies of carving strokes were conducted in [60, 64], to establish a correspondence between human results and robotic motion that approximates the same results. In [65], a step forward was made by adding some sensor-based robotic autonomy in the construction of wooden structures.

2.2.3.4. Games

Some video games provide creative interaction by replicating basic crafting aspects in the contexts of virtual building and decoration.

Creating virtual pottery using wheel-throwing and subsequent decoration is found in several games engaging creativity (e.g. 3D Pottery [66], Pottery Master [67], and Pottery Simulator [68]), albeit not exhibiting high levels of realism, nor addressing practical constraints.

In the adventure game genre [69], the prerequisite of crafting or recipe materials is addressed, by requiring them to be available for the execution of an action. Although recipes do not contain a simulation of how materials are treated, they provide constraints. A central concept in these games is the “recipe”, or the representation of the knowledge necessary to transform a collection of needed ingredients (materials) into a new object. For example, a recipe for a pickaxe specifies two twigs and two flints as the necessary materials. Certain recipes regard only specific types of materials, such as a recipe for tailors that transform fibres into fabric or a recipe for blacksmiths that transform bulk metal into a sword.

Using human motion in gaming interaction has proliferated since the Wii controller. The Knitting Simulator 2014 [70], requires the manipulation of controllers that resemble knitting needles. The usage is simplified as needle motion is only used to advance a knitting animation. In [71], a VR controller is used to edit solids by revolution in wood-turning lathe crafting simulation. An architecture for integrating the Unity game engine as a platform for craft simulation is proposed in [72].

PhysX [73] is a physics engine middleware SDK for the development of gaming applications that are based on hardware acceleration to achieve real-time performance. It supports rigid and body dynamics and volumetric fluid simulation. However, it cannot simulate the generality of phenomena in the context of this work, as it does not support all of the material properties and damage models of interest in this work.

2.2.3.5. Serious games

Woodwork Simulator [74] recreates the experience of working in a carpenter’s workshop. It provides reasonable approximations of the effect of virtual saws, drills, glue, chisels, and sandpaper on virtual wood. Educational uses are found in workspace geography and safety, training in the use of tools, and the planning of woodworking processes that implement specific designs.

In [75], a blacksmith’s forge is simulated in VR providing simplified tool interaction that shapes metallic pieces parts using 3D controllers; the crafted structures can be 3D printed.

Counting and calculating tasks are intrinsic to the weaving of fabrics and wicker. In [76], these capacities are trained to make calculus for young students more interesting, intuitive, and educational on Native American Heritage.

In [77], glasswork actions are simulated to accustom to the weight, balance, and handling of a real blowpipe performed at a real glassblower's bench. A Mixed Reality system tacks human hands and illustrates the user against exemplar hand movements for that action.

Pottery simulations involve the manipulation of clay on a rotating wheel. In [8], a haptic simulation of pottery-making provides users with tactile feedback, enhancing the realism of the experience and enabling the replication of traditional techniques.

Metalworking processes such as forging, casting, and machining involve significant physical transformations. In [12], deformable models that simulate the plastic deformation of materials, which are essential for creating realistic simulations of metalworking operations were developed.

Interactive simulations for mechanical assembly enable the virtual assembly of mechanical systems, providing insights into the assembly process and the interactions between components. In [9], Fang and Chang (2010) developed a system for simulating the assembly of mechanical parts, enabling users to explore different assembly sequences and detect potential issues.

3. Visualisation toolbox

To achieve a generic way of simulating the appearance of artefacts and materials, we created a programmable API. This way, the functionalities of the developed utilities are available to multiple utilities developed in Craeft, as well as to third-party developers.

Rendering Lambertian (or “matte”, or “diffuse”) surfaces under any given illumination and décor has been commonplace in Computer Graphics for decades. Today, several rendering libraries are available with OpenGL and Open3D the most widely used open-source ones. In these libraries, the rendering of textured matte surfaces has been optimised. However, more challenging phenomena, such as light absorption and scattering that occur in shiny, semi-transparent, and translucent materials are not realistically modelled. To solve this problem we use the Mitsuba 3 renderer (see Section 2). Besides conventional, texture-based renderings this provides us with a realistic simulation of the appearance of:

1) scanned artefacts made from challenging (shiny, translucent, and transparent) materials and
2) artefact designs that are modelled in 3D for which a designer wishes to predict the way that they would appear.

However, as this infrastructure is highly technical and research-based, we have developed a wrapper that simplifies its use for broader audiences. Specifically, we created a toolbox that simplifies the rendering of images and videos of arbitrary scenes, composed of 3D models and light sources, made from virtually any material, given its physical properties. In this way, we can create realistic renderings of craft artefacts. Moreover, we can simulate how these artefacts would appear when placed in an arbitrary environment. The latter capability is useful for realistic previews of craft products and typical environments, in which the buyers would place or use them. The toolbox can create both images and videos; it can be found here, along with usage instructions: https://github.com/andriani-st/mitsuba3-util/tree/GeneralUtil_Refactor.

The toolbox is operated using the Python scripting language and we intend to integrate it in the Design Studio. To facilitate the use of the toolbox we created a few wrapper applications that simplify the creation of image and video previews and aid their design.

Our toolbox is an integration of Mitsuba3 and its purpose is to simplify the usage of Mitsuba3 and automate some of its functionality in a more user-friendly way. Utilising Mitsuba's state-of-the-art physically-based rendering (PBR) techniques, our software ensures that the light transport and material interactions are simulated with high fidelity. This allows for realistic previews of objects, capturing the nuances of reflections, refractions, subsurface scattering, and surface textures for materials such conventional and challenging materials such as glass, metal, plastic, wax, marble, and wood.

The rationale for the toolbox's functional design is the following. To use the toolbox only a configuration file is needed as input. This file describes the scene elements (objects, materials, lights) the type of requested output (image or video) and its properties (resolution, field-of-view, camera trajectory etc). In other words, the toolbox is utilised as a “compiler” which receives configuration files and outputs visual media. Our intention behind this choice is to make the toolbox available to multiple user interfaces, that serve different purposes and applications: all that is needed is for the interface to generate the scene file and call the toolbox. Using this approach, we created a few utility applications that specialise in specific craft products.

Using the toolbox, users can define their scenes and adjust illumination settings to match specific environments. Whether it’s natural sunlight, indoor lighting, or complex studio setups, the software

provides the flexibility to recreate the desired lighting conditions for the most accurate visualisation. Using 360 photography users can also import their environments in the appearance simulation.

A software interface allows programmers to adjust parameters, experiment with different materials, and visualise the results. The toolbox is designed as middleware that can be used by a GUI that implements an interactive design process that enhances creativity and ensures that the final product meets the desired aesthetic and functional requirements.

The toolbox creates high-resolution images and videos that showcase the intricate details of the simulated 3D models. This feature is aimed at presentations, client approvals, and marketing materials, providing a powerful tool to communicate the artistic vision of the practitioner. A basic application simulates the appearance of 3D models when made from different types of material, such as different types and colours of glass. The example below illustrates a glass body made from variations on the type of glass and metal from the same geometry (see Figure 1).

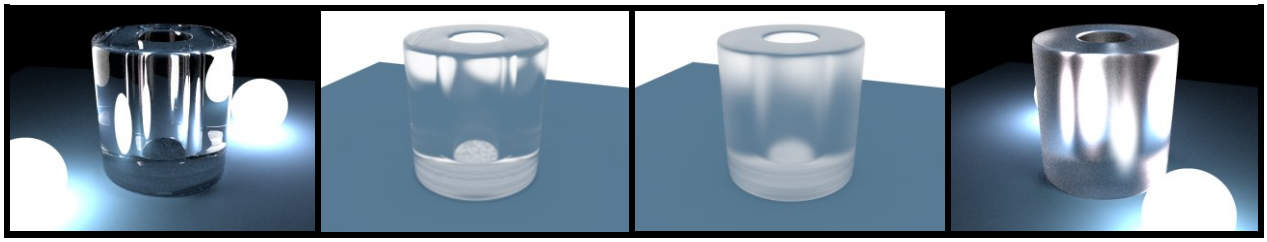


Figure 1. Simulation of materials. The first three illustrate different types of glass. The last simulates a metallic composition.

Next, an application renders a video from a 3D model as if it were placed on a turntable. This way, the user can inspect the appearance of a designed product from all viewpoints around it. In addition, this application can be constrained to rotate the object in a smaller range of angles. In this way, the user can inspect how light interacts with a designed object, such as when rotating a metallic anaglyph under the light to see how light is reflected upon it (see Figure 2).

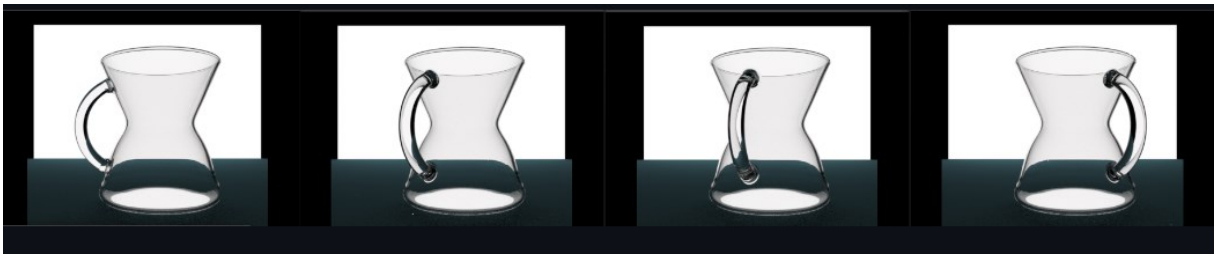


Figure 2. Top: 360 video rendering of a glass body. Bottom: photorealistic rendering of the glass body.

In the following sections, it is described in detail how such a configuration file can be created and its expected outcome.

3.1.1. Objects

To add 3D objects to the scene the objects array must be filled. There are two options for adding objects to the scene.

- Setting the parameters of each object separately by providing a path to the object file for the filename field and setting its material options

- Set the parameters of multiple objects (in case the objects to add to the scene share the same material) by providing a path to a folder with object files for the filename field. If a material that supports colour is selected for the group of objects, a .txt file with RGB colours for each file can be optionally provided for each object.

The objects are placed on the scene according to the coordinates set that is specified in the object file. The objects can be provided in Wavefront OBJ (.obj) and Standford PLY (.ply) formats.

The utility toolbox facilitates the definition of scenes by providing auxiliary objects, specifically a floor (ground plane) and a background plane. The use of these objects is optional. Using the floor option places a rectangle object on the floor. The position of the floor is computed based on the camera pose and the centre of the bounding box that contains the objects added to the scene. The background is implemented as a planar "wall" behind the object. Like the background, the position of the background is computed based on the camera pose and the centre of the bounding box that contains the objects added to the scene.

3.1.2. Materials

All materials are described through their Bidirectional Scattering Distribution Functions (BSDFs) [17], which are mathematical functions that characterize how light is scattered from a surface. that are documented here are supported. Given its BSDF any material can be simulated. To facilitate frequently used materials, the utility toolbox provides some predefined materials. These materials are:

- Glass and rough glass (with optional parameters of their colour) create solid glass objects. Also, thin glass to create hollow glass objects (with the optional parameter of its colour).
- Plastic and rough plastic (with optional parameter its colour).
- Metal and rough metal. All metals are supported by acquiring their BSDFs from [18]. However, if an alloy is needed to be rendered its BSDF function is required as input.

The rendering of the materials is demonstrated in the figures below.

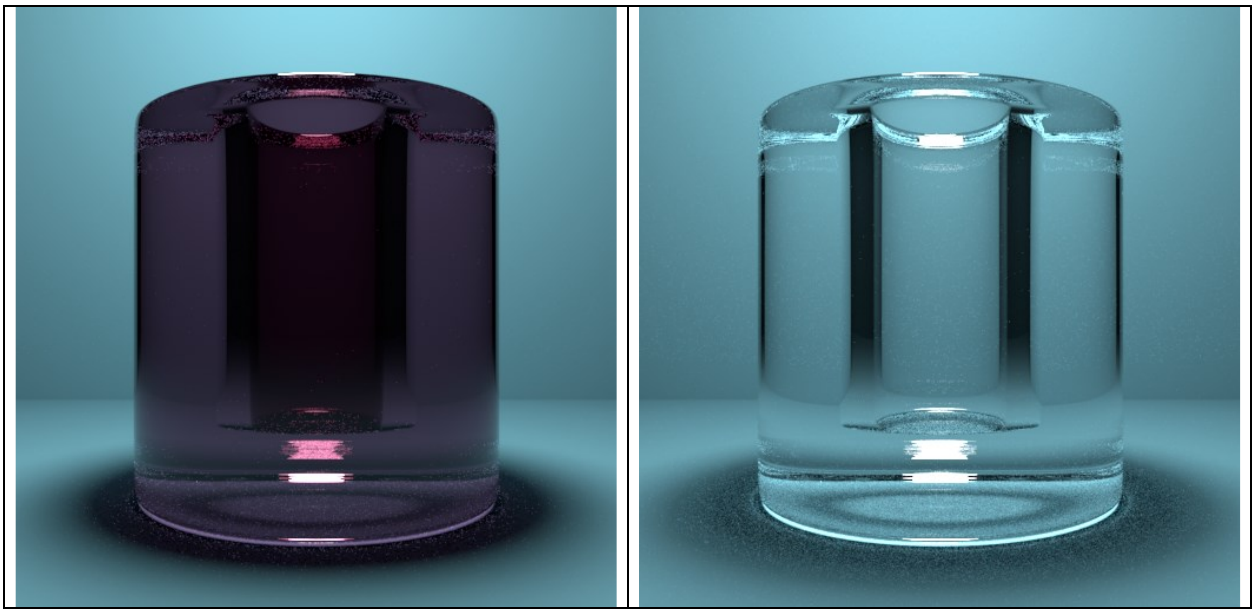


Figure 3. Smooth glass. Coloured (left) and transparent (right).

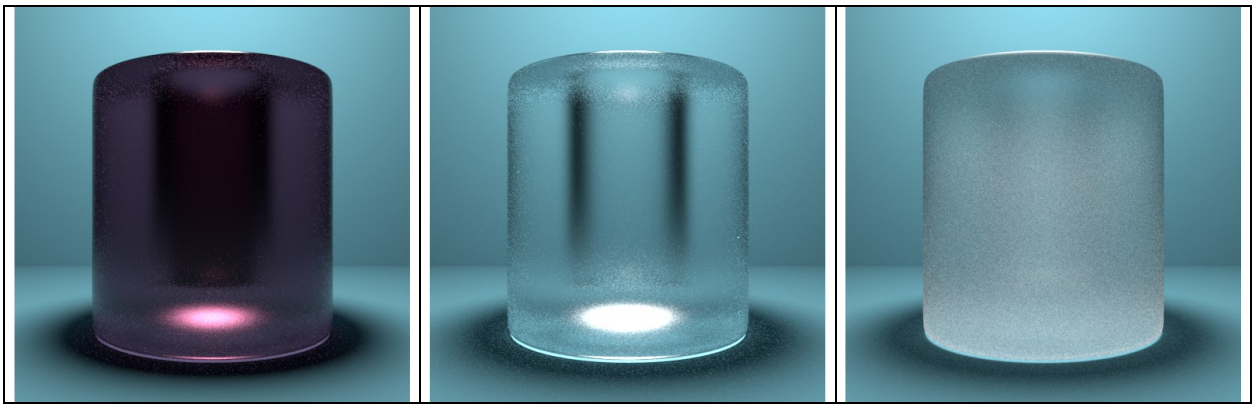


Figure 4. Rough glass.

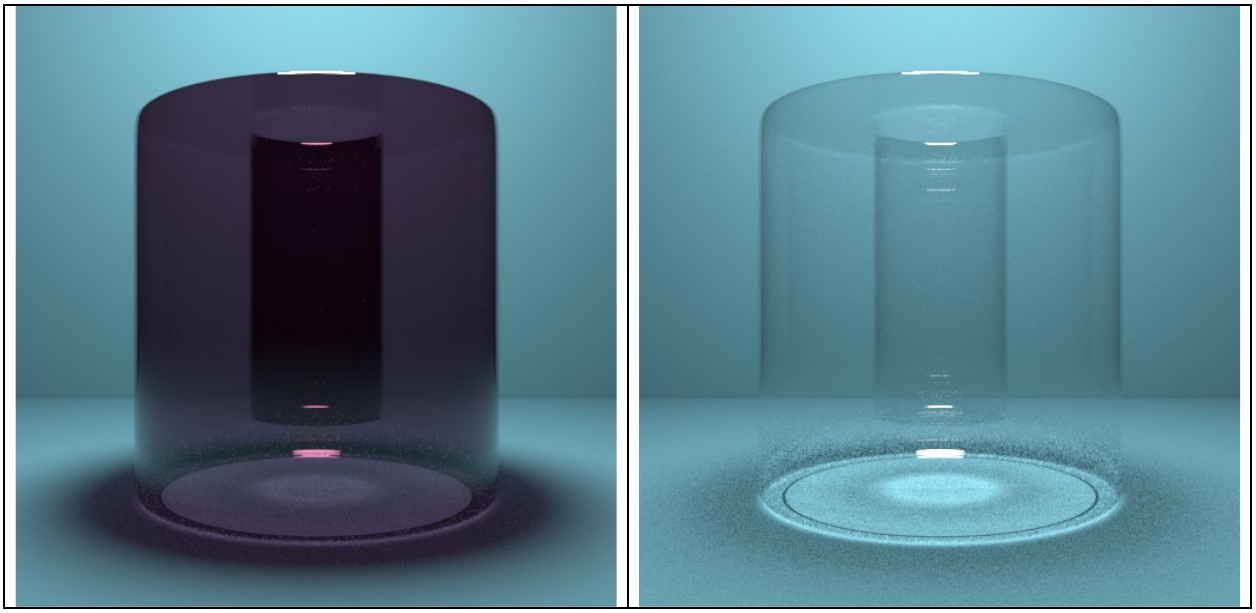


Figure 5. Thin glass. Coloured (left) and transparent (right).

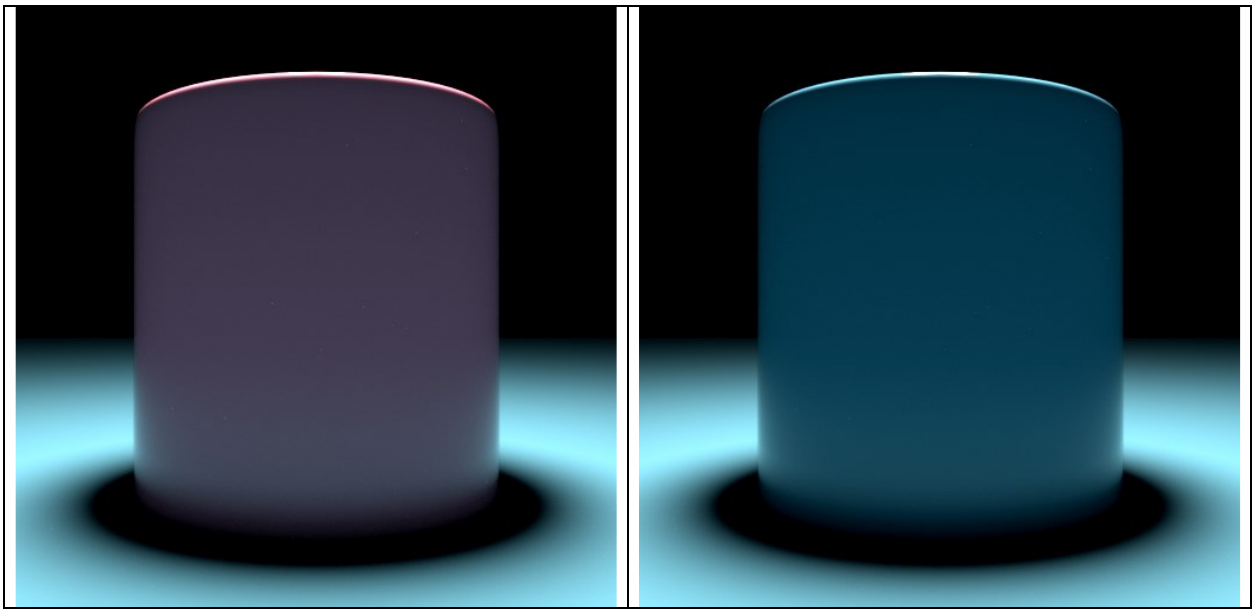


Figure 6. Plastic glass. Transparent (left) and coloured (right).

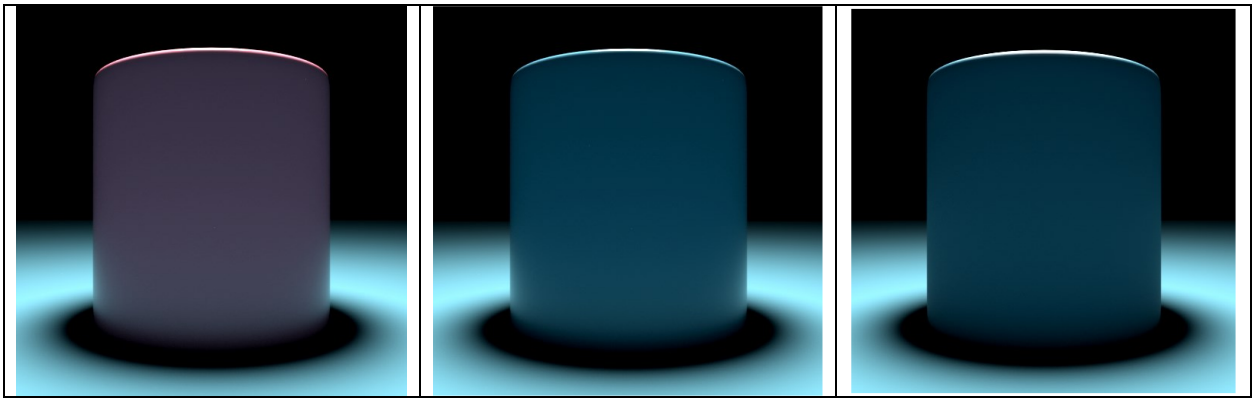


Figure 7. Rough glass.

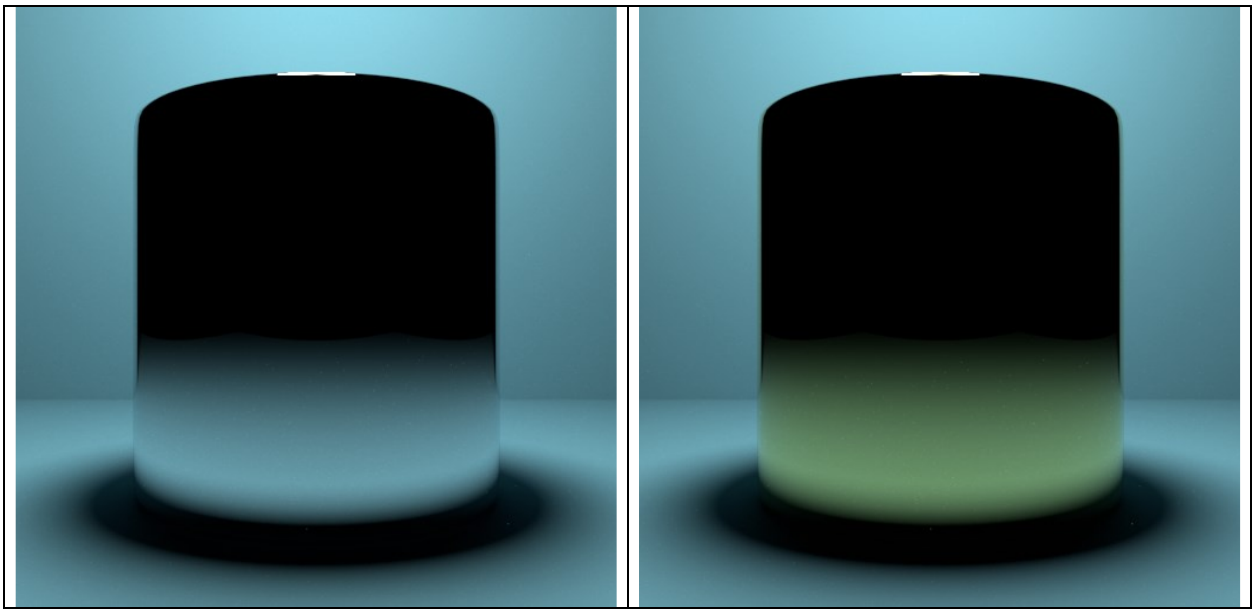


Figure 8. Plastic glass. Transparent (left) and coloured (right).

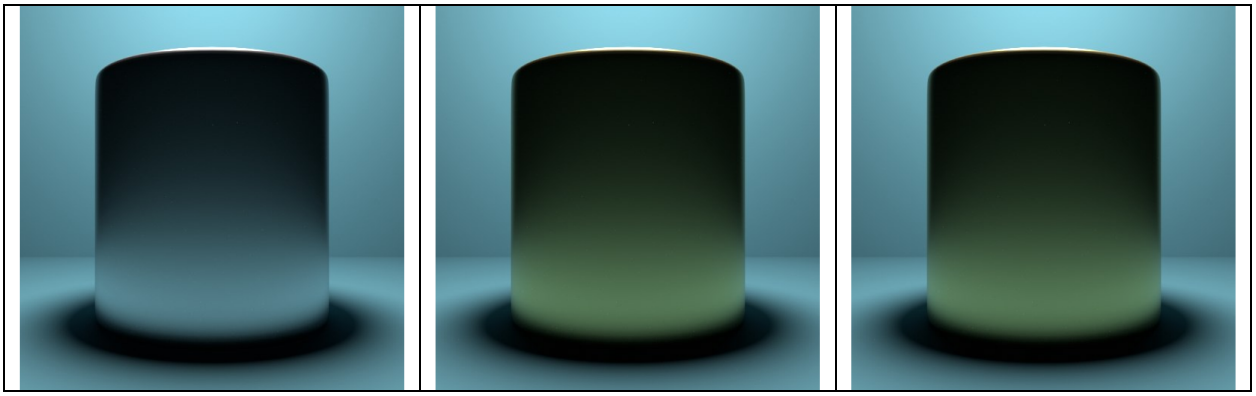


Figure 9. Rough glass.

