Realistic architectural visualizations using HPC resources

White paper

1. Abstract

This white paper deals with designing processes to provide a high-fidelity architectural visualisation in the form of video footage or as an interactive scene exploration in virtual reality (VR). A powerful open-source 3D modelling software Blender [1] was used for these purposes.

As a use case, a design of a set of family houses has been provided. We elaborate on a processing pipeline covering data import, scene enhancement (materials, texture baking, lighting, additional objects population) and rendering. All of that is directly within Blender software. The computationally intensive parts of the process, represented by rendering, were done on the HPC cluster. Our methodology involves Blender's Cycles renderer for photorealistic rendering and baking.

We further elaborate on the VR technology and the use of the BHolodeck add-on (our extension of the Blender's VR Scene Inspection option) to transition an architectural design from a traditional 3D model to an interactive VR experience. This process facilitates