The differentiation between solid and hollow objects for glass is because glass is a transparent or translucent material. A thin glass (dielectric material) is embedded inside another dielectric (e.g., glass surrounded by air). The interior of the material is assumed to be so thin that its effect on transmitted rays is negligible. Hence, light exits such a material without any form of angular deflection (though there is still specular reflection).

The differentiation between smooth and rough surfaces determines the type of light scattering. For smooth surfaces any given incoming ray of light, the model always scatters into a discrete set of directions, as opposed to a continuum. Moreover, internal scattering can also be simulated with the appropriate parameter setting (see the user's manual on the toolbox website). This means that while some of the diffusely scattered illumination directly refracts outwards, a portion of this energy is reflected from the interior side of the dielectric boundary and remains inside the material for time until it is reflected (and, thus, giving rise to translucent appearance).

3.1.3. Lighting

Lighting is necessary for the simulation to result in visible surfaces[1]. Two lighting types are available:

1. Specific light sources. Light sources may be of ovaloid or paralepidid shape. Their size, colour, position, orientation and radiance are user-defined.
2. Environments which simulate ambient illumination. Environments are provided as 360-degree spherical images that define the incoming light from the environment. The utility toolbox is compatible with conventional as well as High Dynamic Range (HDR) images. The toolbox allows for uniform scaling of the environment radiance as well as arbitrary rotations of the spherical image. Spherical images can be acquired by a 360 camera or downloaded from one of the many available vendors on the internet[2].

Figure 10. Lighting using prescribed light sources (left) and environment illumination (right).

An arbitrary number of light sources can be to the simulated scene. Moreover, the two types of lighting can be combined by adding multiple light sources to the simulated environment.

3.1.4. Viewpoint

Once the scene is set, the viewpoint for the image to be rendered needs to be defined. Conventionally, this is defined by specifying the intrinsic and extrinsic camera parameters. The extrinsic parameters regard the camera pose (location and orientation). Intrinsic parameters determine the field of view and the resolution of the output image(s).

An additional parameter is provided that determines the number of samples that will be used to simulate the rendered scene. When rendering an image, the toolbox has to solve a high-dimensional integration problem that involves the geometry, materials, lights, and sensors that make up the scene. Because of the mathematical complexity of these integrals, they are solved numerically by evaluating the function to be integrated at a large number of different positions referred to as samples. To do its work, a rendering algorithm, or integrator, will send many queries to the sample generator. In other words, the higher the value of this parameter is, the better the quality of the image.

[1] Without adding light to the simulated scene, the result would be a black image.
[2] A valuable resource of such images is: https://polyhaven.com/hdris

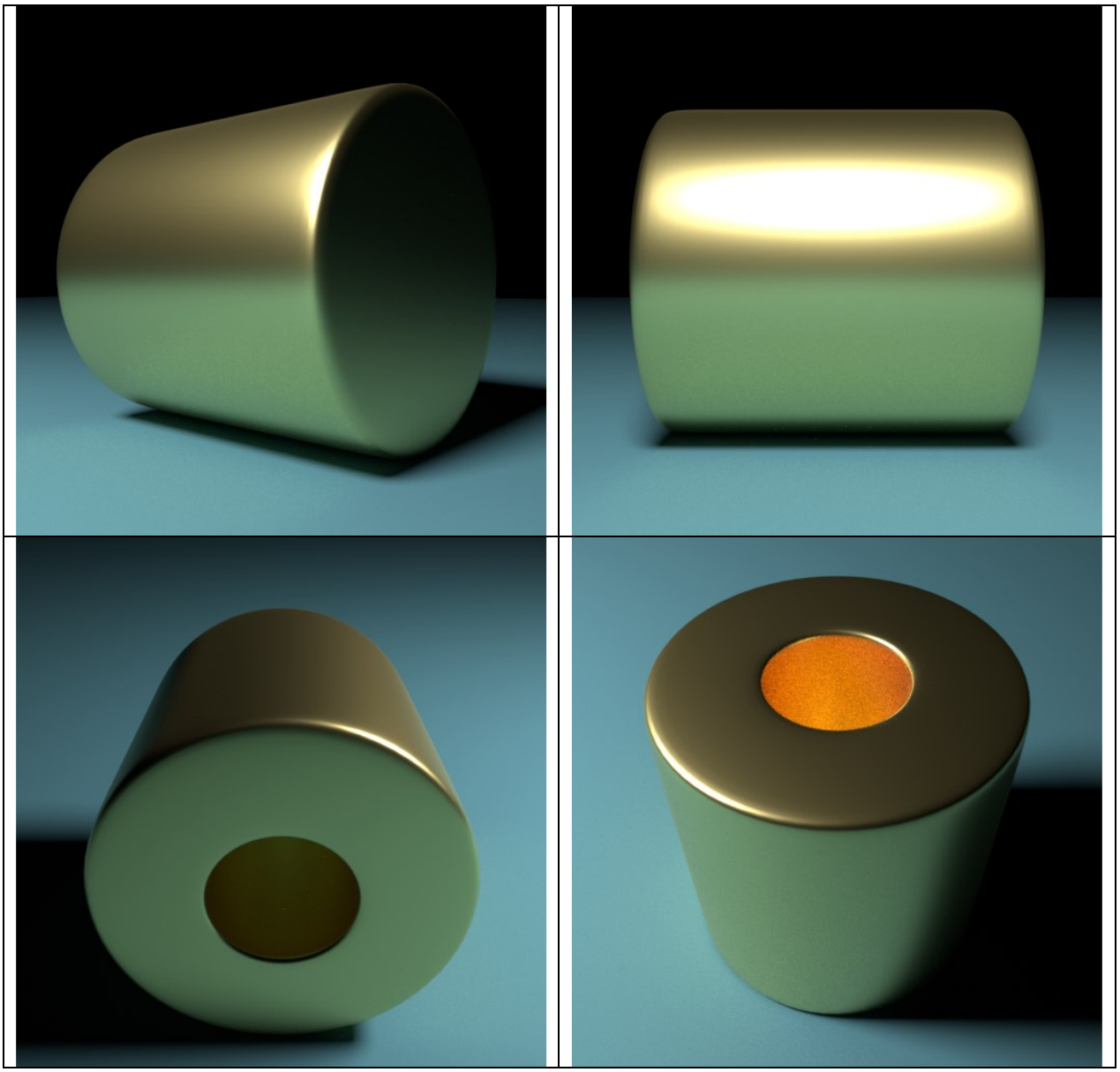


Figure 11. A metallic cylinder is rendered from different viewpoints.

3.1.5. Outputs

Three output types are supported:

1. Single image: the scene is rendered from the specific viewpoint that has been defined.
2. Rotational video: the user defines an axis, step, and angle of rotation and an AVI video is produced that emulates this camera motion. This type of output is useful for creating videos that show artefacts from multiple views.
3. Animation video: in this case, a sequence of scenes is provided to the toolbox, as separate 3D files and a video that renders the corresponding animation is produced. This type of output is

useful for creating videos that show simulation results, rendered with the associated materials rendered realistically.

3.1.6. Execution

The toolbox supports three types of execution, depending on the available hardware:

1. Serial CPU execution, where the simulation is conventionally executed.
2. Parallel CPU execution, where – if available – the parallel capabilities of the multiple CPUs are exploited.
3. Parallel GPU execution, which – if available – exploits the graphics card of the computer.

The aforementioned options are listed in order of execution speed, with the first being the slowest and the last being the fastest. The exact execution times depend on the capabilities of the specific hardware available.

3.1.7. Integration of physics-based simulation and rendering

We integrated our visualisation toolbox renderer with the FEM simulation engine we use (Simulia Abaqus) to create highly realistic renderings of mechanical processes, accurately reflecting the material behaviours and techniques inherent to these crafts. By merging the precision of FEM simulations with the visual fidelity physics-based rendering, we produce detailed visual representations that offer a deeper understanding of the subtleties involved in traditional methods.

This level of realism is invaluable for a variety of applications. In research, it enables a more thorough analysis of how different materials and techniques interact, providing insights into why certain methods were developed and how they can be adapted or improved. For educators and students, these realistic simulations serve as powerful teaching tools, offering an accessible way to explore complex craft techniques.

Moreover, the integration has significant implications for cultural heritage and conservation. By providing accurate visualizations of traditional crafts, we can better appreciate their cultural significance and support efforts to conserve and restore artefacts. This technology also opens new possibilities for innovation within traditional crafts, enabling the exploration of new materials and techniques in a virtual environment before they are applied in practice. In commercial and artistic contexts, this integration offers exciting opportunities for product design, prototyping, and artistic expression. By combining traditional craft techniques with modern technology, designers and artists can create unique works that push the boundaries of what is possible.

The integration works using file communication between the two software suites. Specifically, we extended the visual toolbox to read Simulia Abaqus output files, in OBJ format. From the simulation input we read the materials used and, in the visualisation toolbox, we specify their visualisation properties appropriately. Below we present material-specific visualisations of archetypal simulators presented in D2.1 “Action and affordance modelling”.

The first example, in Figure 12, regards the carving simulation. In this case, it is instantiated for a metal carving, specifically copper, action, using a Tungsten-made wedge. The full video can be found at https://youtube.com/watch/eKw25yT-UkA?feature=share and in https://youtu.be/EpuJlB1bEdc another video shows the same scene from multiple views.

Figure 12. Metal carving simulation and physics-based rendering.

In Figure 13, we present a drilling example using a workpiece and tools from the same materials. The figure shows two instances of the process, in the lefty and middle images. In the right image, we show another capability of our approach, which is "hyper-realistic" rendering, in which we alter the realistic parameters to provide more explanatory visualisation. In this case, the workpiece and supporting plane are (hyper-realistically) rendered as semi-transparent glass to enable visualisation from a "down-under" viewpoint and allow for a better view of the interaction of the tool with the material. The original video can be found at https://youtube.com/watch/q8i8R3cCFMs?feature=share and the hyper-realistic rendering is https://youtu.be/OTW0CeY2O9w.

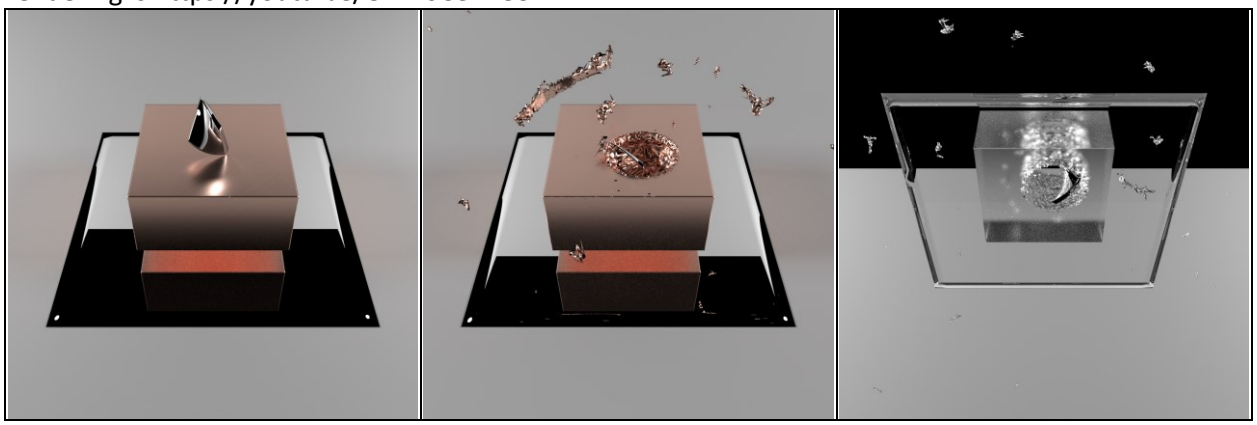


Figure 13. Metal drilling simulation. The left and middle images show two instances of the action. The image on the right, shows an instance of the same action from another viewpoint, down and under the supporting plane; this plane and the workpiece are rendered as semi-transparent materials, to enable visibility of the drilled structure.

Using the toolbox, we can focus rendering on regions of interest to illustrate the details of the technique and the interaction of the tool with the workpiece in detail. The important factor here is that re-rendering at a higher resolution is not a magnification, but a re-rendering at a higher level of detail. In Figure 14, we present such a detail from the previous example, at the regions where the drill interacts with the workpiece.

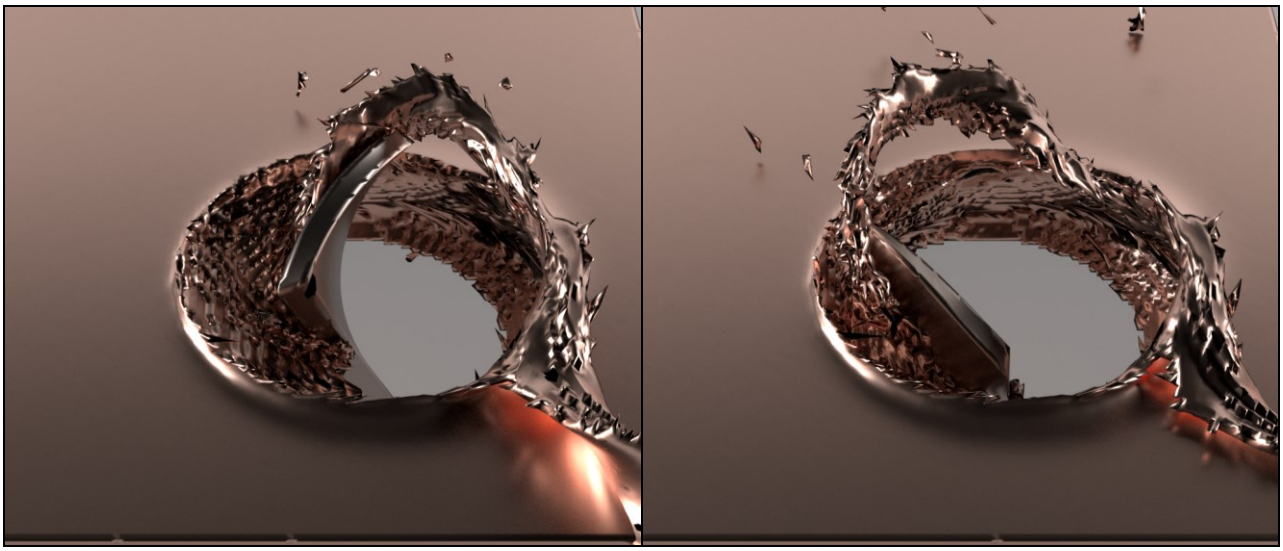


Figure 14. Re-rendering of a detail of Figure 13.

Finally, in Figure 15, we render the hot-rolling process that is executed in two steps, for the consecutive thinning of a metallic workpiece. The video can be found here: https://youtube.com/watch/aus1FQQrNko?feature=share

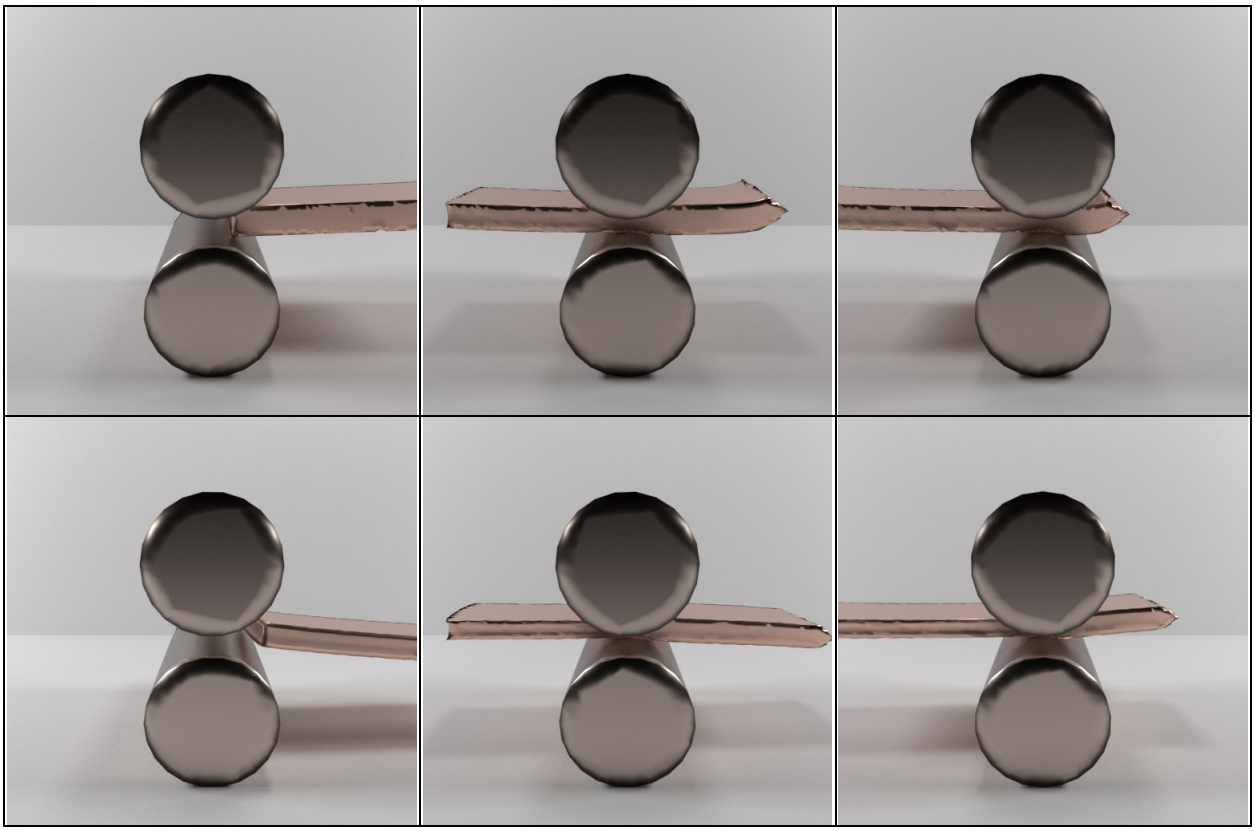


Figure 15. Two-step simulation of hot-rolling of metal and its physics-based rendering. Top: first step. Bottom: the second step.

4. Action simulation toolbox

In this section, we describe our efforts to bridge the computational gap between scientific, multi-physics simulation and the real-time requirements of training applications. The problem we are trying to solve is that accurate physics-based simulations require significant computational time, making them inappropriate for real-time implementation. The way that we approach this problem is to use a simpler physics-based simulation engine which, in addition, includes GPU-based parallel processing to accelerate the computation and meet real-time requirements.

4.1. Action simulation toolbox

The NVIDIA PhysX engine is a real-time physics simulation engine developed by NVIDIA. It is designed to handle complex physics calculations in video games and other real-time applications, enabling realistic simulation of physical phenomena like rigid body dynamics, fluid dynamics, soft body dynamics, and particle systems. PhysX provides detailed and accurate simulation of physical interactions, such as collisions, gravity, and the behaviour of materials under various forces. It is supported across multiple platforms, including Windows, Linux, and macOS. PhysX leverages the power of GPUs to perform physics calculations, which offloads this processing from the CPU, resulting in more realistic simulations and better overall game performance. It is integrated into popular game engines such as Unreal Engine and Unity, making it accessible to game developers for creating rich, interactive environments. Beyond gaming, it can be used in XR applications, where realistic physics simulation is critical for immersion and accuracy.

4.1.1. Rationale

In this subsection, we describe the way we create the software interface between the two. To achieve this we developed a software toolbox called “SimplePhysX5”. The SimplePhysX5 developed is a high-level wrapper for the NVidia PhysX engine v5.4.0 that provides a software API for creating interactive simulations in the Unity game engine. The toolbox is targeted at producing simulations of interactions between “tools” (rigid objects) and materials (soft bodies) using the Unity game engine.
Following our project rationale, which is to analyse complex actions into elementary ones, the toolbox provides only three elementary tools. These are the additive, subtractive, and shaping actions. Interlocking actions need not be simulated with a different tool, but covered by the shaping actions. We do not need an additional tool for interlocking as that interlock is a result of shaping actions (see also D2.1 “Action and affordance modelling”).

4.1.2. Technical specification

The SimplePhysX5 API is exposed as a small set of C# classes, usable from Unity. These classes internally call C/C++ functions from the provided DLLs. A Unity project is also provided, that demonstrates usage of the API. The toolbox requirements are:

- NVidia GPU with the latest drivers installed
- Microsoft Visual C++ 2019 x64 redistributable[3]

[3] Available at: https://aka.ms/vs/17/release/vc_redist.x64.exe

- Unity game engine (version 2022.3.22f1, or later)

There are four different kinds of objects that the API exposes and the developer can create:

- **Ground plane object**: this is a planar object that remains static (does not move) throughout the simulation. As the name suggests, it simulates the ground. It behaves as having infinite mass. Therefore, objects that collide with it eventually stop moving.
- **Rigid dynamic objects**: these are objects that can freely move and rotate. Rigid means that their shape remains constant. The PhysX engine calculates and updates their position and orientation as they collide with other objects or the ground plane. At each frame, the developer is responsible for querying their current position and orientation (using the SimplePhysX5 API) and updating the transformation of the corresponding entities in Unity (called "GameObjects" in the Unity jargon).
- **Rigid dynamic kinematic objects**: these are objects that can move and rotate. Their shape cannot be deformed. Unlike the simple rigid dynamic objects described above, PhysX does not control the position and orientation of kinematic objects. Instead, the developer is responsible for setting these properties at each frame (using the SimplePhysX5 API). They still take part in the simulation and they can collide and interact with other objects. Rigid dynamic kinematic objects are used to simulate objects that are controlled by the user, using either some controller, Unity's animation timeline, or scripting.
- **Soft bodies**: these are objects that get deformed when other objects collide with them. When collisions do not exist anymore, their shape can be (fully or partially) reverted to the initial state. Alternatively, the deformation can be permanent. The exact behaviour and the amount of deformation are controlled by their material settings. At each frame, the developer is responsible for querying the new positions of the mesh's vertices (using the SimplePhysX5 API) and updating the corresponding GameObject's mesh in Unity.

In general, the developer creates a scene in Unity and populates it with 3D objects. It is assumed that some of the objects' transformations could be animated by either the user, using some controller, or Unity, for example using scripts or a timeline object. Unity acts both as the 3D renderer and the animation controller, while SimplePhysX5 is responsible for calculating collisions and deformations.

3.2.1. Simulated objects

The objects that take part in collisions, should be registered with the SimplePhysX5 API during start-up, providing their triangle mesh and transformation. Then, at each frame, for objects that are moved and rotated by the user (kinematic objects), the developer should inform the SimplePhysX5 API about their current position and orientation. In addition, for objects that are not controlled by the user, the developer should query (using the SimplePhysX5 API) for their current position and orientation (which could have changed due to gravity or collisions) and update the corresponding GameObject's transformation accordingly. Finally, for soft bodies, the developer should ask the SimplePhysX5 API for the object's deformed vertices and update the corresponding GameObject's mesh vertices accordingly.

3.2.2. SimplePhysX5 API classes

The core of the API is the PxPhysics class, which enables the definitions of materials and objects. It also provides the *simulation* method, which runs the physics simulation. This method should be called at each frame from a Unity script. The class also provides the *PxErrorCallback* delegate that can be set to print messages useful for debugging. It is advised to set an error call-back during development and

remove it in the release version. The provided C# classes and their methods are named as their corresponding C++ classes of the NVidia PhysX engine.

The PxRigidDynamic and PxSoftBody classes are used to set or retrieve the transformation of rigid bodies and retrieve the deformed vertices of soft bodies, respectively. The PxMaterial and PxFEMSoftBodyMaterial classes are used to control friction parameters for rigid dynamic or soft bodies, respectively. Also, the PxFEMSoftBodyMaterial and PxFEMParameters classes control the deformation properties of soft bodies.

Most classes implement the IDisposable interface. Therefore, the Dispose method should be called at program termination to avoid memory leaks. The objects need to be disposed of in the reverse order in which they were created. If a *PxErrorCallback* delegate is used in the PxPhysics class, the delegate should not throw exceptions and should only be destroyed after destroying the PxPhysics instance.

The SimplePhysX5 API is contained in the SimplePhysX5.dll, which should be loaded as a plugin in Unity. It is statically linked to the nVidia PhysX DLLs (PhysX_64.dll, PhysXCommon_64.dll, PhysXCooking_64.dll, PhysXFoundation_64.dll), which should be located in the same folder as the SimplePhysX5.dll. The PhysXDevice64.dll and PhysXGpu_64.dll implement the GPU-accelerated part of the nVidia PhysX API. They are loaded at runtime. They should be placed in a folder and the path to that folder should be passed to the PxPhysics class constructor.

3.2.3. Mesh cooking

When creating physics objects, the triangle meshes of the objects must be cooked (preprocessed) by the PhysX engine to generate the structures that accelerate collision detection. Cooking can take some time (up to a few minutes) depending on the mesh's size and required accuracy. To accelerate program start-up, the PxPhysics class exposes two methods of creating objects:

1. (slow) simply cooks the mesh in memory every time the application runs.
2. (fast) cooks the mesh and saves the cooked mesh in a file. This file can then be loaded in subsequent application runs to create the physics object without performing the cooking again.

3.2.4. Limitations

The current version of the toolbox has the following limitations

- Even though multiple soft bodies can be created, only one can be movable/rotatable. In other words, a single tool can be used each time, implying that a single practitioner uses deformation tools each time.
- Applying spatial scaling in Unity does not work correctly with collisions. 3D models should be scaled to the correct size before getting imported into Unity.

4.2. Solids by revolution

An algorithmic approach is a proposed method for generating 3D models that represent the geometry of solids by revolution. Such solids are found in several crafts such as woodturning, glassblowing, and pottery.

Given that the solids are generated by revolution they have a vertical symmetry. This property is exploited for computational efficiency as follows. The editing takes place on a hypothetical planar segment, which is represented by a binary image called a “template”. The editing of the shape takes place in this image which is rotated to generate the simulated solid by revolution.

Editing the shape on the template is performed concerning the physical constraints of the simulated mechanical configuration. Specifically, the infrastructure enables interaction only with the outer surface of the revolved solid. In the template image, this translates into editing a contour which, when the template is revolved, generates the solid.

The method involves manipulating a template image and applying various transformations to produce and modify a solid by revolution. The generated 3D model is symmetrical about the Y-axis, with additional features for user-defined modifications and transformations. The proposed approach provides tools for additive, subtractive, and mass-preserving modifications.

4.2.1. Template image

The central data structure is a 2D binary matrix (or image), referred to as the "template." In this image, white pixels represent material, while black pixels indicate empty space. By conceptually rotating this template 360 degrees around its first column (aligned with the Y-axis in 3D space), we create a 3D model. The height of the 3D model is fixed at one world unit, while the dimensions along the X and Z axes are proportional to the template's aspect ratio. The model is centred at the origin (0,0,0) of the coordinate system.

4.2.2. User-defined tools

Users can manipulate the template image using three types of tools:

1. **Subtractive Tool**: Removes material by converting white pixels to black at points of contact.
2. **Additive Tool**: Adds material by converting black pixels to white at points of contact.
3. **Mass-Preserving Tool**: Moves material by converting white pixels to black at contact points and then adding an equal number of white pixels elsewhere.

Internally, each tool is represented as a sphere in the 3D world and visualised as a circle on the template image. The reason for selecting the particular shape is the efficiency of the sphere in the computation of collision detection, meaning the contact of the tool with the material. The generation of tools of more complex shapes is possible, by combining multiple such spheres. Users can apply translations, rotations, and uniform scaling to these tools, affecting the size and position of the circles on the template.

4.2.3 Algorithmic workflow

4.2.3.1. Initialization

First, the template image is allocated according to the user preferences that determine its size and resolution. Next, the simulated tools are initialised according to user-specified parameters that determine the type (subtractive, additive, mass preserving) and active status. Also, each tool has an integer that serves as its unique identifier, used to change its parameters later.

The 4x4 transformation matrix (Q_t) for each tool brings its 3D shape from its “local coordinate frame” to the “world coordinate frame”. In the local coordinate frame, the centre of the sphere is the point (0,0,0)

and the sphere has a radius of 0.5 units. The world coordinate frame is chosen by the user according to the requirements of each application. The transformation matrix can contain any combination of translations, rotations and uniform scaling. The world coordinate frame is the same for all tools.

A Boolean flag indicating whether the tool is currently “active”. If the tool is not active, it is not taken into account by the API and does not change the template.

The API stores these parameters for each tool. The user can change these parameters (except the unique ID of the tool) at any time using the corresponding API functions.

At initialisation, the template is uniformly downscaled (keeping the aspect ratio) to a size specified by the user, through the "resolution" parameter of the API. Downscaling is done to reduce the execution time and (at the same time) the complexity of the generated 3D model. Downscaling is optional.

4.2.3.2. Runtime

For generality, during the execution of the interactive simulation, we assume that the user may have changed the initial (or preceding) parameters of the tools and the transformations of the tools and the solid. This may have been done using controllers, through Unity's animation timeline or scripting. In each case, we assume that the user has called the corresponding API functions so that the API has been informed of the changes.

During runtime, at each frame, the active tool changes the pixels of the template according to its type. More details on how this is implemented are provided below. At each frame, contours are traced in the binary template using the method in [14] and its implementation in OpenCV's “findContours” function. Importantly, the algorithm returns the contours it found in the image as a sequence of 2D points and the points always have a consistent ordering. This allows us to produce a 3D model with consistent triangle winding and consistent normal orientation.

Contour tracing typically results in >1 contours. We select only one, which we will use to generate the solid and we discard the rest. We select the contour that contains the bottom left pixel of the image template. The lower left pixel of the image template corresponds to the centre of the base of the solid[4]. We note here that our API guarantees that the specific pixel will always be white in the template and will therefore be contained in one of the contours.

To achieve real-time performance, the resolution of the image template is small. To achieve solids of high resolution, we smooth and interpolate the traced contour. Smoothing is implemented by convolution with a 1D Gaussian kernel. This ends up giving a smoother 3D model. Smoothing is optional and enabled by default. It is disabled using the appropriate API function. Functions are also provided with which the user can also set the kernel size and variance of the kernel.

Thereafter, we calculate the 2D normal vectors of the smoothed contour. The calculation is based on computing the gradient direction (first derivative) of the contour at each of its points. Therefore, we have one normal vector for each point of the contour. These will be utilised later to produce the 3D normal vectors of the 3D model.

At the end of each frame, we create the 3D triangle mesh from the smoothed contour, as follows:

1. We create an array of 3D points, one 3D point for each 2D contour point. 3D points are calculated as (x*s, y*s,0), where x, y are the pixel coordinates of the 2D contour point. The scale

[4] We chose this particular pixel for aesthetic reasons: if we had kept another contour, we might have ended up with a solid that would "float in the air".

s is calculated as 1/H, where H is the height of the template image so that the height of the 3D model is 1.

2. We create an array of 3D normals, one 3D normal for each 2D contour point. The 3D normals are calculated as (n_x, n_y, 0), where n_x, n_y are the coordinates of the corresponding 2D normal of the contour that we have already calculated above.
3. We append the above two arrays to 2 empty arrays respectively, *points* and *normals*, which will be the points and normals arrays of the final mesh.
4. We rotate the 2 above arrays by *i* degrees around the Y axis and append the result to *points* and *normals* arrays respectively. Repeat several times using angles 2**i*, 3**i*, and so on, until 360 degrees of rotation are completed. The number to repeat is determined by the parameter *slices* of the API and determines the number of slices the generated solid will have. By default, we use 72 slices, so the corner *i* is 360/72=5 degrees.
5. We create the array *indices*, which we fill with the triangle indices. These are easily calculated (...). We take care to maintain consistent triangle winding when calculating the triangle indices.
6. The triangle mesh of the solid consists of *points*, *normals* and *indices* arrays. The API provides a function to return these arrays to Unity. Note that we do not generate texture coordinates because we encountered a problem generating texture coordinates in the regions near the poles of the solid. Instead, texturing is done through Unity using the tri-planar texturing technique, which is often used in 3D graphics for texturing unknown or procedurally generated meshes such as terrain, etc.

The user replaces in Unity the existing triangle mesh of the solid with the new one, simultaneously applying the transformation Q_s.

The API, after calculating the new mesh, also generates a 2D image showing some of the intermediate results (e.g. detected and smoothed contours, 2D normals) for debugging use. The image can be displayed to the user in a window using the appropriate function.

We note that to save computing power, the API also returns a Boolean flag that indicates whether the mesh has changed or not since the last time it was generated. The mesh will not have changed unless the algorithm parameters have changed (e.g. *resolution*, *slices*, etc.), if the transformations of the tools and solid have not changed, if all tools were inactive, or if there were no collisions between tools and solid. In these cases, the mesh is not calculated again and, if requested, the mesh that was calculated last time is returned.

4.2.3.3. Manipulation of the image template by the tools

As mentioned, we consider that the sphere used by each tool is initially located in the local coordinate frame (centre (0,0,0), radius 0.5) and the user applies a transformation to it. We chose this convention because this is the mesh that Unity uses when creating a sphere object. Therefore, Q_t coincides with the transformation that the user will apply through the Unity editor and is provided by the corresponding function of the Unity engine.

The following only applies if the tool is active. If it is not active, the corresponding code is not executed.

First, for each tool, the centre and radius of the sphere used by the user should be calculated, *in the local coordinate frame of the solid*, i.e. after the Q transformation and Q_s. We calculate the 4x4 transformation matrix M, which transfers from the local coordinate frame of the tool to the local coordinate frame of the solid: $M = Q_s^{-1} * Q_t$. Thus, the centre of the sphere in the local coordinate frame of the solid is C = M * (0, 0, 0). We also consider a point on the sphere (0.5, 0, 0), which in the local

coordinate frame of the solid is P = M * (0.5, 0, 0). The radius of the sphere in the local coordinate frame of the solid is R = d(C, P), where d is the distance between two 3D points.

Then, the circle (centre and radius) in which this sphere is projected in the template image should be calculated. The y coordinate of the centre of the circle is C.y*H, where H is the height of the template in pixels. Since the solid is symmetric about the Y axis, it follows that the x coordinate of the centre of the circle is hypot(C.x, C.z) * H, where hypot(x,y) is the hypotenuse of the triangle, $\text{sqrt}(x^2 + y^2)$. Thus, the centre of the sphere is projected on the template image at pixel c = (C.y*H, hypot(C.x, C.z)*H). The radius of the circle in pixels is ρ = R * H.

By having the centre and radius of the circle, we know which pixels of the image template the tool will modify. The behaviour from here on depends on the tool as described below.

4.2.4. Tools

4.2.4.1. Subtractive

All the pixels of the template inside the circle are painted black, except for the following special cases:

1. If the template does not contain white pixels inside the circle (= there is no collision between the object and the solid) we do nothing and it is noted that the tool did not make any changes (this information is then used to skip the mesh generation if necessary).
2. If all the pixels of the template inside the circle are white (= the tool is "inside" the solid) we do nothing and it is noted that the tool did not make any changes (this information is then used to skip the mesh generation if necessary). This is because if we paint the specific pixels black, the solid gets "holes" internally and then some phenomena are observed where the manipulation of the solid by the tools becomes unexpected from the point of view of user interaction.

After the changes, the tool "paints" the lower left pixel of the template white and runs connected components labelling on the template image. If the template has ended up having more than one connected component (= the solid was "split" into two or more pieces) we keep the component that contains the bottom left pixel of the template (the "base" of the solid) and paint all the rest black.

4.2.4.2. Additive

All the pixels of the template inside the circle are painted white.

Special cases:

1. If all the pixels of the template inside the circle are black (= there is no collision between the object and the solid) we do nothing and it is noted that the tool did not make any changes (this information is then used to skip the mesh generation if necessary).
2. If the template does not contain black pixels inside the circle (= the tool is "inside" the solid) we do nothing and it is noted that the tool did not make any changes (this information is then used to skip the mesh generation if necessary). This is done to save computing power.

After the changes, the tool "paints" the lower left pixel of the template white and runs connected components labelling on the template image. If the template has ended up having internal “holes” (= the tool connected edges of the solid), the tool fills the holes with white pixels.

4.2.4.3. Mass preserving

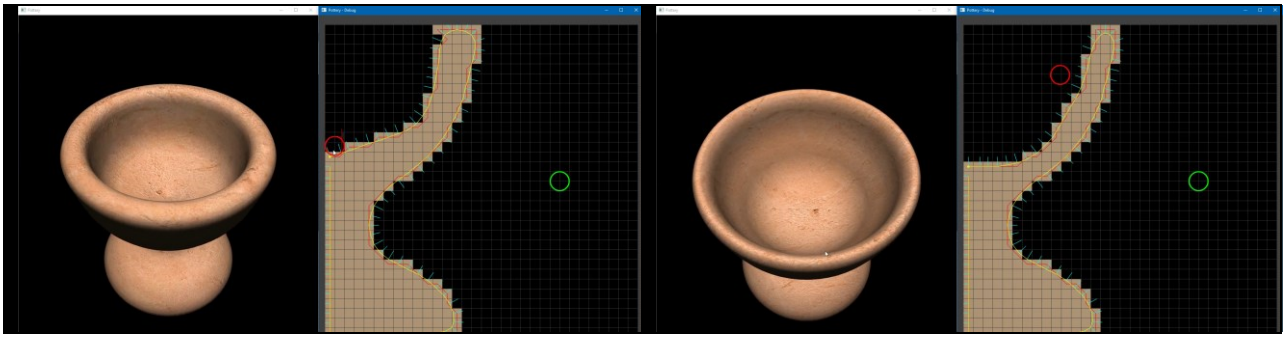
Initially, it behaves like the subtractive tool + special cases (see 1). After the changes, the tool "paints" the lower left pixel of the template white and runs connected components labelling on the template image. If the template has ended up having more than one connected component (= the solid was "split" into two or more pieces) we keep the component that contains the bottom left pixel of the template (the "base" of the solid) and paint all the rest black (same behaviour as the subtractive tool). But if we have only one component, the tool adds white pixels as follows:

1. First, we find all the edge pixels of the template (the "surface" of the solid).
2. The edge pixels are sorted based on the (geodesic) distance from the pixels where there was a collision of the tool with the solid.
3. We paint white the pixels that are adjacent to the edge pixels, starting from those with the smallest distance and continuing to those with a greater distance. Thus, "mass" is moved to other parts of the surface of the solid, while at the same time the solid "thickens" faster near the points where there was a collision than at other points.

We stop painting white pixels when there are no more edge pixels or if the surface of the template image (number of white pixels) has become equal to *targetArea*. The *targetArea* initially is the same as the number of white pixels that the template had when we loaded it from the disk when starting the program. The *targetArea* however, can be changed in some cases by the tools (e.g. in the case that the solid "breaks" into two pieces and we throw away one). In case we run out of edge pixels before we have reached the *targetArea*, moving pixels continues the next time the mass preserving tool is used (at the next frame of the animation). Therefore, after repeated use of the tool, the surface of the solid will at some point reach *targetArea*.

4.2.5. Example

To reduce computation and make the simulation interactive, the geometry of solids by revolution is exploited. In the examples shown in Figure 16, the left panels show the real-time updated 3D rendering and the right panels show the inner representation maintained by the toolbox. The user can edit the voxel grid and inspect the result from any viewpoint, in real-time. The video demonstration can be found at https://youtu.be/Yc7FtCdOeSs



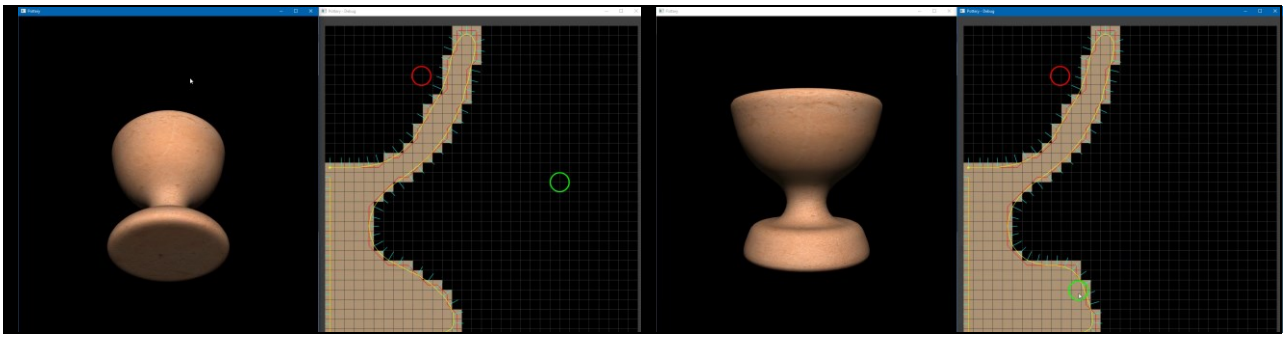


Figure 16. Virtual pottery and software representation are maintained for real-time rendering in the Design Studio.

In Figure 17, we show the user's view of the interaction. Specifically, the spherical tool is now substituted by a 3D model of a human hand, manipulated by a 3D controller. Moreover, the shaping tool is demonstrated, in the context of pottery. The full video of the demo9nstration can be found at https://youtu.be/W2tzByCnlpc.

Figure 17. User's view of interaction with a clay body and the usage of a hand as a shaping tool.

5. Interactive simulations

In the pursuit of interactive and craft-specific simulations, a significant challenge arises in the form of computational time. Finite Element Method (FEM) simulations, which are crucial for accurately simulating the interaction between a tool and a workpiece, are not feasible in real time due to their computational intensity. These simulations can require several hours to produce detailed and precise results, making them unsuitable for applications requiring interactivity.

To address this challenge, a preliminary but computationally demanding solution was proposed in Deliverable D2.1, "Action and Affordance Modelling". This approach involved precomputing all possible outcomes based on user control parameters, such as tool speed and angle of incidence, and subsequently rendering the pre-computed results according to the specific parameters employed by the user. While this method proved functional, it is not scalable, as it demands substantial storage capacity and high-speed hard drives to operate in real time.

Given these limitations, we have adopted a more efficient approach leveraging machine learning, specifically artificial intelligence (AI). Instead of precomputing and storing action results for every possible parameter configuration, we generate training data by conducting a select number of simulations. These simulations are strategically chosen to sparsely sample the parameter space. Using this data, we train a neural network to approximate the simulation results based on the given parameters and material properties. This trained network is then integrated with the PhysX physics-based engine, allowing for real-time approximation of the original simulation.

The following subsections detail the methodology used to produce the training data and the subsequent application of this data to achieve real-time simulation capabilities.

5.1. Training data

Training data are produced using the Simulia Abaqus simulator. The simulations are constrained to the material properties relevant to the studied action.

We present two cases of training data production. The first regards the most studied approach and regards structural changes. We have recently started studying the role of temperature in these simulations so that we can modulate this parameter as well in our training simulation. Thus, the second example studies the production of temperature-dependent data.

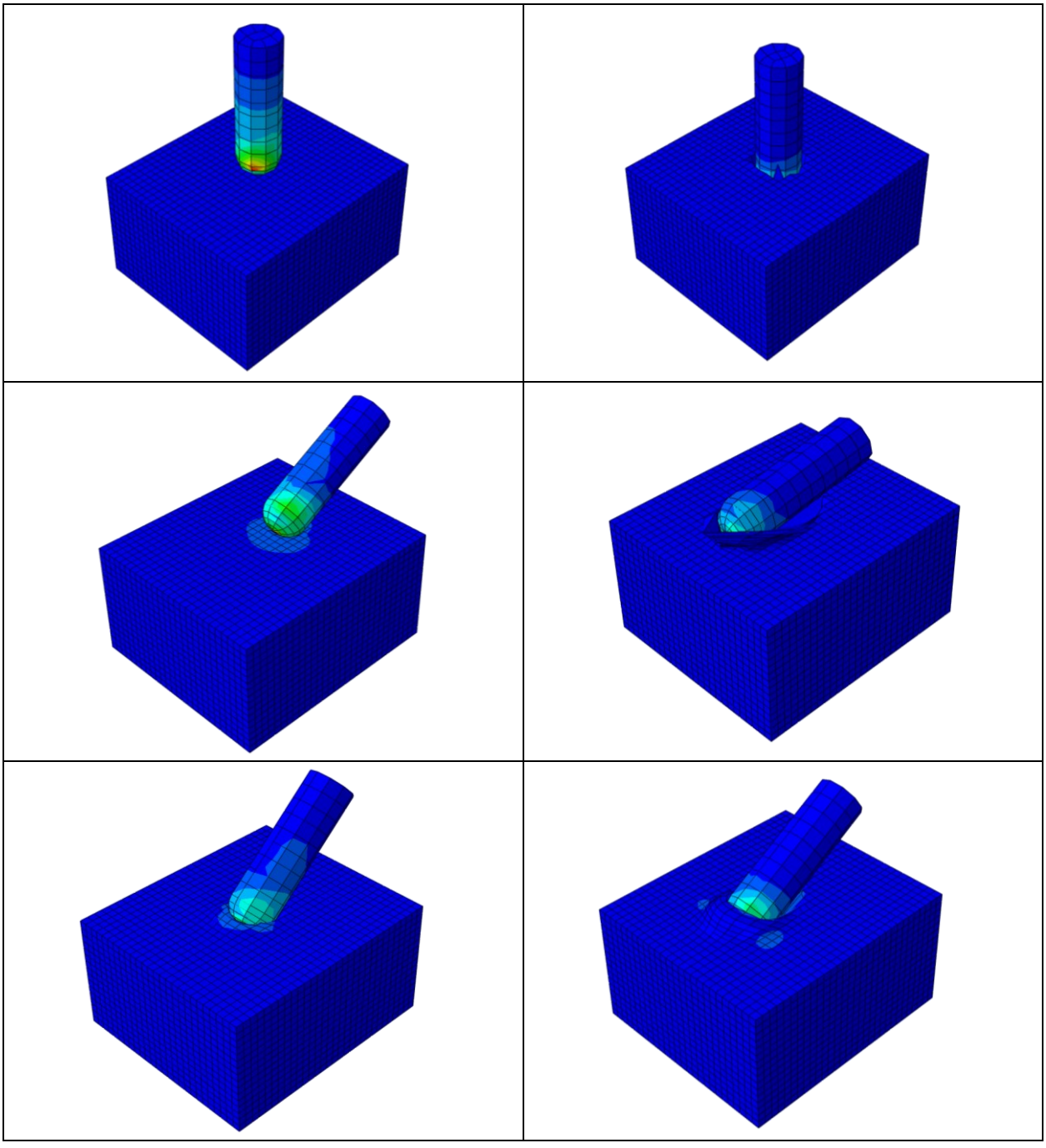
5.1.1. Object interaction data

The data generated from the simulator are provided as training data to the AI through a structured and methodical process. First, the relevant parameter space is defined, including variables such as tool speed, angle of incidence, and material properties. A representative subset of these parameters is carefully selected to ensure that the simulations cover a broad range of possible scenarios. This selection is essential for enabling the neural network to generalise effectively across different conditions.

Next, these selected parameters are used to run a series of simulations, using the FEM simulator. Each simulation produces detailed output data, capturing the specific results of the interaction between the

tool and the workpiece under the given conditions. These results might include data points related to stress distributions, deformations, and material removal. The data are, then, labelled and structured into a format suitable for training the neural network. Each dataset includes the input parameters (e.g., tool speed, angle, material type) and the corresponding simulation results (e.g., deformation patterns, stress levels). This structured data set serves as the training input for the AI model.

Following the rationale of D2.1, we created an example, in which a tool interacts with (deforms) a workpiece at multiple angles of incidence. In this case, however, we simulated the data sparsely, every 15 degrees and not every one degree as we did in D2.1.



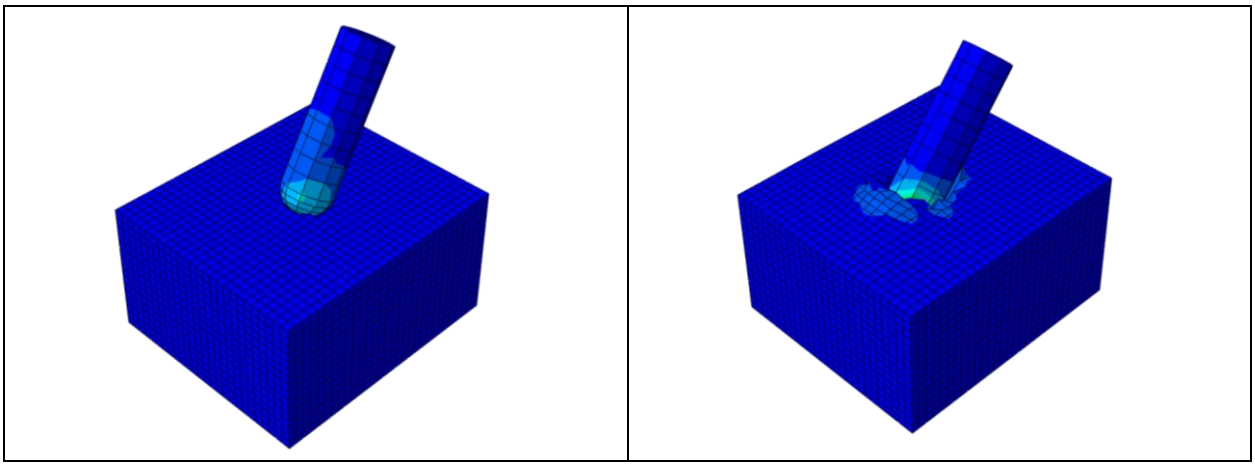


Figure 18. Creation of training data for a tool and a workpiece for incidence angles 0, 30, 45, and 60 degrees (from top to bottom). The left column shows the instance before and the right column the instance after the tool impact.

5.1.2. Thermal-dependent data

A first investigation was performed to qualitatively extend the generated data, by making them both material-specific and temperature-dependent. We present a preliminary study on how we intend to treat this task, in our next steps.

In this case, we are studying the plasticity of metals concerning temperature. The objective of the simulation study is to analyse the mechanical behaviour of aluminium under varying temperatures and mechanical loading conditions. Specifically, the study investigates how temperature affects the stress and pressure distribution within an aluminium rectangular parallelepiped (workpiece) subjected to mechanical pressure. In the simulation, an imprinting (debossing) tool is moved along the Y-axis to apply pressure on the workpiece, which is positioned on top of a rigid ground plane. The tool has the shape of a parallelepiped.

The focus is on analysing the plastic properties of the workpiece at a displacement of 0.001 m and 0.0015 m and three temperatures: 20°C, 300°C, and 660°C. The simulation model includes gravity as a fundamental external force acting on the system. The simulation is conducted using Abaqus to observe how temperature influences the stress and pressure distribution within the workpiece.

Following the approach presented in “D2.1. Action and affordance modelling” we specialise the archetypal action for deformation and instantiate the following scene elements:

- **Tool**: The tool has dimensions of 0.04 m × 0.05 m ×0.04 m.
- **Workpiece**: The workpiece has dimensions of 0.09 m × 0.09 m × 0.01 m
- **Ground Plane**: The ground plane is a rigid body that acts as a fixed support for the workpiece and does not deform. The ground plane has dimensions of 0.12 m × 0.12 m × 0.005 m

The tool is made of steel and the workpiece from aluminium. The material properties are defined as follows. For the tool, we use the environment temperature (i.e. 20 °C) and its properties are defined in that temperature. The workpiece is heated and as such more material properties are needed to describe its expansion due to heat and, most importantly, the change of plasticity and elasticity as a function of material temperature.

As such for steel we use the following properties.

- Density: 7850 kg/m^3

- Elastic Modulus: 200 GPa
- Poisson's Ratio: 0.3
- Yield Stress: 355 MPa with Plastic strain 0 and Yield Stress: 470 MPa with Plastic strain 0.178

For aluminium besides density (2700 kg/m^3) all other properties are temperature and provided in Table 1, Table 2, and Table 3.

Table 1. Young modulus and Poisson ratio of Aluminium as a function of temperature.

Young's Modulus (GPa)	Poisson's Ratio	Temperature (°C)
70	0.33	20
69	0.33	100
68	0.33	200
67	0.33	300
66	0.33	400
65	0.33	500
64	0.32	600
63	0.32	700

Table 2. Expansion coefficient of Aluminium as a function of temperature.

Expansion Coefficient ($10\times^{-5}$)	Temperature (°C)
2.3	20
2.5	100
2.6	200
2.7	300
2.9	400
3.2	500
3.5	600
3.7	660

Table 3. Yield stress and plastic strain of Aluminium as a function of temperature.

Yield Stress (MPa)	Plastic Strain	Temperature (°C)
275	0	20
255	0.005	100
230	0.01	200
200	0.015	300
175	0.02	400
150	0.025	500
120	0.03	600

In the simulation, the debossing moves in the Y-axis with displacements of 0.001 m and 0.0015 m. The simulations are conducted at three district temperatures for each displacement: 20°C, 300°C, and 660°C. Gravity is implemented in the simulation to represent the effects of gravitational force on the parts. The magnitude of gravity is set according to standard gravitational acceleration, typically 9.8 m/s^2. The element type used is C3D8R (an 8-node linear brick, reduced integration).

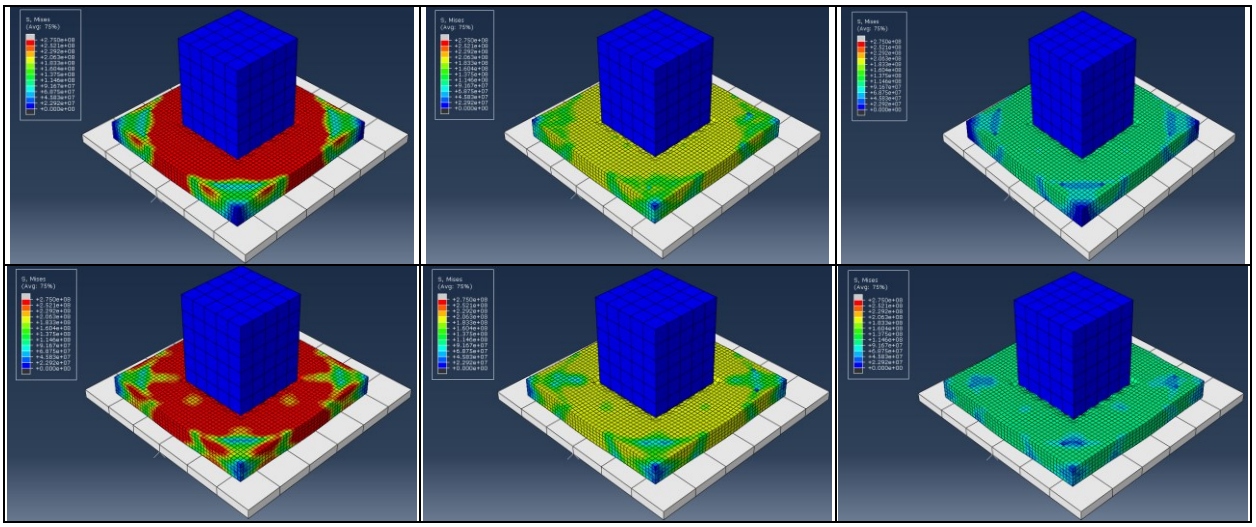


Figure 19. Von Mises Stress spatial distribution on the workpiece at three different temperatures 20°C (left), 300°C (middle), and 660°C (right), for two displacements of 1mm (top) and 1.5mm.

In Figure 19, the von Mises stress distribution on the workpiece is depicted for three different temperatures: 20°C, 300°C, and 660°C and the two displacement values. The colour bar located at the top left of the figure indicates the stress values, with higher stresses represented by colours near red and lower stresses represented by colours near blue or green. As the temperature increases, the colour of the workpiece shifts towards green, indicating that the stress within the aluminium decreases at higher temperatures.

Observations:

- **20°C**: The pressure is highly concentrated around the contact region with sharp pressure gradients.
- **300°C**: The pressure distribution at 300°C is more uniform compared to 20°C. At this temperature, the aluminium becomes more flexible, so the pressure spreads out more evenly. Although the pressure is still high near the contact area, it decreases in intensity, with more areas showing light blue colours.
- **660°C**: The pressure distribution is significantly more uniform and spread out across the surface. The workpiece shows an even greater reduction in pressure values, with a predominance of light blue colour. This indicates a decrease in pressure intensity, as the material deforms more easily under the applied load.

Across all three temperatures, pressure waves are created over the surface of the workpiece. These waves are indicative of the distribution and transmission of pressure from the point of contact where the debossing applies the load. The waves propagate outward from the contact area, showing how the pressure is dispersed across the plate. The simulation results for 0.0015 m demonstrate that the fundamental observations regarding stress and pressure distributions in the workpiece remain consistent across different displacements. With greater displacement (0.0015 m), the magnitude of pressure increases.

As the temperature increases, the overall colour of the workpiece shifts towards light blue. This change illustrates the decrease in pressure intensity, which is a direct result of the reduction in the yield stress and increased plasticity of aluminium at higher temperatures.

The stress distribution results demonstrate a clear influence of temperature on the von Mises stress within the workpiece. As the temperature increases from 20°C to 660°C, the stress values decrease significantly. This stress reduction is due to the decrease in the yield stress of aluminium and the increased plasticity at higher temperatures.

The observations of the pressure distribution results demonstrate the formation of pressure waves across the aluminium surface and the increase in pressure with greater displacement of the debossing. Additionally, the influence of temperature is evident, with higher temperatures leading to a more uniform and widespread pressure distribution, and an overall decrease in pressure intensity.

5.2. Material approximations

Real-time simulation of the manipulation of real-world volumetric deformable objects is a challenging task. Accuracy can be achieved with commercially available simulators [20] that rely on finite element methods. However, these simulators do not perform in real-time and are thus unsuitable for interactive applications. A recent review of simulators that are suitable for robotic applications and therefore typically perform in real-time can be found in [23].

To fulfil both simulation accuracy and real-time requirements we investigate three different approaches, two based on learning and one that relies on a GPU-powered real-time simulator, PhysX [22]. Learning-based approaches rely on training data to model several aspects of a physical system. In this work, we test two such approaches, one that models the object geometry only, 3DNS [27] and one that models both the geometry and the dynamics, ACID [26].

Our progress is the following. We constructed 3D manipulation data using Simulia which is considered as ground truth for all the methods. Initial experiments with this data for 3DNS and PhysX are shown in the experiments section. The experimental evaluation of ACID is pending.

5.2.1. Methods

5.2.1.1. 3DNS

Implicit surface representations are becoming increasingly popular and achieve state-of-the-art results in several tasks such as shape representation and reconstruction. 3DNS [27] proposes an approach for making local edits in objects that are represented using implicit surfaces. 3DNS uses a brush-based framework that is intuitive and can be used by sculptors and digital artists which is in line with CRAEFT goals. The experimental evaluation results show that 3DNS is accurate, in terms of modelling the desired edits, while preserving the overall object geometry outside the interaction areas, see Figure 45.

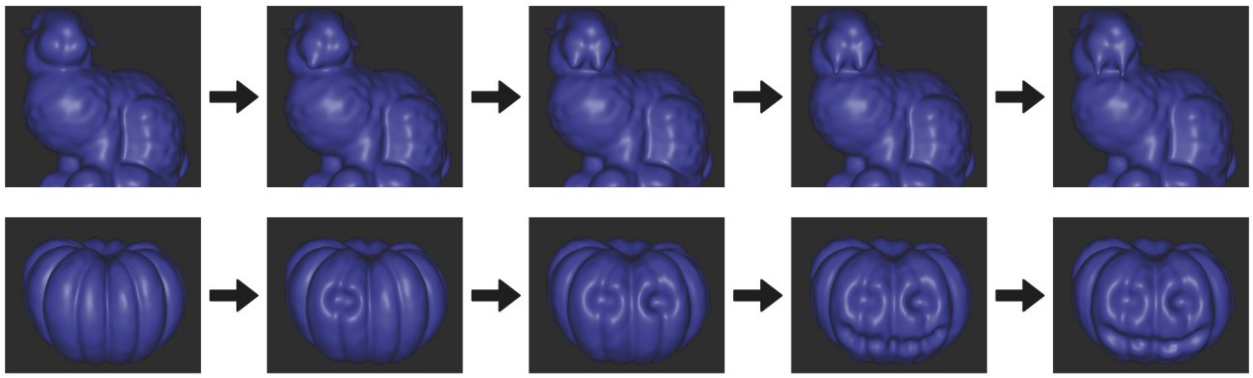


Figure 20. Example of multiple edits on two models using 3DNS.

5.2.1.2. ACID

ACID [26] is a dynamics model for deformable object manipulation based on implicit neural representations. Within ACID two techniques are leveraged: implicit representations for action-conditional dynamics and geodesics-based contrastive learning. Deformable dynamics and occupancy representations are learnt from simulation data. More specifically, the framework is built with the NVIDIA PhysX simulator used to generate a deformable dynamics dataset. By learning the deformation dynamics from simulation ACID can provide a more accurate simulation in an evolving manipulation setup compared to methods that only model geometrical deformations such as 3DNS. We plan to experimentally validate its applicability in the following period.

5.2.1.3. PhysX

NVIDIA PhysX [21] can simulate complex physics interactions in real-time applications, particularly in the realm of soft-body dynamics. By leveraging the parallel processing capabilities of GPUs, PhysX achieves high-performance simulations, enabling real-time interaction and rendering of soft bodies. Soft body dynamics involve the simulation of deformable objects that respond realistically to external forces, such as gravity, collisions, and pressure. PhysX employs the FEM to handle the complexities of soft body dynamics, ensuring that interactions between objects and the environment are faithfully represented. To achieve this the objects are assigned properties such as elasticity, friction, and mass distribution. Given that the focus of PhysX is on real-time applications the set of physical properties is kept small. In the scenarios considered in Craeft, the main external force is the one applied by the tool on the object that is manipulated.

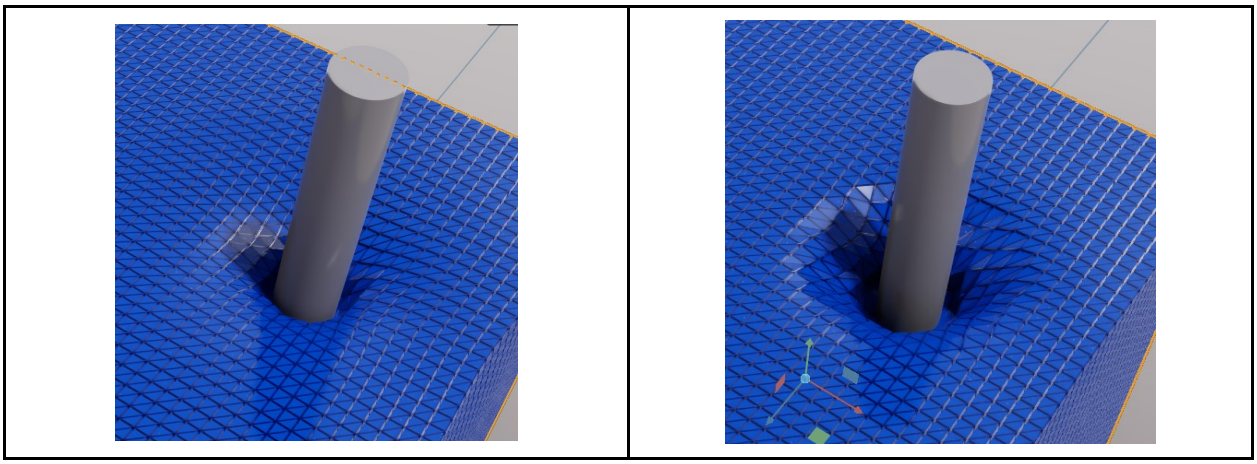


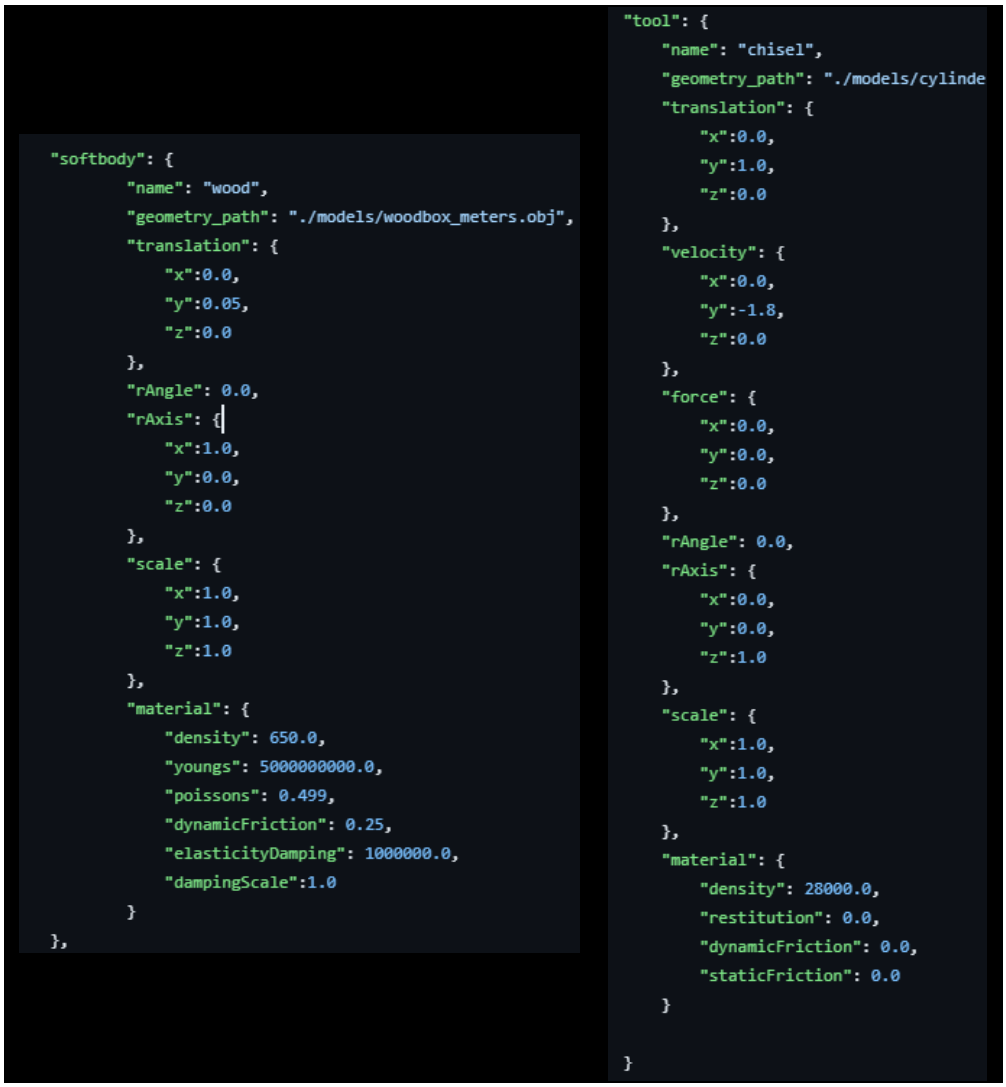
Figure 21. : Example of object deformation simulation when hit by a rigid rod using PhysX. The depicted blocks are the elements of the FEM method used by the simulator.

5.2.2. Implementation

We performed PhysX experiments using two different approaches. Initially, we used USD Composer from Omniverse [22]. Using its graphical tool, we were able to easily perform some first experiments to establish the proof of concept. Subsequently, we used the PhysX SDK directly, to build a platform that enables us to have full control over the performed manipulations, enabling us to assess the results quantitatively in an automated fashion. In the following, we describe the basic elements of the PhysX SDK-based platform

5.2.2.1. Input/Output

Physical properties and initial states of the actors are defined in JSON files and imported using the boost property tree library [24]. Different sets of parameters are defined for the soft bodies (materials that are manipulated) and rigid bodies (tools). Figure 22 shows example properties.



```
"softbody": {
        "name": "wood",
        "geometry_path": "./models/woodbox_meters.obj",
        "translation": {
            "x":0.0,
            "y":0.05,
            "z":0.0
        },
        "rAngle": 0.0,
        "rAxis": {
            "x":1.0,
            "y":0.0,
            "z":0.0
        },
        "scale": {
            "x":1.0,
            "y":1.0,
            "z":1.0
        },
        "material": {
            "density": 650.0,
            "youngs": 5000000000.0,
            "poissons": 0.499,
            "dynamicFriction": 0.25,
            "elasticityDamping": 1000000.0,
            "dampingScale":1.0
        }
},
```

```
"tool": {
    "name": "chisel",
    "geometry_path": "./models/cylinde
    "translation": {
        "x":0.0,
        "y":1.0,
        "z":0.0
    },
    "velocity": {
        "x":0.0,
        "y":-1.8,
        "z":0.0
    },
    "force": {
        "x":0.0,
        "y":0.0,
        "z":0.0
    },
    "rAngle": 0.0,
    "rAxis": {
        "x":0.0,
        "y":0.0,
        "z":1.0
    },
    "scale": {
        "x":1.0,
        "y":1.0,
        "z":1.0
    },
    "material": {
        "density": 28000.0,
        "restitution": 0.0,
        "dynamicFriction": 0.0,
        "staticFriction": 0.0
    }

}
```

Figure 22. Left: Soft body properties JSON. Right: Rigid body properties JSON.

Object geometry is imported using the Assimp library [69]. Assimp supports several formats, in our case the wavefront '.obj' file format is used for mesh definition [28]. The deformed meshes that result from the manipulations are also exported in the same format. Figure 23 shows the code snippet for the mesh import.

5.2.2.2. Simulation

The PhysX scenes contain objects called actors. Each actor has a physical state and properties that include position, orientation etc. Actor states evolve over time due to applied forces and interactions. In our case two actors are defined at each time step of the simulation: the tool as a rigid body actor, and the manipulated material as a soft body actor. Given the actor properties and initial states from the previous section, simulation is straightforward using the appropriate PhysX SDK API calls. The GPU pipeline is used since it is the only one supporting soft bodies. The simulation results are shown in the next section.

```
void createMeshAssimp(PxArray<PxVec3> &triVerts, PxArray<PxU32> &triIndices,
                                        const PxVec3 &pos, const PxVec3 &scale, const aiMesh *mesh)
{
        triVerts.clear();
        auto NumVertices = mesh->mNumVertices;
        triVerts.resize(NumVertices);
        for (int i = 0; i < NumVertices; ++i)
        {
                auto v = mesh->mVertices[i];
                triVerts[i] = PxVec3(scale.x * v.x, scale.y * v.y, scale.z * v.z);
                // std::cout << "Adding assimp vertex " << v.x << v.y << v.z << std::endl;
        }

        triIndices.clear();
        auto NumFaces = mesh->mNumFaces;
        for (int i = 0; i < NumFaces; ++i)
        {
                auto f = mesh->mFaces[i];
                triIndices.pushBack(f.mIndices[0]);
                triIndices.pushBack(f.mIndices[1]);
                triIndices.pushBack(f.mIndices[2]);
        }
```

Figure 23. Mesh import using Assimp.

5.2.3. Results

In this section, we present some preliminary results from evaluating the 3DNS and PhysX methods. Simulia (Dassault Systems, 2023) was used to generate 3D scenes that show the manipulation of a deformable timber block using a rigid metallic rod. The deformation caused by the timber in Simulia is considered as ground truth. The physical parameters of the material used in Simulia are shown in Table 1. Our goal is to accurately reproduce this deformation using 3DNS and PhysX.

Table 4. Timber material properties from Simulia.

Density	650 kg/m^3
Elastic modules	– E_1 = 12 GPa – E_2 = 1.848 GPa – E_3 = 1.2 GPa
Poisson ratios	– v_{12} = 0.4 – v_{13} = 0.5 – v_{23} = 0.65

Shear modules	– G_{12} = 1.2 GPa – G_{13} = 8 GPa – G_{23} = 6 GPa
Plastic **(Type: isotropic)**	– Yield stress 1 = 30 MPa – Yield stress 2 = 40 MPa – Plastic strain 1 = 0 – Plastic strain 2 = 0.2

Table 5. Aluminium material properties from Simulia.

Density	2700 kg/m^3
Elastic modules	70 GPa
Poisson ratios	v=0.27
Plastic **(Type: isotropic)**	– Hardening: Johnson-Cook – A = 1.48 GPa – B = 3.41 GPa – n= 0.18 – m = 0.859 – Melting Temperature = 5000 K – Transition Temp = 1

5.2.3.1. 3DNS Evaluation

For 3DNS we select a suitable radially symmetrical brush. For this purpose, we use a smooth step function, which is a function that takes the value 0 for $x < 0$, the value 1 for $x > 1$, and goes from 0 to 1 in the interval $[0, 1]$ in a continuously differentiable increasing manner. If f is a smooth step function, then we can define a radially symmetric brush template, as follows:

$$b_T(x) = f(1 - |x|)$$

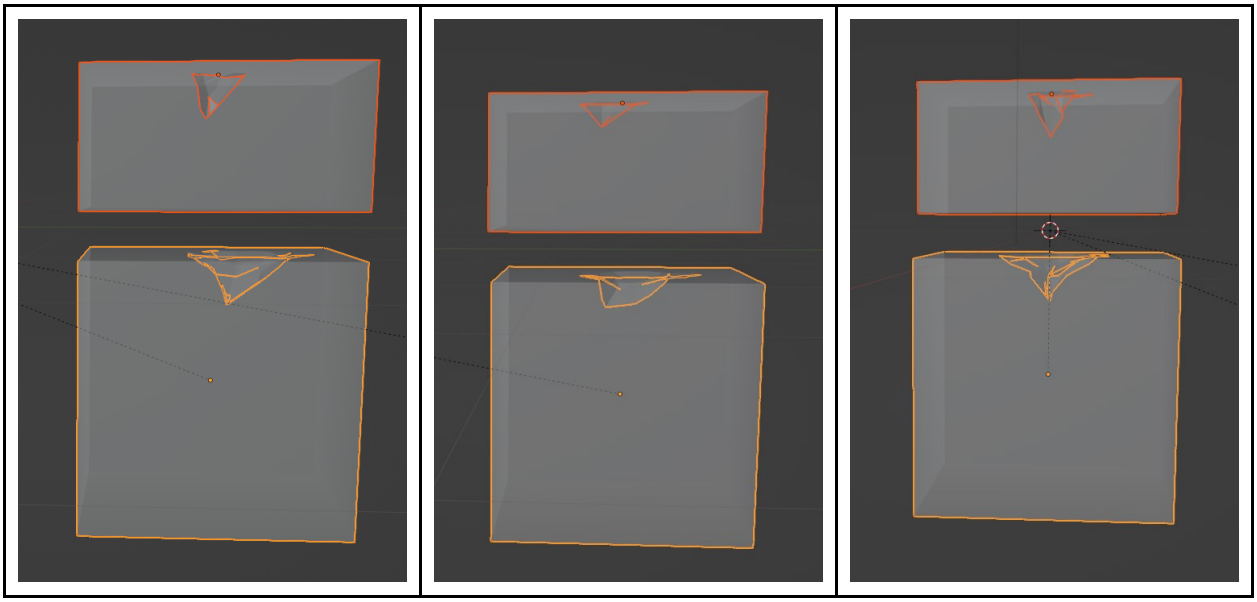
By modifying the intensity of the stroke, we can successfully reproduce the ground truth deformations for various applied force levels. One downside of 3DNS is that for each deformation a retrain of the network is required which requires several (~10) seconds to complete in a GPU thus real-time manipulation is not possible.

5.2.3.2. PhysX Evaluation

The PhysX experiments were performed in Nvidia's Omniverse platform. Two objects were defined in the simulated scene, the rod as a rigid object and the timber block as a deformable object. The scene is shown in Figure 21. The deformable object properties are shown in Table 6. At this moment there is not an officially supported database with these parameters for known materials but such a database will be released soon. Therefore, to define these properties we used the corresponding values from Simulia where possible. Since there is no one-to-one correspondence some experimentation and fine-tuning were required. The resulting deformations for strikes at three angles are shown in Figure 24. With PhysX, the simulation is performed in real-time. Figure 25 shows simulations for timber and aluminium.

Table 6. Timber material properties from PhysX.

Density	650 kg/m^3
Dynamic Friction	0.25
Young Modulus	5 GPa
Poisson Ratio	0.499
Elasticity Damping	1.0
Damping Scale	1.0



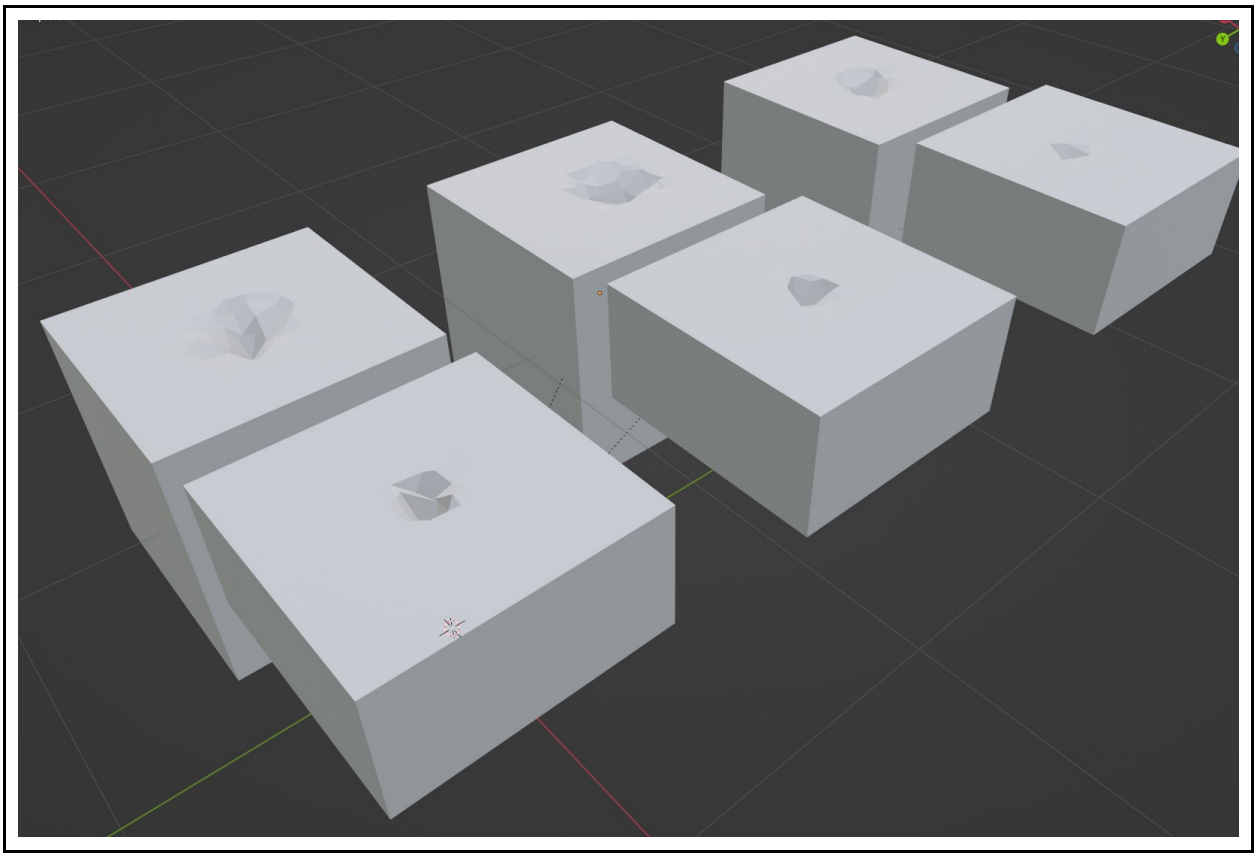
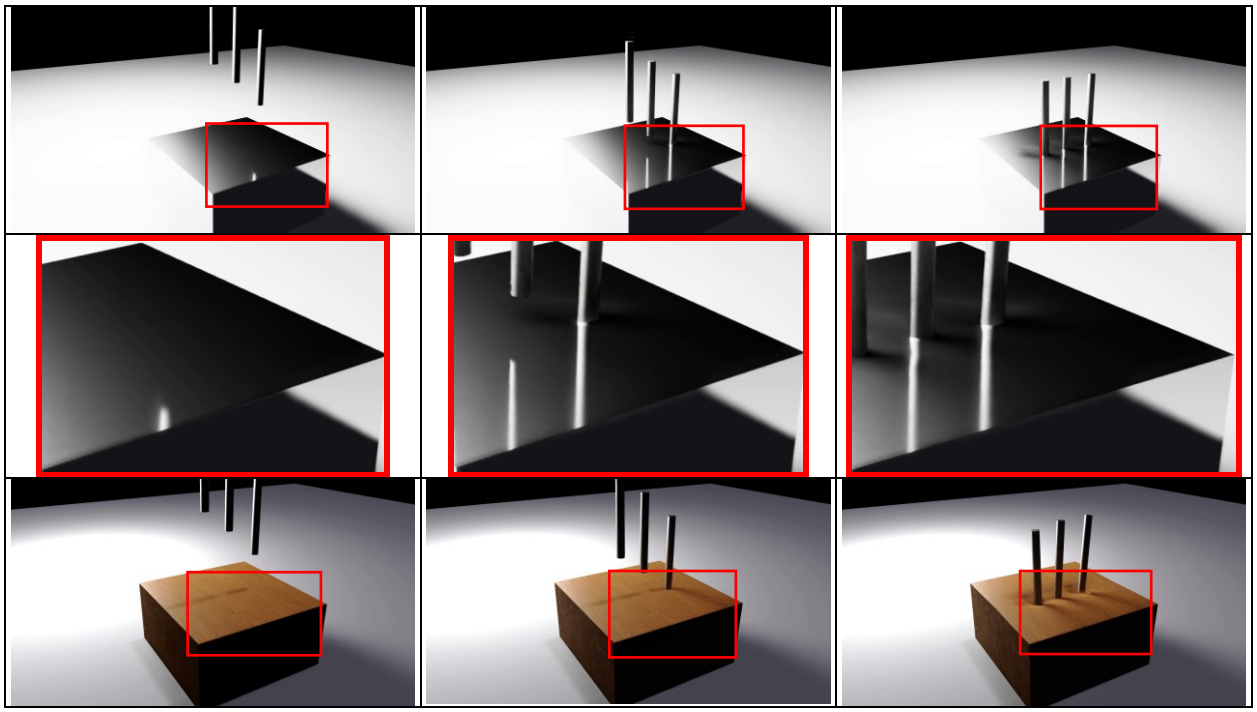


Figure 24. 3DNS results compared to Simulia, for three angles of incidence. Top: profiles of stroke results for 0º, 15º, and 60º (left to right, respectively). 3DNS results are plotted in red and Simulia results in orange. Bottom: 3D renderings of the results above, in the same order as above from left to right; Simulia results in the back and 3DNS results in the front.



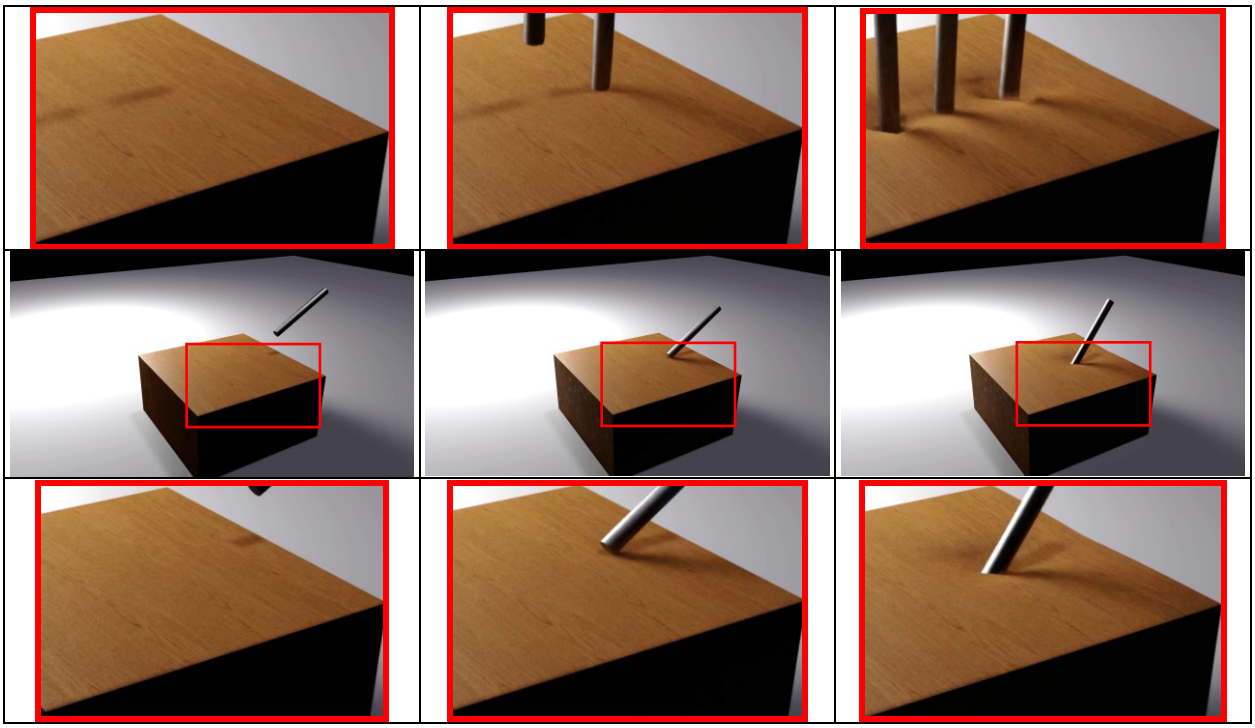


Figure 25. Simulation of the effect of cylindrical tools on different surfaces (aluminium and wood). Every row visualizes exemplar frames from different simulations. Different contact forces and angles are simulated, showcasing the different impacts on the deformation of the surface.

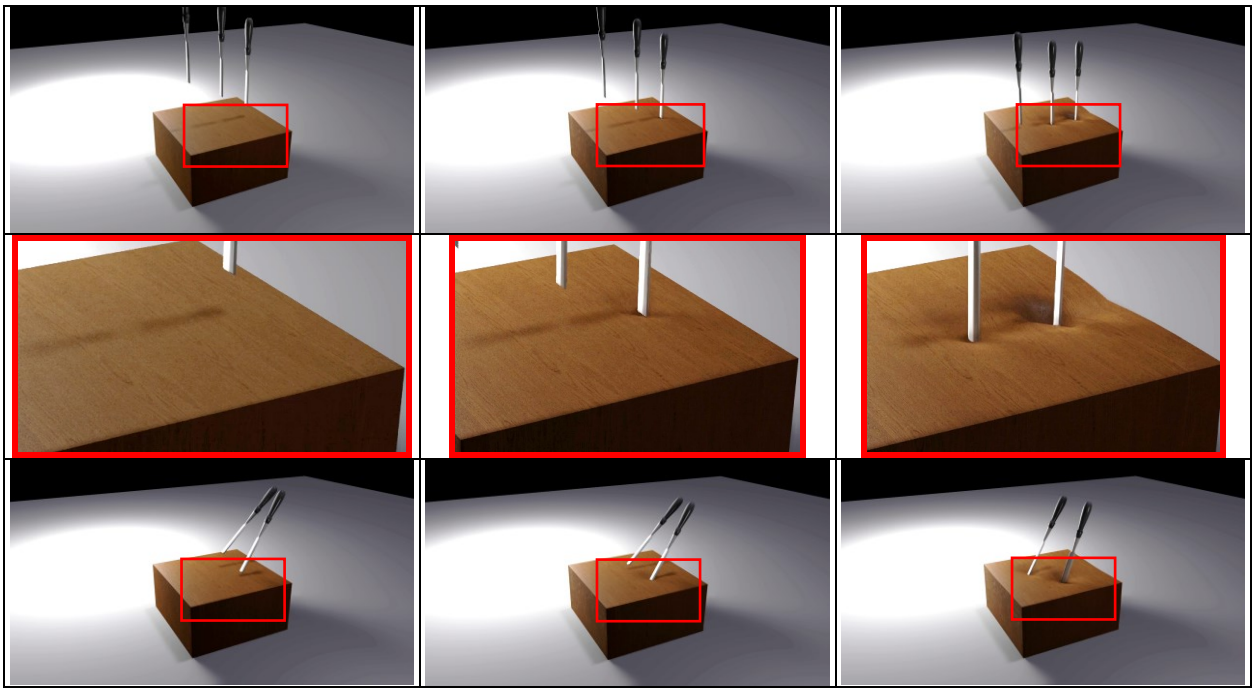


Figure 26. Simulation of the effect of chisel tools on a wooden surface. Every row visualizes exemplar frames from different simulations. Again, different contact forces and angles are simulated.

6. Process-specific simulators

Using the visualisation toolbox, we created a few simulators, which produce 3D models of objects in a process-specific fashion. That is, the simulation acknowledges the process that is followed for the objects to be crafted, as well as associated constraints, and produces results that are compatible with it. These simulators are used in the Design Studio and they are demonstrated in D5.1, “Craft Design Revisited”. In this section, we present their technical implementation; more extensive demonstrations of their results can be found in D5.1.

6.1. Moulded, cast, and sculpted objects

Texture-based renderings of matte objects are supported by conventional, off-the-shelf rendering engines and exhibit no challenge. The greatest challenge is the treatment of dielectric materials, such as metals, porcelain, and glass, which cannot be rendered using conventional texture-based methods. The result is the visualisation of the final appearance of custom moulded, cast, and sculpted pieces, from vases and sculptures to jewellery and decorative items, across a variety of materials. The benefit is the feedback during the design process. Visual updates enable quick iterations and refinements of the design, reducing the time and cost associated with physical prototypes. This is particularly important in sculpted, cast, and moulded products as the mistakes cannot be remedied.

The examples below demonstrate this for two 3D models. In Figure 27, shown is a 3D model rendered as is made from glossy plastic.

Figure 27. A 3D model of a sculpture rendered as made from glossy plastic and shown from three viewpoints.

Moreover, the software can be used to predict the appearance of more complex structures made from combined materials. The example Figure 28 shows the rendering of a composition made from an assortment of materials (plastic, metal, and glass of various colours).

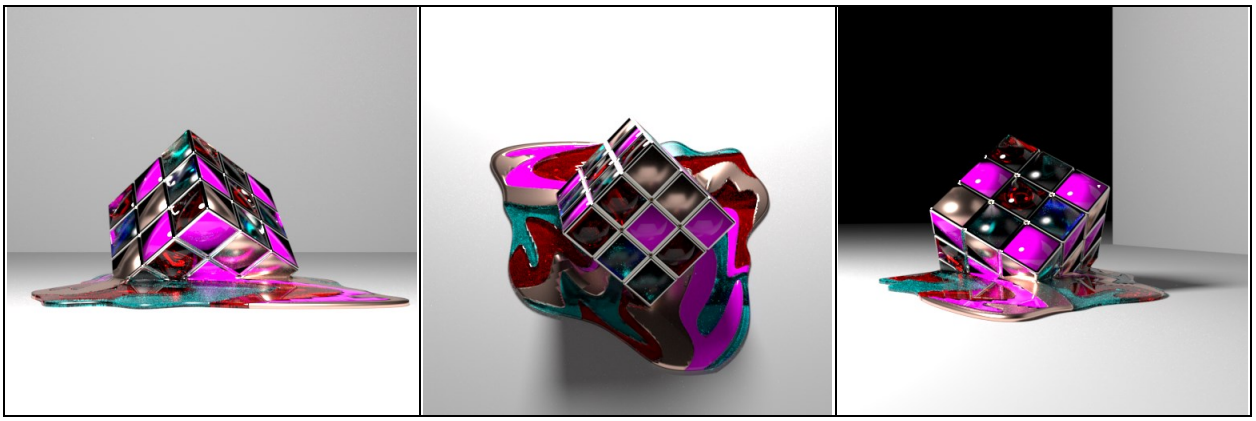


Figure 28. A 3D composition is simulated to be made from multiple materials and shown from three viewpoints.

6.2.1. Traditional stained-glass windows

Traditional stained-glass windows used the "came glasswork"[5] process of joining cut pieces of glass. Several types of metal have been used, initially lead, but also brass, zinc, and copper. Copper foil is an easy, versatile alternative to came and is particularly useful for small projects.
The provided application takes as input an image and automatically transforms it into a such came glasswork composition. The user specifies:

1. The thickness of the glass pieces and, correspondingly, that of the metallic rig.
2. The distance ("gap") between the glass pieces or, otherwise, the width of the soldering material that joins the glass pieces.
3. The material that the skeleton rig is made of.

The result is twofold:

1. A 3D design of what pieces would be needed for the composition
2. A photorealistic prediction of how the implementation of this design would look in a given environment.

The process of transforming an image into a stained-glass window involves several computational steps, including image processing, colour segmentation, morphological operations, and 3D modelling. The image is segmented into regions of approximately the same colour. The segmentation method employed is [13], but any other segmentation approach could be used. Once the image has been segmented into colour regions the mean colour of each region is calculated, in the HSV domain[6]. The mean colour is going to be assigned as the colour of the particular piece of stained glass needed to represent the corresponding image region.
The method comprises the following steps:

[5] Came glasswork is the process of joining cut pieces of art glass through the use of came strips or foil into picturesque designs in a framework of soldered metal.
[6] We prefer averaging colours in the HSV domain because the "mean colour" result is more compatible to human perception than averaging colours in the RGB domain.

1. **Image Reading and Pre-processing**: The initial step involves reading an image and preparing it for subsequent processing.
2. **Colour Quantisation**: The image is colour-quantised to reduce the number of distinct colours, facilitating the segmentation process.
3. **Colour Segmentation**: The quantised image is segmented based on colour regions, representing different pieces of stained glass. The mean HSV colour is computed for each segment.
4. **Morphological Image Operations**: Morphological operations, specifically opening and closing, are applied to the segmented image to refine the shapes and sizes of the segments, ensuring that they are appropriately sized for stained glass pieces.
5. **3D Model Creation**: The refined image segments are converted into 3D models representing the individual glass pieces.
6. **Skeleton Rig Modelling**: A binary template from the morphological operations is used to create a 3D skeleton rig that will support the stained-glass pieces.

In Figure 29, an illustrative example of the operation is shown.

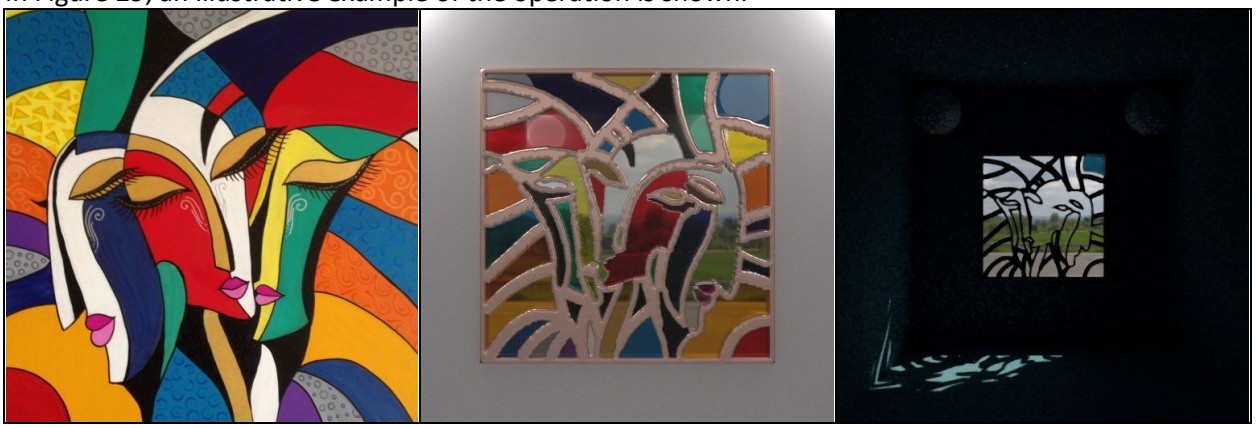


Figure 29. An art image (left), its rendering as a stained-glass window (middle), and a prediction of how light is projected from the external environment, in an empty room (right).

The application produces the 3D files for the glass pieces and the skeleton, as well as a configuration file for the toolbox to render the scene.

6.2.1.1. Glass pieces

Traditional stained-glass windows used the "came glasswork"[7] process of joining cut pieces of glass. Several types of metal have been used, initially lead, but also brass, zinc, and copper. Copper foil is an easy, versatile alternative to came and is particularly useful for small projects.

The provided application takes as input an image and automatically transforms it into a such came glasswork composition. The user specifies:

1. The thickness of the glass pieces and, correspondingly, that of the metallic rig.
2. The distance ("gap") between the glass pieces or, otherwise, the width of the soldering material that joins the glass pieces.

[7] Came glasswork is the process of joining cut pieces of art glass through the use of came strips or foil into picturesque designs in a framework of soldered metal.

3. The material that the skeleton rig is made of.

The result is twofold:

1. A 3D design of what pieces would be needed for the composition
2. A photorealistic prediction of how the implementation of this design would look in a given environment.

The image is read into the algorithm using a standard image processing library. Pre-processing may include resizing, converting to grayscale if necessary, and normalising the pixel values. Colour quantisation is performed using techniques such as K-means [139] clustering or median-cut algorithm [140]. The aim is to reduce the number of colours in the image while preserving the essential visual characteristics. This step is crucial for simplifying the segmentation process. Pixels are grouped into k clusters based on colour similarity. The centroid of each cluster represents the colour for that segment.

The quantised image is segmented into distinct regions based on colour. This can be achieved through region-growing algorithms, thresholding, or edge detection methods. Starting from seed points, regions are grown by adding neighbouring pixels that have similar colour values.

In the segmentation result, neighbouring regions are “connected”, meaning that their boundary pixels are neighbouring. As such their union covers entirely the template image and, thus, there is no room for the metallic skeleton that is to hold them together. Therefore, each region needs to be uniformly reduced in size (or “shrunk”) in such a way that the gap among pieces has the same thickness. As such, we cannot simply scale each region, because due to their different shapes, this would result in unequal distances (“gaps”) between regions. For this reason, morphological operations are employed to refine the segmented regions. Opening **r**emoves small objects from the foreground (usually bright regions) of the image. This is done by an erosion operation followed by a dilation. Closing fills small holes in the foreground. This is achieved by a dilation operation followed by an erosion. These operations help in refining the boundaries of the segments, ensuring they are well-defined and suitable for conversion into glass pieces.

Each segment, now representing a piece of stained glass, is extruded into a 3D model. The thickness of the glass and the dimensions of the segment are defined based on the image resolution and the intended real-world dimensions. The 2D segment is given a uniform thickness to create a 3D object. A mesh is generated for each segment to represent it in 3D space.

To create the 3D model of a piece of glass out of its image region we execute the following steps:

1. Triangulate the image region using the Delaunay method [19].
2. Trace the contour boundary points of the region, using the method in [14].
3. Convert the 2D region points into 3D points by adding zero (0) as their Z-coordinate.
4. Replicate the 3D points of step #3 and assign them as Z-coordinate the thickness of the stained-glass parameter provided by the user.
5. Replicate the triangles found in step #2 for the new points generated in step #4.
6. Identify the 3D boundary points of the point sets, produced in steps #3 and #4, using the 2D points from step #2 as pivots.
7. Create boundary triangles, by following the 3D boundary points from step #6, creating two triangles for a pair of consecutive boundary points.

6.2.2.2. Skeleton rig

To create the skeleton rig, the procedure is as follows. We start with the binary image of the template. A "padding" is first created to represent the outer part of the rig (see Figure 30).

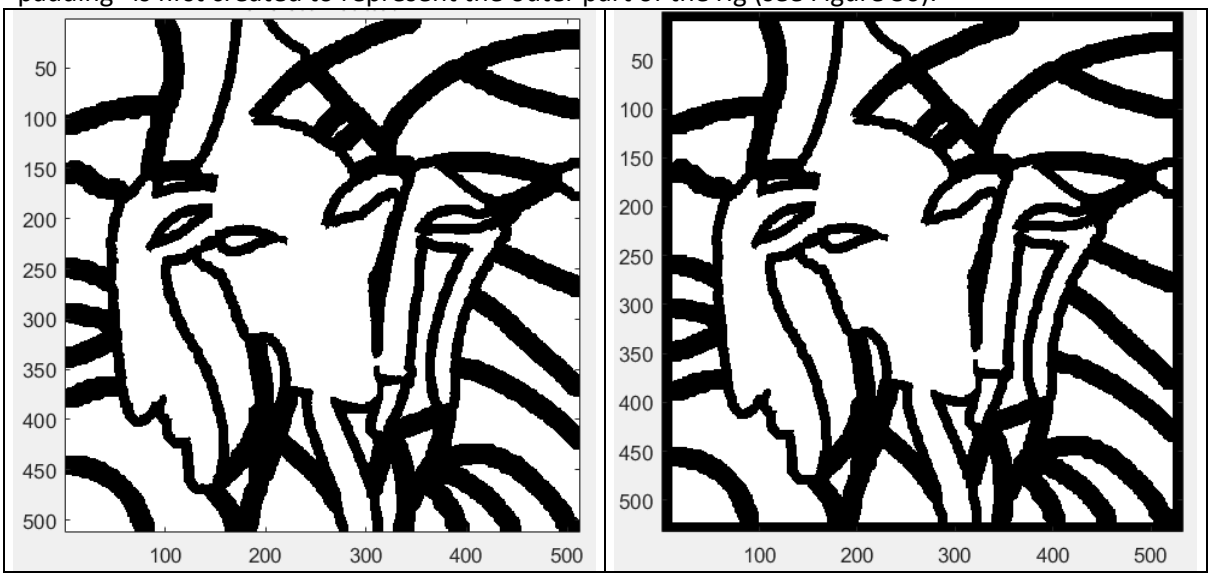


Figure 30. Original (left) and padded (right) template image.

After that we need to create the 3D model of the skeleton rig, from the template image; in Figure 30, this corresponds to creating a 3D model for the region occupied by black pixels. This is achieved as follows. For each point $\mathbf{p} = (p_x, p_y)$ of the template except points of last column and last row, we define 2×2 kernel with its top-left point at coordinates (p_x, p_y), which has points $(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4)$.

Then, we examine the number of black pixels in this kernel and define triangles by examining the number of black pixels in this kernel:

1. If this number is 4, then we define 2 triangles $(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3)$ and $(\mathbf{p}_3,\mathbf{p}_4,\mathbf{p}_1)$
2. If this number is 3 (meaning that a white pixel exists in the kernel), then we define 1 triangle, according to the following subcases, according to the location of the white pixel:
   a. $(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3)$
   b. $(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_4)$
   c. $(\mathbf{p}_1,\mathbf{p}_3,\mathbf{p}_4)$
   d. $(\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4)$

Case 1 and the four subcases of case 2 are illustrated in Figure 31, along with the triangle(s) defined in each case.

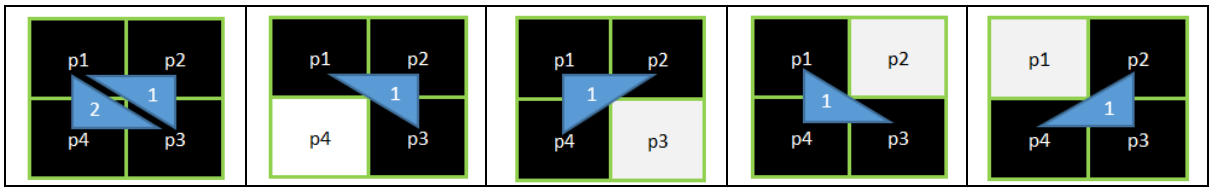


Figure 31. Triangulation cases are distinguished by the method for

We then duplicate the created triangles to provide "thickness" to the skeleton rig and upgrade them from 2D to 3D, in the same way that we did for the glass pieces: that is, one triangle has Z = 0 and the other has Z equal to the thickness of the rig.

Then, for each triangle, we order its vertices so that they are normal towards the external part of the skeleton; that is:

1. for the upper triangles (where, z = thickness), vertices are ordered so that their normal vector is $[0, 0, 1]^T$.
2. and the opposite ($[0, 0, -1]^T$) otherwise.

Using the same method as for the glass pieces we collect all outline border points. As we previously did for the triangles, we duplicate these points for the upper and lower parts of the rig. This task is illustrated in Figure 32.

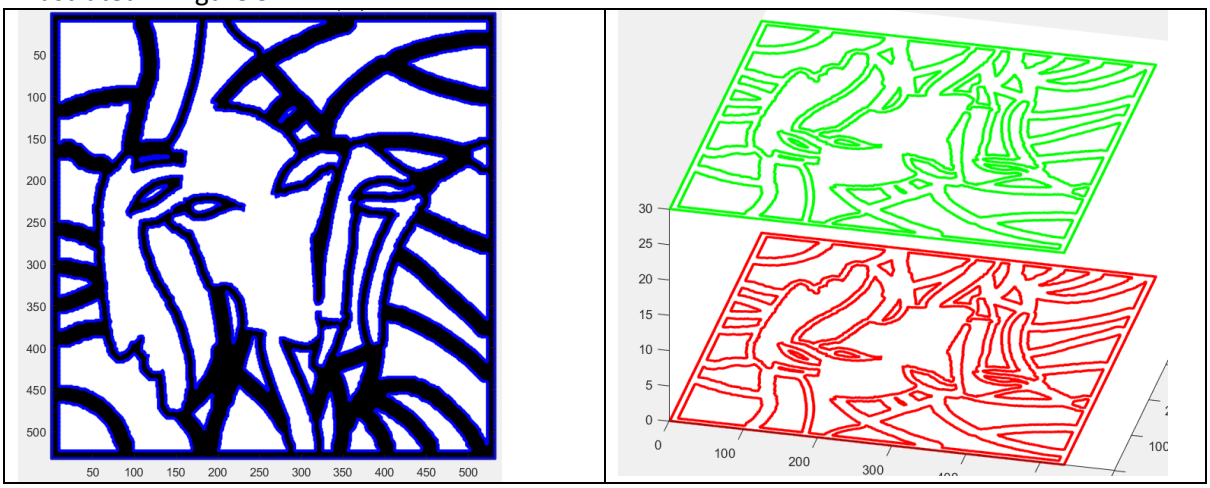


Figure 32. 2D boundary points of the skeleton rig and their conversion to two layers of 3D points.

For each upper point (green points in Figure 32) We collect its 8-connectivity neighbours and keep the orthogonally connected neighbours in case they exist; otherwise, we keep the diagonally connected neighbours. For each neighbour $\mathbf{p}_n$, we define 2 triangles:

1. ($\mathbf{p}$, $\mathbf{p}_{n\_bottom}$, $\mathbf{p}_{\_bottom}$)
2. ($\mathbf{p}$, $\mathbf{p}_n$, $\mathbf{p}_{n\_bottom}$)

where $\mathbf{p}_{n\_bottom}$ is the corresponding bottom point (from outlineBorderPoints3D_bottom) of $\mathbf{p}_n$ and $\mathbf{p}_{n\_bottom}$ the corresponding bottom point of $\mathbf{p}_n$ (see Figure 33, left).

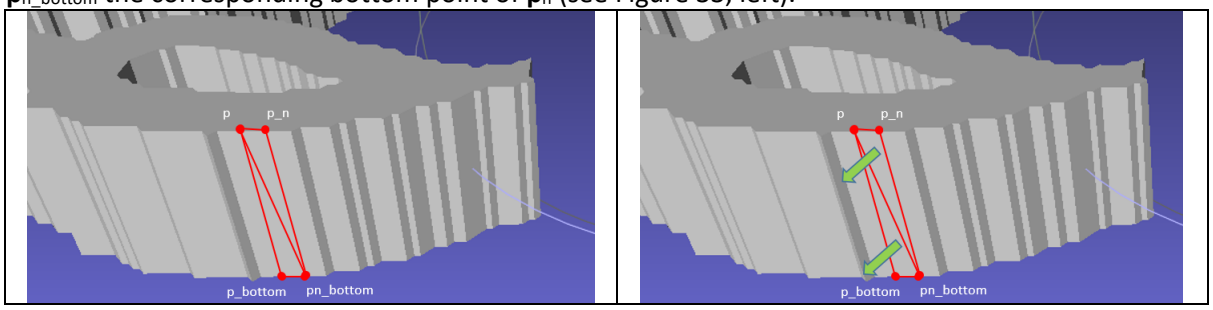


Figure 33. Triangulation of lateral faces of the skeleton rig (left) and definition of their vector normal (right).

Then we need to define the vector normal for each triangle and, thus, we order ($\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$) so that the normal points outwards to the skeleton. For this running example, the result is shown in Figure 34.

Figure 34. 3D model of a skeleton rig (left) and its rendering as a copper solid (right).

6.2.3. Composite objects

New types of products can be designed using the visualisation toolbox. We target objects that are composed of multiple materials and can be assembled. The individual pieces can be printed or handcrafted given their 3D models. This utility creates planar and 3D composite objects that can be assembled from multiple parts, taking as input conventional images, such as photographs or drawings.

6.2.3.1. Planar objects

A software utility was developed that takes as input an image and performs colour quantisation according to a number of colours determined by the user. It then distinguishes the masks corresponding to each colour. However, the quantisation result will typically contain holes, which would make the physical implementation of the design impossible for some materials or, at least, extremely tedious for a practitioner to craft or assemble. An example[8] is given in Figure 35, showing the original image and the quantised regions for the facial and hair-coloured pixels. Due to the intricate details, the 3D printing of such components would result in numerous small pieces making the assembly task difficult or impossible to achieve.

[8] The example is inspired by the "Shot Marilyns", a series of silkscreen paintings produced in 1964 by Andy Warhol, each canvas measuring 40 inches square, and each one portraying actor Marilyn Monroe. See https://en.wikipedia.org/wiki/Shot_Marilyns for more information.

Figure 35. Original image and selection of pixels that correspond to two colours after colour quantisation.

To address this problem, the utility performs a morphological transformation (an "open and close image" operation [74]) on each quantised layer to close small holes and smooth intricate boundary details that would complicate the physical crafting of the final artefact. The computation implements a simple image segmentation method based on colour quantisation and morphology; depending on style, numerous other segmentation variants can be utilised [136, 137]. For this approach and seven colours, the results are shown in Figure 36. In this case, the segmentation process results in 44 individual pieces. The hair (yellow_ and shadow (dark grey) layers give rise to multiple small pieces.

Figure 36. Original image and binary masks obtained from colour and morphological image segmentation.

Thereafter, we used the same method as for the stained-glass windows to create the pieces to be printed. However, this time we did not include the metallic skeleton rig, since the pieces should be glued together. The image below illustrates the expected result. Numerous materials and illumination configurations can be used to investigate the interaction of light with semi-transparent and translucent materials that are printed or moulded and used for internal decoration. In Figure 37 (top two rows), we illustrate the generality of the approach by simulating a light source, a titled colour pane, and two planar and grey surfaces. Clear-coloured glass is simulated in this case to create the coloured projection on the background wall. As demonstrated the simulation is capable of predicting the illumination effects

introduced by the designed artefact. In Figure 37, we changed the glass surface to be rough and, thus, scatter light more. In the bottom row, real 360 images are used for the environment illumination to predict how the design would suit specific environments. A video of this demonstration can be found at https://www.youtube.com/shorts/vKlDdEHxMqA.

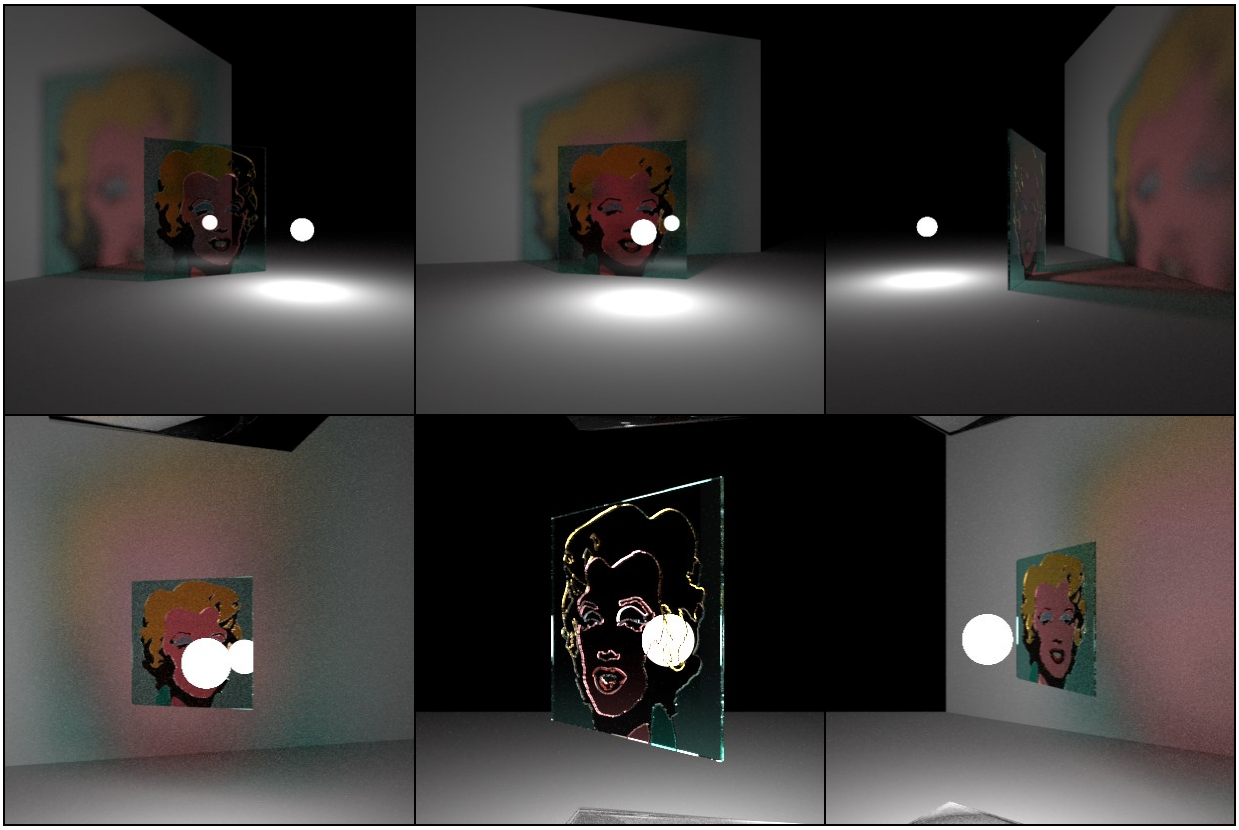


Figure 37. Top two rows: simulation experiments that illustrate the interaction of light with semi-transparent objects (see text). Bottom row: photorealistic previews of the designed artefact, in specific environments, from both sides.

6.2.3.2. 3D objects

Using the same artwork and the original photograph[9] used to create the silkscreens, we extended the utility to create sculptures composed of multiple pieces that a practitioner can assemble. In this case, each piece corresponds to a colour. To create a 3D sculpture, the original photograph was treated by the method in [74] which provides depth estimation from a monocular image. The original photograph and the generated depth map are shown in Figure 38.

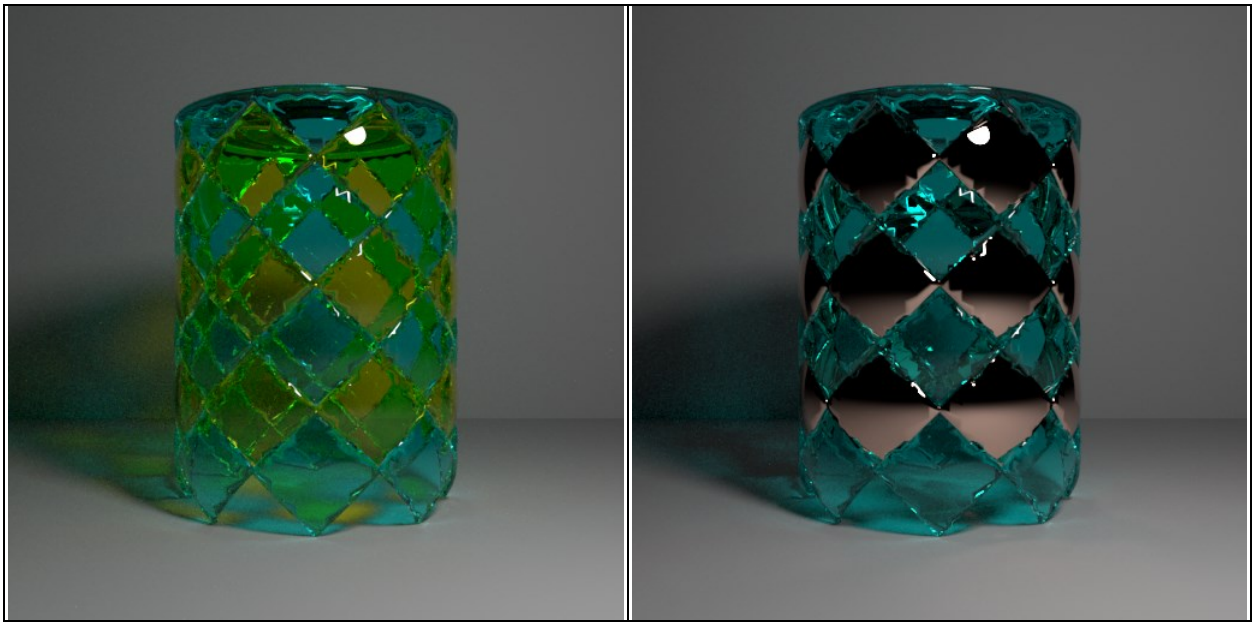
[9] Publicity portrait of Marilyn Monroe as Rose Loomis in the 1953 film Niagara. Photograph by Gene Kornman.

Figure 38. Original image and generated depth map, using [74].

Using conventional 2D image registration, we aligned the original photograph and Warhol's work. This way, the depth map associated with the photograph can be "transferred" to Warhol's work and provide a depth estimate for each location on the artwork. As shown in Figure 39 (top two rows), each segmented image region can be directly converted into a 3D solid. The assembly procedure exhibits similar constraints to solving a conventional jigsaw puzzle. In the example, three pieces must be placed before being able to insert the eyelids in place. The last row of the figure predicts the same structure as if printed from a more expensive material in a resin 3D printer.

Figure 39. Visual demonstration of the assembly of a multi-coloured printed statue, using matte filament; the last image shows the assembly result.

6.3. Lamps



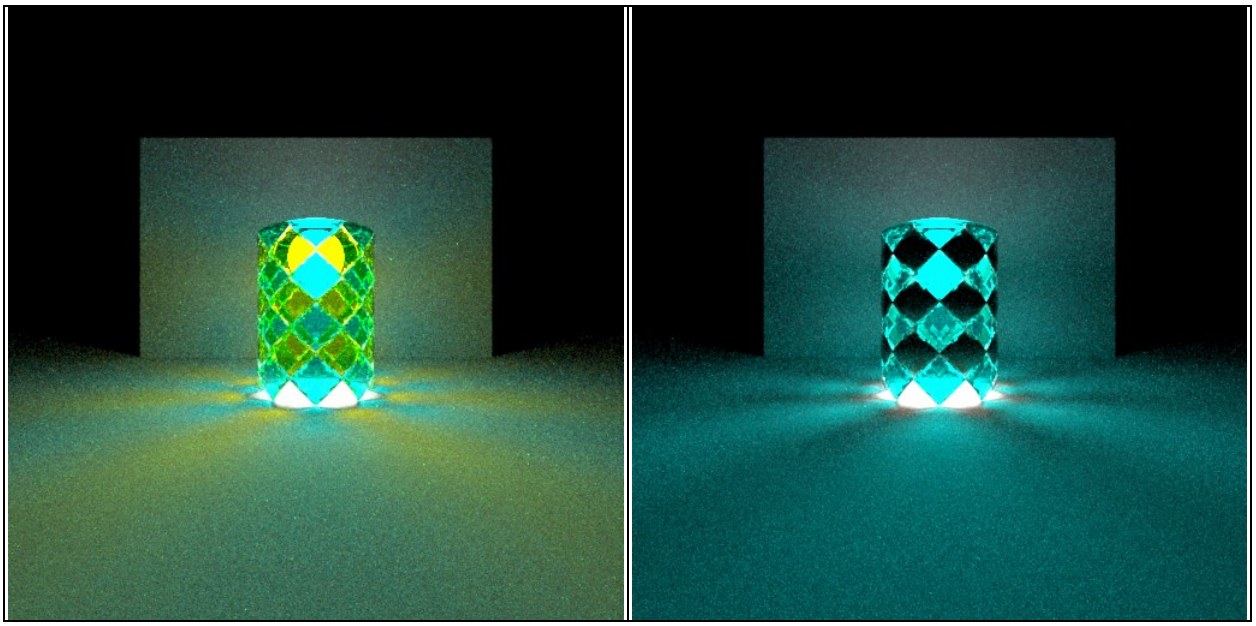
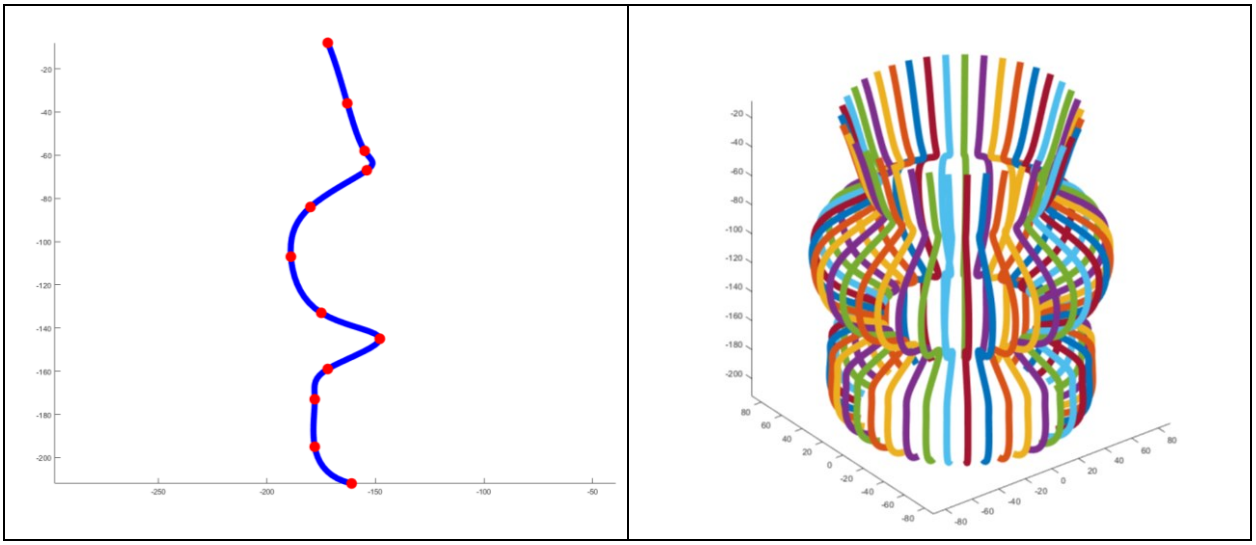


Figure 40. Rendering of 3D lamp models with external (top) and internal (bottom) light sources.

6.3. Cane working

In glassblowing, cane refers to rods of glass with colour; these rods can be simple, containing a single colour, or they can be complex and contain strands of one or several colours in the pattern. Cane working refers to the process of making a cane, and also to the use of pieces of cane, lengthwise, in the blowing process to add intricate, often spiral, patterns and stripes to vessels or other blown glass objects. Cane working is an ancient technique, first invented in southern Italy in the second half of the third century BC and elaborately developed centuries later on the Italian island of Murano.

An application is provided that simulates the appearance of cane work compositions. The user specifies the number, shape and dimensions of the canes that will comprise the result. In addition, a twisting parameter is provided that determines the final shape of the simulated artefact.



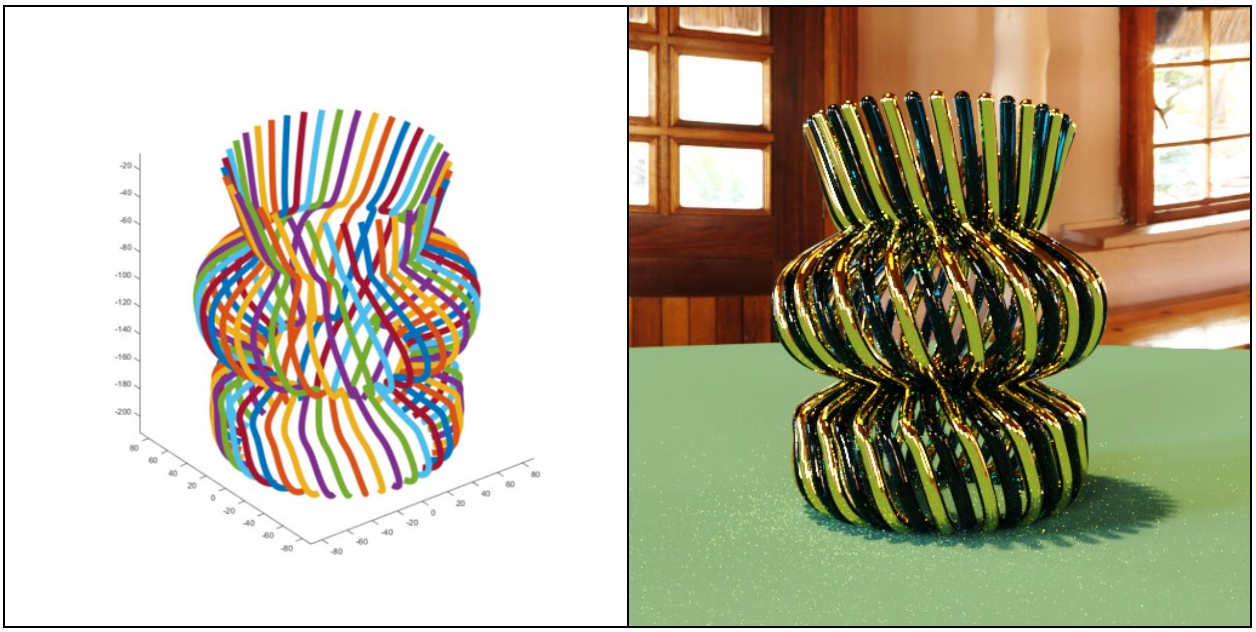


Figure 41. Numerical design of cane work structures and rendering (see text).

6.4. Metal engraving

Metal engraving is a process of cutting or carving designs, patterns, or text into the surface of a metal object. This technique has been used for centuries to decorate, personalize, and mark metal items. The process involves several key steps and can be applied to various types of metals, including gold, silver, brass, copper, and steel. The application predicts the appearance of engraved pieces of metal, by converting 2D carving designs into 3D models of carved items.

The process of converting 2D designs, such as engravings or line drawings, into 3D models involves creating a relief or surface representation of the design in a three-dimensional space.

The process begins with the creation of a 2D design. This design can be a simple geometric pattern, a line drawing, or any other visual representation. For example, we created an initial geometric design using sine and cosine functions, and later a continuous line drawing pattern, which mimics hand-drawn lines.

The 2D design is then translated into a 3D surface by interpreting the design as variations in height (or depth). Each point in the 2D space is assigned a height value, effectively transforming the flat design into a relief. In the examples provided, the height values were generated using mathematical functions that create smooth transitions, resulting in a visually appealing 3D surface.

The 3D model is created by generating a mesh, which is a collection of vertices (points in 3D space) and faces (polygons that connect the vertices). The vertices are derived from the 2D coordinates of the design, with an added height dimension. The faces are constructed by connecting adjacent vertices, forming the surface of the model. The height values (z-values) are often scaled to ensure the depth of the engraving is appropriate for the intended use. Normalization may also be applied to maintain consistency across different designs. Surface normals, which are vectors perpendicular to the faces of the mesh, are calculated to ensure the 3D model renders correctly in 3D environments. These normals are crucial for realistic lighting and shading effects.

The final step is exporting the 3D model to the OBJ format, a widely-used format that supports both the geometry (vertices and faces) and the associated surface normals. This format is compatible with most 3D modelling software and can be used for further editing, visualization, or fabrication.

The 3D models were visualized using 3D plotting tools to inspect the surface relief and ensure the design's integrity. Depth maps were generated to provide a 2D grayscale representation of the 3D surface, where the intensity of the grey colour represents the height. This visualization helps in understanding the depth variations in the design. These designs and their implementation as a piece of metal engraving are shown in Figure 42.

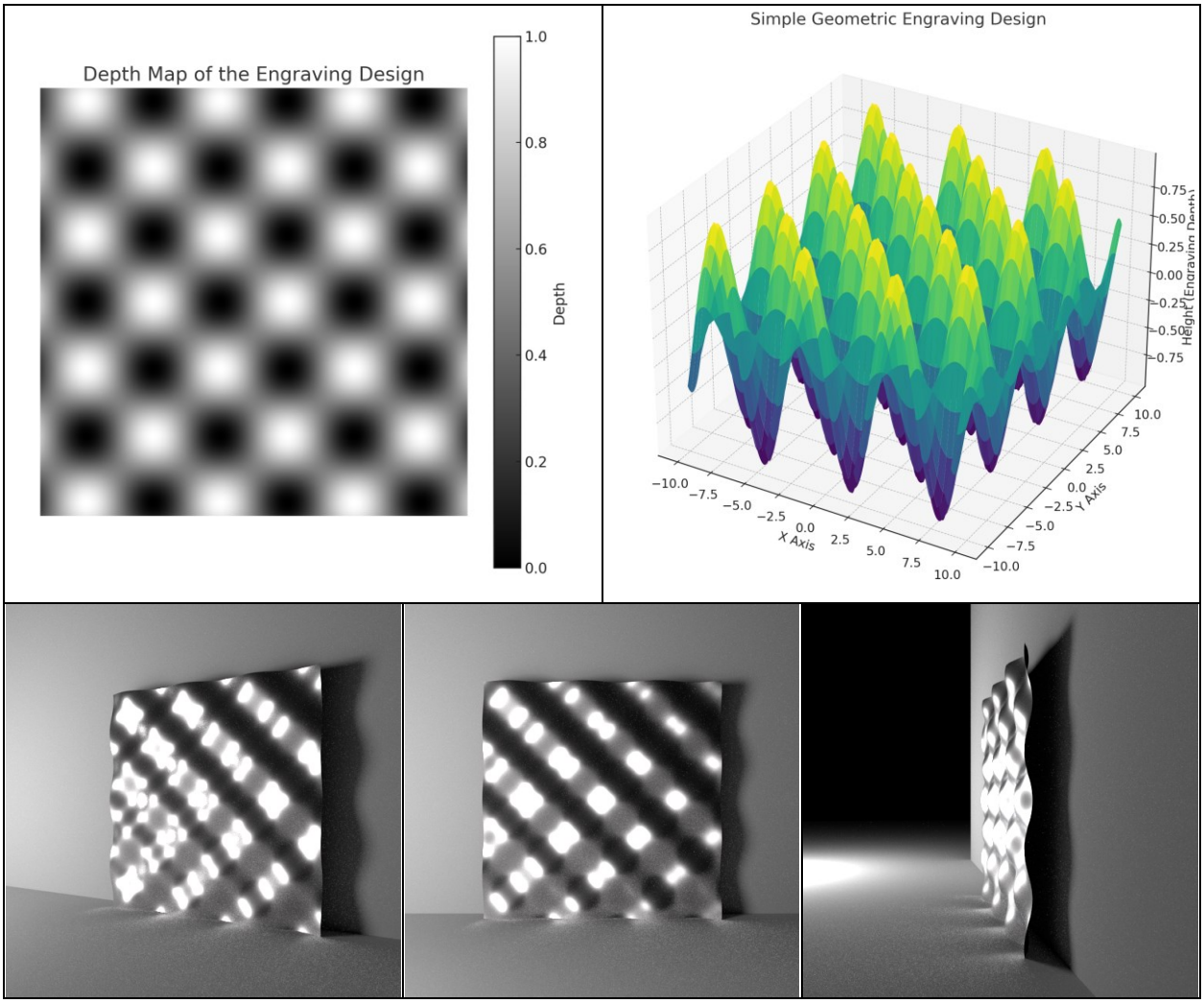


Figure 42. Numerical design of an engraving pattern (top) and renderings (bottom).

Converting 2D designs into 3D models involves a blend of artistic design and technical modelling. By translating 2D patterns into height data, we can create detailed 3D surfaces that add depth and dimension to otherwise flat designs. The process is versatile and applicable across various fields, making it a valuable technique in both creative and industrial domains. Figure 43, shows three engraving designs and their implementation on silver (top), gold (middle), and copper (bottom).

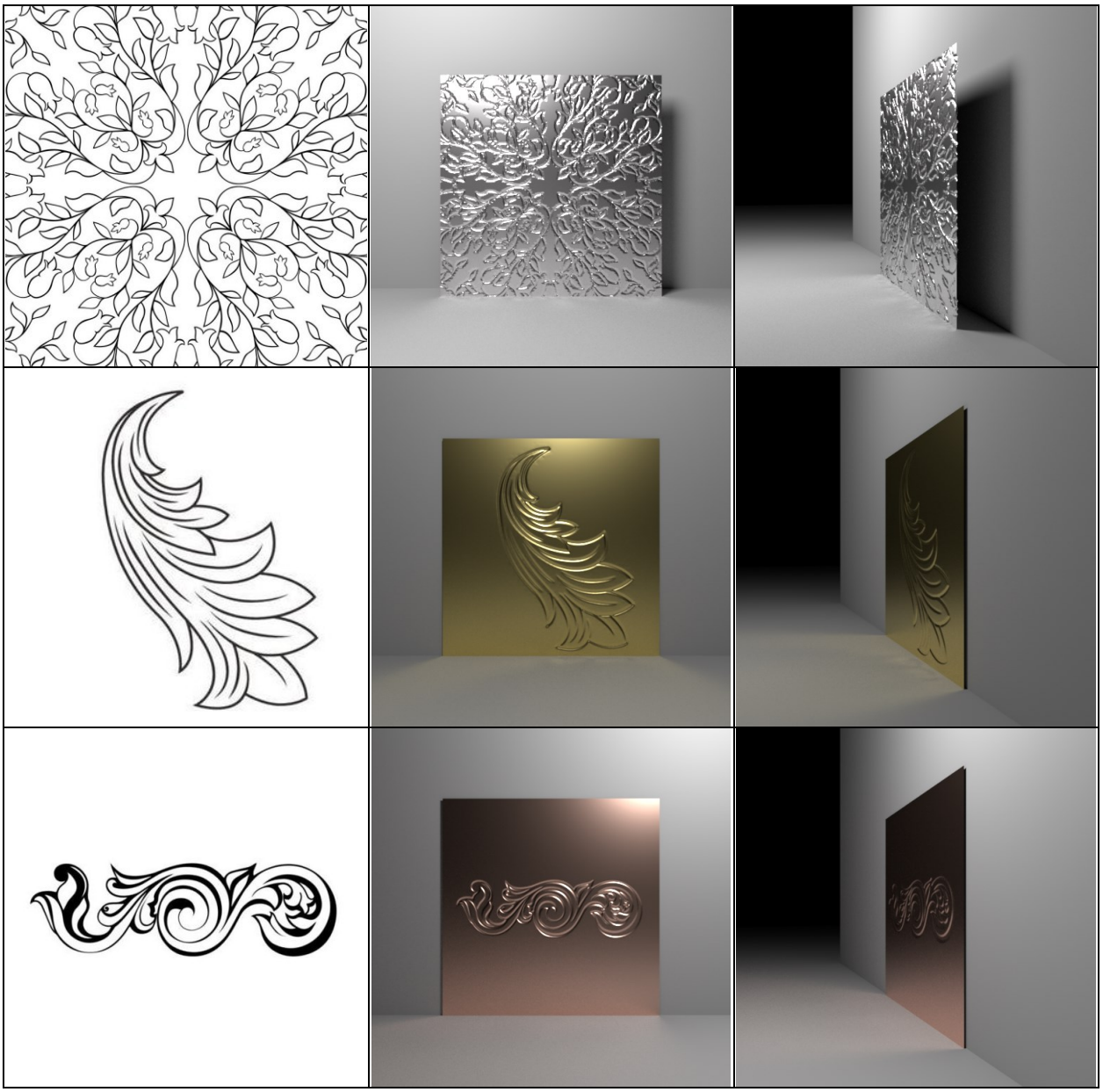


Figure 43. Design images of engravings and renderings of their 3D interpretations.

A more systematic illustration of carving strokes is shown in Figure 44, where a carving style index is transferred to a silver surface.

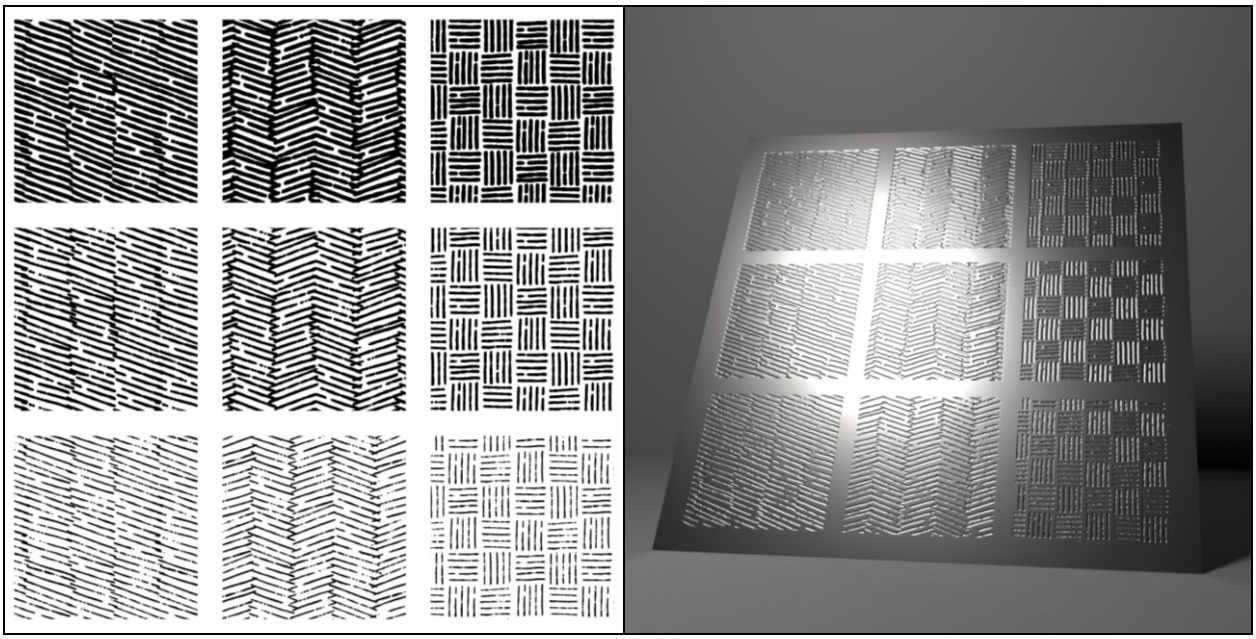


Figure 44. A style guide for carving transferred to a metallic surface.

6.5. Ceramics and glazes

Glazing in ceramics enhances both aesthetic and functional qualities. It provides a wide range of colours and finishes, enabling intricate decorative effects such as glossy, matte, or textured surfaces. Functionally, glazing makes ceramics waterproof, more durable, and easier to clean by creating a non-porous surface that resists scratches, stains, and bacteria. Additionally, glazes offer chemical and thermal resistance, protecting ceramics from acidic or alkaline substances and enhancing their suitability for culinary, laboratory, and oven use. Properly formulated glazes ensure food safety by preventing harmful substance leaching. Artistically, glazing allows for creative expression through techniques like layering and sgraffito, producing unique effects such as crackling and drips.

To produce the appearance of glazed ceramics, we have created a software utility that operates as follows. The input is a 3D model of the ceramic artefact. The model may be textured or not, in which case its absorption spectrum or material must be defined.

In both cases, the procedure is the same and requires as input the following parameters:

1. the thickness of the glaze (typically in the order of one to three millimetres)
2. the roughness of the glaze

Our software utility scales the 3D input mesh to inflate according to the required glaze thickness. This results in a second 3D model, which is larger than the original and is in the same coordinate frame. Since the two models are aligned, we proceed to use the algorithm in [15] to perform their Boolean subtraction. This results in a new 3D mesh, representing the difference between the two mashes.

Since the two meshes are aligned and the one is a scaling of the other the difference is a “hollow” mesh that represents the structure of the glazing material. A simple example below illustrates the procedure.

1. **Original Mesh**: Let a 3D sphere be the original mesh.

2. **Inflated Mesh**: Scaled original mesh uniformly in all directions. In this example, this is simply a larger sphere.
3. **Difference of Meshes**: The hollow mesh is created by subtracting the original mesh from the inflated mesh. This means the final object will look like a shell with the thickness determined by the scaling factor.

Figure 45, shows the visualization of this process:

1. **Original Mesh**: The blue sphere represents the original mesh.
2. **Inflated Mesh**: The red transparent sphere is the scaled version of the original mesh.
3. **Hollow Mesh**: The final image shows the hollow mesh, where the blue sphere (original mesh) is subtracted from the red sphere (inflated mesh), resulting in a spherical shell.

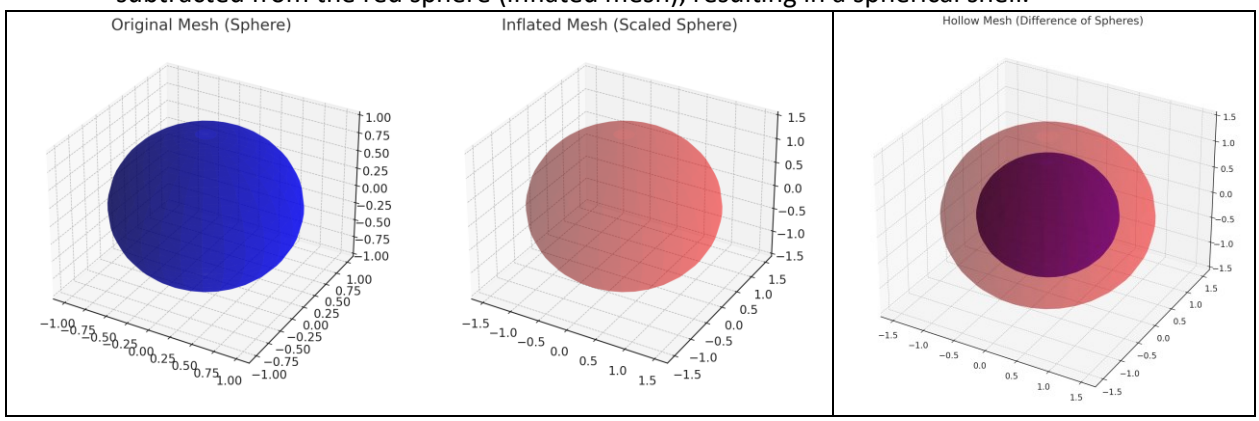
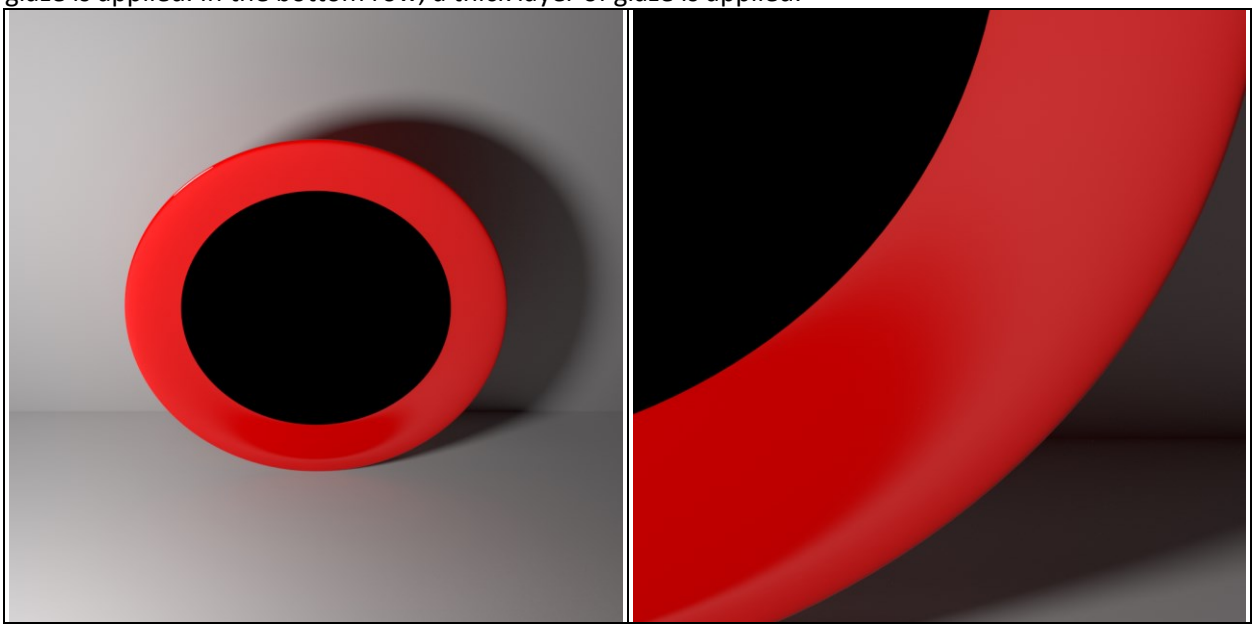


Figure 45. Example illustration of mesh subtraction. Left: original mesh. Middle: inflated mesh. Right: "hollow" mesh.

In Figure 46, we demonstrate the technique, for different types of glazes on a clay body, painted with underglaze colours (typically made from metal oxides). In the first row, no glazing is applied to the painted body, which is treated as a diffuse (or Lambertian) surface. In the middle row, a thin layer of glaze is applied. In the bottom row, a thick layer of glaze is applied.



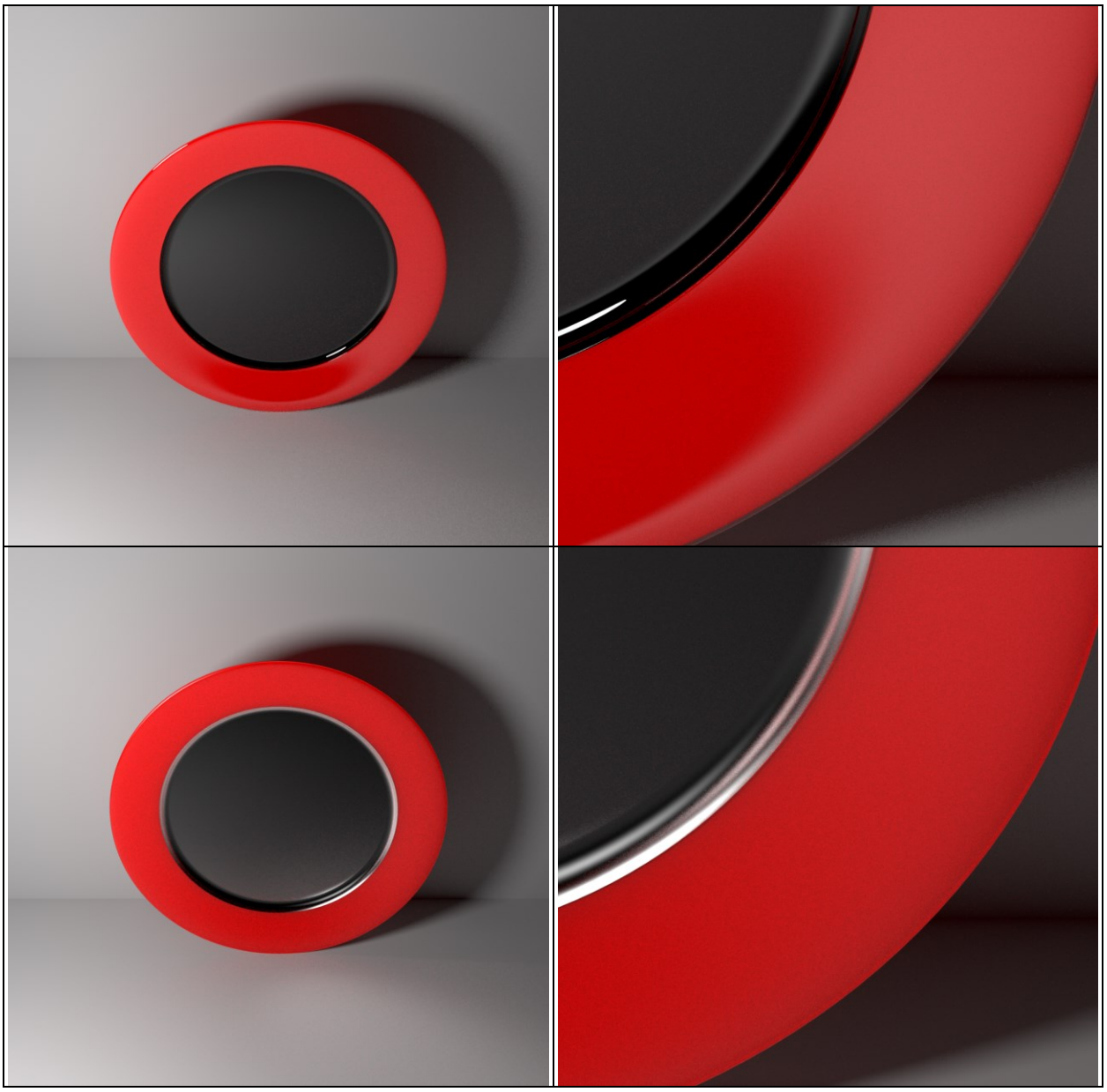


Figure 46. Visual simulation of glazed appearance. Top: no glazing. Middle: thin glazing. Bottom: thick glazing.

7. Conclusions

This deliverable has presented a detailed exploration of the tools and methodologies developed to enhance the simulation of craft-related activities through advanced computational techniques. By integrating physics-based rendering, real-time simulations, and artificial intelligence, we have created a robust platform that enables the realistic visualization and simulation of traditional crafting processes. These tools allow for the accurate representation of materials and actions, as well as provide a valuable resource for artisans, designers, and researchers to experiment with and refine their craft in a virtual environment.

Our work has demonstrated the potential of combining modern digital tools with the timeless principles of craftsmanship. The visualisation toolbox built on the Mitsuba 3 renderer has simplified the process of scene composition, enabling users to realistically simulate the appearance of objects and materials under various conditions. Additionally, the integration of the PhysX physics engine has facilitated real-time interaction within simulated environments, allowing for the development of craft-specific simulators that accurately replicate the physical processes involved in pottery, woodturning, glassblowing, and more.

Despite these advancements, the computational demands of physically realistic simulations remain a significant challenge. To address this, we introduced an artificial intelligence approach capable of approximating simulation results in real time, thereby bridging the gap between computational intensity and the need for immediacy in the design and testing phases.

Looking ahead, there are several promising avenues for expanding and refining the tools and methodologies presented in this deliverable. One important direction is the expansion of craft-specific simulators to encompass a wider array of traditional and modern techniques, such as metalworking, textiles, and advanced woodworking. This would allow for more comprehensive simulations across various disciplines. Additionally, further integration of sophisticated machine learning algorithms could significantly enhance the accuracy of real-time simulations, enabling more precise approximations of complex physical interactions. Improving the user interface to make these tools more intuitive and accessible, particularly for non-experts, is another critical area for development. This could involve creating custom interfaces tailored to specific crafts or incorporating augmented and virtual reality technologies to offer a more immersive and interactive experience.

Collaborating directly with practitioners and designers will be essential in refining these tools to meet practical, real-world needs. Such partnerships could also explore the educational potential of these simulations, using them as a means to teach and preserve traditional skills. Moreover, ongoing research into GPU acceleration and parallel computing could further bolster the real-time capabilities of our simulations, allowing for the accurate representation of increasingly complex scenes and interactions. Finally, the application of these tools in studying and enhancing the sustainability of craft practices represents a meaningful area of future research. By simulating the environmental impact of various materials and techniques, we can support artisans in making decisions that not only honour their craft but also contribute to a more sustainable future.

References

1. Liang, H. (2012). Advances in multispectral and hyperspectral imaging for archaeology and art conservation. *Applied Physics A*, 106, 309-323. https://doi.org/10.1007/s00339-011-6689-1
2. Aila, T., & Laine, S. (2009). Understanding the efficiency of ray traversal on GPUs. *Proceedings of the Conference on High Performance Graphics 2009*, 145-149. https://doi.org/10.1145/1572769.1572792
3. Jensen, H. W. (2001). *Realistic Image Synthesis Using Photon Mapping*. A K Peters. ISBN: 1568811470
4. Debevec, P. (2008). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In ACM SIGGRAPH 2008 Classes (pp. 1-10). https://doi.org/10.1145/280814.280864
5. Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. Proceedings of the European Conference on Computer Vision (ECCV), 405-421. https://doi.org/10.1145/3503250
6. Park, J. J., Sinha, S. N., Efros, A. A., & Snavely, N. (2021). Deformable Neural Radiance Fields. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 5791-5800. 10.1109/ICCV48922.2021.00581
7. Igarashi, T., Igarashi, Y., & Zorin, D. (2007). Interactive design of period styles for textile patterns. *ACM Transactions on Graphics*, 26(3), 1-7. https://doi.org/10.1145/3272127.3275105
8. Baxter, W. V., Scheib, V., Lin, M. C., & Manocha, D. (2001). DAB: Interactive haptic painting with 3D virtual brushes. *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 461-468. https://doi.org/10.1145/383259.383313
9. Yunsheng Tian, Jie Xu, Yichen Li, Jieliang Luo, Shinjiro Sueda, Hui Li, Karl D. D. Willis, and Wojciech Matusik. 2022. Assemble Them All: Physics-Based Planning for Generalizable Assembly by Disassembly. ACM Trans. Graph. 41, 6, Article 278. https://doi.org/10.1145/3550454.3555525
10. Eberly, D.H. 2004. "Game Physics", Morgan Kaufmann Publishers. ISBN 0123749034.
11. Kaufman, D., Edmunds, T. and D. Pai, 2010, “Fast Frictional Dynamics for Rigid Bodies”, ACM SIGGRAPH 2005. https://doi.org/10.1145/1073204.1073295
12. Trzepieciński T, dell’Isola F, Lemu HG. Multiphysics Modeling and Numerical Simulation in Computer-Aided Manufacturing Processes. *Metals*. 2021; 11(1):175. https://doi.org/10.3390/met11010175
13. Guru Prashanth Balasubramanian, Eli Saber, Vladimir Misic, Eric Peskin, Mark Shaw, Unsupervised color image segmentation using a dynamic color gradient thresholding algorithm, Volume 6806, Human Vision and Electronic Imaging XIII; 68061H (2008) https://doi.org/10.1117/12.766184
14. Satoshi Suzuki and others. Topological structural analysis of digitised binary images by border following. Computer Vision, Graphics, and Image Processing, 30(1):32–46, 1985. https://doi.org/10.1016/0734-189X(85)90016-7
15. Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. ACM Trans. Graph. 35, 4, Article 39. https://doi.org/10.1145/2897824.2925901
16. Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022. Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering. In Transactions on Graphics (Proceedings of SIGGRAPH) 41(4). https://doi.org/10.1145/3528223.3530099

17. F O Bartell, E. L. Dereniak, W. L Wolfe, "The Theory and Measurement of Bidirectional Reflectance Distribution Function and Bidirectional Transmittance Distribution Function" Proc. SPIE 0257, Radiation Scattering in Optical Systems, (3 March 1981); https://doi.org/10.1117/12.959611
18. Seung-Hwan Baek, Tizian Zeltner, Hyun Jin Ku, Inseung Hwang, Xin Tong, Wenzel Jakob, and Min H. Kim. 2020. Image-based acquisition and modeling of polarimetric reflectance. ACM Trans. Graph. 39, 4, Article 139 (August 2020), 14 pages. https://doi.org/10.1145/3386569.3392387
19. Delaunay, Boris (1934). "Sur la sphère vide" [On the empty sphere]. Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles (in French). 6: 793–800.
20. Dassault Systems, SIMULIA Simulation Software. https://www.3ds.com/products/simulia 2024.
21. Nvidia PhysX developer site. https://developer.nvidia.com/physx-sdk 2024
22. NVIDIA Omniverse, The platform for connecting and developing OpenUSD applications. Available at: https://www.nvidia.com/en-us/omniverse/ 2024.
23. Collins, J., Chand, S., Vanderkop, A., Howard, D., 2021. A Review of Physics Simulators for Robotic Applications. IEEE Access 9, 51416–51431. https://doi.org/10.1109/ACCESS.2021.3068769
24. Marcin Kalicinski and Sebastian Redl, 2024. Boost Property Tree Library, https://github.com/boostorg/property_tree.
25. Open Asset Import Library, https://github.com/assimp/assimp, 2024.
26. Shen, B., Jiang, Z., Choy, C., Guibas, L.J., Savarese, S., Anandkumar, A., Zhu, Y., 2022. ACID: Action-Conditional Implicit Visual Dynamics for Deformable Object Manipulation. https://doi.org/10.15607/rss.2022.xviii.001
27. Tzathas, P., Maragos, P., Roussos, A., 2022. 3D Neural Sculpting (3DNS): Editing Neural Signed Distance Functions.
28. Wavefront .obj file, 2024. https://en.wikipedia.org/wiki/Wavefront_.obj_file . Wikipedia.
29. P. Tzathas, P. Maragos, and A. Roussos, 3D Neural Sculpting (3DNS): Editing Neural Signed Distance Functions, IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), January 2023. https://doi.org/10.1109/wacv56688.2023.00450
30. Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. Bulletin of the American Mathematical Society, 49(1), 1-23.
31. Clough, R. W. (1960). The finite element method in plane stress analysis. Proceedings of the 2nd ASCE Conference on Electronic Computation, 8, 345-378.
32. Zienkiewicz, O. C., & Taylor, R. L. (2000). The Finite Element Method. Butterworth-Heinemann.
33. MacNeal, R. H. (1994). NASTRAN Theoretical Manual. MacNeal-Schwendler Corporation.
34. Bathe, K. J. (1996). Finite Element Procedures. Prentice Hall.
35. Cook, R. D., Malkus, D. S., & Plesha, M. E. (1989). Concepts and Applications of Finite Element Analysis. John Wiley & Sons.
36. Hibbitt, H. D., & Karlsson, B. I. (1978). ABAQUS User's Manual. Hibbitt, Karlsson & Sorensen, Inc.
37. Moaveni, S. (1999). Finite Element Analysis: Theory and Application with ANSYS. Prentice Hall.
38. Anderson, J. D. (1995). Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill.
39. Versteeg, H. K., & Malalasekera, W. (2007). An Introduction to Computational Fluid Dynamics: The Finite Volume Method. Pearson Education.
40. Dawson, C. (1993). Doom Game Engine. id Software.
41. Havok. (2004). Havok Physics Engine. Havok.
42. Hecker, C. (2010). Physics simulation in games. In Game Developer Conference Proceedings.
43. NVIDIA. (2008). NVIDIA PhysX Technology Overview. NVIDIA Corporation.

44. Slater, M., & Sanchez-Vives, M. V. (2016). Enhancing our lives with immersive virtual reality. Frontiers in Robotics and AI, 3, 74.
45. Mendiburu, B. (2020). 3D TV and 3D Cinema: Tools and Processes for Creative Stereoscopy. CRC Press.
46. Susi, T., Johannesson, M., & Backlund, P. (2007). Serious games: An overview. Technical Report HS-IKI-TR-07-001. University of Skövde.
47. Thompson, J. (2019). Level of Detail for 3D Graphics. Morgan Kaufmann.
48. Waveren, J. V. (2016). Real-time AI for Games: Techniques and Applications. CRC Press.
49. D. Pye, The nature and art of workmanship, Cambridge University Press, London, 1968.
50. Keller C, Keller J. Imagery in cultural tradition and innovation. Mind Cult Act. 1999;6(1):3-32. doi:10.1080/10749039909524711.
51. Aktas B, Mäkelä M. Negotiation between the Maker and Material: Observations on Material Interactions in Felting Studio. International Journal of Design. 2019.
52. Cutsuridis, V, Taylor, J. A Cognitive Control Architecture for the Perception-Action Cycle in Robots and Agents. Cognitive Computation, 2013;5:383–395 doi:10.1007/s12559-013-9218-z.
53. Raspall F. Design with Material Uncertainty: Responsive Design and Fabrication in Architecture. In: Thomsen M, Tamke M, Gengnagel C, Faircloth B, Scheurer F, editors. Modelling Behaviour. Cham: Springer; 2015. doi:10.1007/978-3-319-24208-8_27.
54. Iyobe M, Ishida T, Miyakawa A, Sugita K, Uchida N, Shibata Y. Development of a mobile virtual traditional crafting presentation system using augmented reality technology. Int J Space-Based Situat Comput. 2017;6(4):239–251.
55. F Fuchigami R, Ishida T. Proposal of a Traditional Craft Simulation System Using Mixed Reality. In: Barolli L, Takizawa M, Yoshihisa T, Amato F, Ikeda M, editors. Advances on P2P, Parallel, Grid, Cloud and Internet Computing. Cham: Springer; 2021. doi:10.1007/978-3-030-61105-7_32.
56. Edlin L, Liu Y, Bryan-Kinns N, Reiss J. Exploring Augmented Reality as Craft Material. In: Stephanidis C, Chen JYC, Fragomeni G, editors. HCI International 2020 – Late Breaking Papers: Virtual and Augmented Reality. Lecture Notes in Computer Science, vol 12428. Cham: Springer; 2020. doi:10.1007/978-3-030-59990-4_5.
57. Agelada A, Rizopoulos G, Flamos I, Kasapakis V. Users as Craftspeople: Demonstrating Traditional Crafts using Interactive Immersive Virtual Reality. 2022 IEEE International Conference on Artificial Intelligence and Virtual Reality; 2022. p. 245-247.
58. Rossau I, Skovfoged M, Czapla J, Sokolov M, Rodil K. Dovetailing: safeguarding traditional craftsmanship using virtual reality. International Journal of Intangible Heritage. 2019;14:104-120.
59. Hägele M, Nilsson K, Pires JN, Bischoff R. Industrial Robotics. In: Siciliano B, Khatib O, editors. Springer Handbook of Robotics. Springer Handbooks. Cham: Springer; 2016. doi:10.1007/978-3-319-32552-1_54.
60. Guo Z, Zhang Z, Li C. Robotic Carving Craft, Research on the Application of Robotic Carving Technology in the Inheritance of Traditional Carving Craft. Proceedings of the CAADRIA Conference; 2022 Apr 9-15; Sydney. p. 747-756. doi:10.52842/conf.caadria.2022.1.747.
61. Shaked T, Bar-Sinai KL, Sprecher A. Adaptive robotic stone carving: Method, tools, and experiments. Automation in Construction. 2021;129:103809. doi:10.1016/j.autcon.2021.103809.
62. Brugnaro G, Hanna S. Adaptive Robotic Training Methods for Subtractive Manufacturing. In: Acadia 2017 Disciplines & Disruption: Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture. Cambridge, MA: Acadia Publishing Company; 2017. p. 164-169.

63. Brugnaro G, Hanna S. Adaptive Robotic Carving. In: Willmann J, Block P, Hutter M, Byrne K, Schork T, editors. Robotic Fabrication in Architecture, Art and Design 2018. Cham: Springer; 2019. doi:10.1007/978-3-319-92294-2_26.
64. T. Shaked, K. Bar-Sinai, A. Sprecher, Adaptive robotic stone carving: Method, tools, and experiments, Automation in Construction, Volume 129, 2021, 103809, ISSN 0926-5805, doi:10.1016/j.autcon.2021.103809.
65. Brugnaro G, Figliola A, Dubor A. Negotiated Materialization: Design Approaches Integrating Wood Heterogeneity Through Advanced Robotic Fabrication. In: Bianconi F, Filippucci M, editors. Digital Wood Design. Lecture Notes in Civil Engineering, vol 24. Cham: Springer; 2019. doi:10.1007/978-3-030-03676-8_4.
66. Google, 3D Pottery, Available from: https://experiments.withgoogle.com/3d-pottery (Accessed on 29 July 2024).
67. Eyewind, Pottery Master, Available from: https://play.google.com/store/apps/details?id=com.create.pottery.paint.by.color (Accessed on 29 July 2024).
68. AZ Games. Master of Pottery, Available from: https://store.steampowered.com/app/1160490/Master_Of_Pottery/ (Accessed on 29 July 2024).
69. Grow, A., Dickinson, M., Pagnutti, J., Wardrip-Fruin, N., Mateas, M. (2017). Crafting in games. Digital Humanities Quarterly, 11(4).
70. Karastone, Knitting Simulator 2014. Available from: https://karastonesite.com/2014/07/19/knitting-simulator-2014/ (Accessed on 29 July 2024).
71. Chotrov D, Uzunova Z, Maleshkov S. Real-time 3D model topology editing method in VR to simulate crafting with a wood-turning lathe. 2019.
72. Xu R, Xu J. Research on Interactive Traditional Craft Diagram Model and Simulation System: Take Nanjing Rong Hua Craft as an Example. In: Proceedings of the International Conference on Advances in Energy, Environment and Chemical Engineering; 2016. p. 261-265. doi:10.2991/aeece-16.2016.54.
73. NVIDIA, PhysX System Software, Available from: https://developer.nvidia.com/physx-sdk (Accessed on 29 July 2024).
74. The Irregular Corporation, Woodwork Simulator, 2019. Available from: https://steamdb.info/app/510230/info/ (Accessed on 29 July 2024).
75. Murray J, Sawyer W. Virtual Crafting Simulator: Teaching Heritage Through Simulation. In: Proceedings of the International Conference on Education and New Learning Technologies; 2015. p. 7668-7675.
76. Eglash R. Ethnocomputing with Native American Design. In: Dyson LE, Hendriks M, Grant S, editors. Information Technology and Indigenous People. IGI Global; 2007. p. 210-219. doi:10.4018/978-1-59904-298-5.ch029.
77. Carre A, Dubois A, Partarakis N, Zabulis X, Patsiouras N, Mantinaki E, et al. Mixed-Reality Demonstration and Training of Glassblowing. Heritage. 2022;5(1):103-128. doi:10.3390/heritage5010006.
78. Canyon Art, WeaveIt, Available from: http://www.weaveit.com/ (Accessed on 29 July 2024).
79. Fiberworks, Fiberworks PCW, Available from: http://www.fiberworks-pcw.com/ (Accessed on 29 July 2024).
80. Arahne, ArahneWeave, Available from: http://www.arahne.si/ (Accessed on 29 July 2024).
81. pixeLoom, Available from: http://www.pixeloom.com/ (Accessed on 29 July 2024).
82. WeavePoint, Available from: http://www.weavepoint.com/ (Accessed on 29 July 2024).

83. Eisenstein, J., Softweave, WIF Visualizer, Available from: https://softweave.com/software/wif-visualizer/ (Accessed on 29 July 2024).
84. EAT - The DesignScope Company, Available from: https://www.designscopecompany.com/simulation-knitting/77/the-art-knitting (Accessed on 29 July 2024).
85. Torii T. Hand-painted yuzen-dyeing simulation for online handcraft experience. Computer Animation and Virtual Worlds. 2023;e2200. doi:10.1002/cav.2200.
86. Xu S, Mei X, Dong W, Zhang Z, Zhang X. Real-time ink simulation using a grid-particle method. Computer Graphics. 2012;36:1025-1035.
87. Nam-Ho K, Bhavani S, Ashok K. Introduction to Finite Element Analysis and Design. Wiley; 2018.
88. Fish J, Belytschko T, A First Course in Finite Elements, 2007, Wiley.
89. Baek C, Johanns P, Sano TG, Grandgeorge P, Reis PM. Finite Element Modeling of Tight Elastic Knots. Journal of Applied Mechanics. 2021;88(2):024501. doi:10.1115/1.4049023.
90. Crassous J. Discrete-element-method model for frictional fibers. Physics Review E. 2023;107(2):025003. doi:10.1103/PhysRevE.107.025003.
91. Inoue T. Tatara and the Japanese sword: the science and technology. Acta Mechanica. 2010;214:17–30. doi:10.1007/s00707-010-0308-7.
92. He Z, Xu J, Tran KP, et al. Modeling of textile manufacturing processes using intelligent techniques: a review. International Journal of Advanced Manufacturing Technology. 2021;116:39–67. doi:10.1007/s00170-021-07444-1.
93. Orlik J, Krier M, Neusius D, Pietsch K, Sivak O, Steiner K. Recent Efforts in Modeling and Simulation of Textiles. Textiles. 2021;1(2):322-336. doi:10.3390/textiles1020016.
94. Xie J, Guo Z, Shao M, Zhu W, Jiao W, Yang Z, et al. Mechanics of textiles used as composite preforms: A review. Composite Structures. 2023;304(2):116401. doi:10.1016/j.compstruct.2022.116401.
95. Fang G, Liang J. A review of numerical modeling of three-dimensional braided textile composites. Journal of Composite Materials. 2011;45(23):2415-2436. doi:10.1177/0021998311401093.
96. Li Y, Du T, Wu K, Xu J, Matusik W. DiffCloth: Differentiable Cloth Simulation with Dry Frictional Contact. ACM Transactions on Graphics. 2022;42(1):2. doi:10.1145/3527660.
97. Boisse P, Hamila N, Vidal-Sallé E, Dumont F. Simulation of wrinkling during textile composite reinforcement forming. Composites Science and Technology. 2011;71(5):683-692. doi:10.1016/j.compscitech.2011.01.011.
98. Durville D. Simulation of the mechanical behaviour of woven fabrics at the scale of fibers. International Journal of Material Forming. 2010;3(Suppl 2):1241–1251. doi:10.1007/s12289-009-0674-7.
99. Brown L. TexGen. In: Kyosev Y, Boussu F, editors. Advanced Weaving Technology. Cham: Springer; 2022. p. 253-291.
100. Research & Education Unit, Cal/OSHA Consultation Service, California, Department of Industrial Relations and the National Institute for Occupational Safety and Health. Easy Ergonomics: A Guide to Selecting Non-Powered Hand Tools. California Department of Industrial Relations and the National Institute for Occupational Safety and Health; 2004. Publication No. 2004-164.
101. Radwin R, Haney J. An Ergonomics Guide to Hand Tools. 1996. doi:10.13140/RG.2.1.4761.5446.
102. Dababneh A, Lowe B, Krieg E, Kong Y, Waters T. A Checklist for the Ergonomic Evaluation of Nonpowered Hand Tools. Journal of Occupational and Environmental Hygiene. 2005;1. doi:10.1080/15459620490883150.
103. Uicker J, Pennock G, Shigley J. Theory of machines and mechanisms. New York: Oxford University Press; 2011. ISBN 978-0-19-537123-9.

104. Bowser EA. An elementary treatise on analytic mechanics: with numerous examples. New York: D. Van Nostrand Company; 1884. p. 202–203.
105. Lemaître J, Desmorat R. Engineering Damage Mechanics: Ductile, Creep, Fatigue, and Brittle Failures. Springer; 2005.
106. Hashin Z. Failure criteria for unidirectional fiber composite. Journal of Applied Mechanics. 47(2):329-334. doi:10.1115/1.3153664.
107. Anderson T. Fracture Mechanics: Fundamentals and Applications. 4th ed. CRC Press; 2017.
108. Xu X, Needleman A. Numerical Simulations of Fast Crack Growth in Brittle Solids. Journal of the Mechanics and Physics of Solids. 1994;42(9):1397-1434.
109. Virigiri VKR, Gudiga VY, Gattu US, Suneesh G, Buddaraju KM. A review on the Johnson-Cook material model. Materials Today: Proceedings. 2022;62(6):3450-3456. doi:10.1016/j.matpr.2022.04.279.
110. Johnson GR, Cook WH. Fracture Characteristics of Three Metals Subjected to Various Strains, Strain Rates, Temperatures and Pressures. 1985.
111. Chen W, Han D. Plasticity for Structural Engineers. Springer; 1988.
112. Drucker D, Prager W. Soil Mechanics and Plastic Analysis or Limit Design. Quarterly of Applied Mathematics. 1952;10:157-165.
113. Wood DM. Soil Behaviour and Critical State Soil Mechanics. Cambridge: Cambridge University Press; 1990.
114. Frost H, Ashby M. Deformation-Mechanism Maps: The Plasticity and Creep of Metals and Ceramics. Pergamon Press; 1982.
115. Nabarro F, de Villiers H. The Physics of Creep: Creep and Creep-Resistant Alloys. Taylor & Francis; 1995.
116. Versteeg H, Malalasekera W. An Introduction to Computational Fluid Dynamics: The Finite Volume Method. 2nd ed. Pearson Education; 2007.
117. Glotzer S, Rapaport D. Computer Simulation of Liquids. Oxford University Press; 2004.
118. Dantzig J, Rappaz M. Solidification. EPFL Press; 2009.
119. Chao-Yang W, Beckermann C. A two-phase mixture model of liquid-gas flow and heat transfer in capillary porous media—I. Formulation. International Journal of Heat and Mass Transfer. 1993;36(11):2747-2758. doi:10.1016/0017-9310(93)90094-M.
120. Ferziger J, Perić M. Computational Methods for Fluid Dynamics. Springer; 2002.
121. Lewis R, Morgan K, Thomas H, Seetharamu K. The Finite Element Method in Heat Transfer Analysis. John Wiley & Sons; 1996.
122. Çengel Y, Ghajar A. Heat and Mass Transfer: Fundamentals and Applications. 5th ed. McGraw-Hill Education; 2014.
123. ISO. ISO 10303-21:2016 Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva: International Organization for Standardization; 2016.
124. Library of Congress. Document ID: fdd000507. 2020 Jan 23.
125. Tsai M, Prindible M. Wood Gouge-Capturing Human Skill. 2019. Available from: https://courses.ideate.cmu.edu/16-455/s2019/index.html%3Fp=973.html (Accessed on 29 July 2024)
126. Harrison C, Childs H, Gaither K. Data-Parallel Mesh Connected Components Labeling and Analysis. Eurographics Symposium on Parallel Graphics and Visualization; 2011. doi:10.2312/EGPGV/EGPGV11/131-140.
127. He L, Ren X, Gao Q, Zhao X, Yao B, Chao Y. The connected-component labeling problem: A review of state-of-the-art algorithms. Pattern Recognition. 2017;70:25-43. doi:10.1016/j.patcog.2017.04.018.

128. Seymour J. The Forgotten Arts: A practical guide to traditional skills. Angus & Robertson Publishers; 1984. p. 54. ISBN 0-207-15007-9.
129. Khaledi K, Rezaei S, Wulfinghoff S, Reese S, A microscale finite element model for joining of metals by large plastic deformations, Comptes Rendus Mécanique, Volume 346, Issue 8, 2018, Pages 743-755, ISSN 1631-0721, https://doi.org/10.1016/j.crme.2018.05.005.
130. Kumar RK, Babu AS. Finite element analysis and experimental study on metal joining by mechanical crimping. International Journal of Service and Computing Oriented Manufacturing. 2014;1:295-306. doi:10.1504/IJSCOM.2014.066489.
131. Mui G, Wu X, Hu K, Yeh C, Wyatt K. Solder joint formation simulation and finite element analysis. 1997 Proceedings 47th Electronic Components and Technology Conference; 1997; San Jose, CA, USA. p. 436-443. doi:10.1109/ECTC.1997.606207.
132. Anca A, Cardona A, Risso J, Fachinotti V. Finite element modeling of welding processes. Applied Mathematical Modelling. 2011;35(2):688-707. doi:10.1016/j.apm.2010.07.026.
133. Cao J, Akkerman R, Boisse P, Chen J, Cheng H, de Graaf E, et al. Characterization of mechanical behavior of woven fabrics: Experimental methods and benchmark results. Composites Part A: Applied Science and Manufacturing. 2008;39(6):1037-1053. doi:10.1016/j.compositesa.2008.02.016.
134. Vilfayeau J, Crépin D, Boussu F, Soulat D, Boisse P. Kinematic modelling of the weaving process applied to 2D fabric. Journal of Industrial Textiles. 2015;45(3):338-351. doi:10.1177/1528083714532114
135. Digital Image Processing (Third Edition) by Rafael C. Gonzalez and Richard E. Woods, ISBN 978-93-325-7032-0 (2008).
136. Nikhil R Pal, Sankar K Pal, A review on image segmentation techniques, Pattern Recognition, Volume 26, Issue 9, 1993, Pages 1277-1294, ISSN 0031-3203, https://doi.org/10.1016/0031-3203(93)90135-J.
137. S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 7, pp. 3523-3542, 1 July 2022, doi: 10.1109/TPAMI.2021.3059968.
138. Yang, L., Kang, B., Huang, Z., Xu, X., Feng, J., & Zhao, H. (2024). Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data. ArXiv, abs/2401.10891.
139. Lloyd, Stuart P. "Least squares quantization in PCM." Information Theory, IEEE Transactions on 28.2 (1982): 129-137.
140. (2008). Median Cut Algorithm. In: Furht, B. (eds) Encyclopedia of Multimedia. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-78414-4_36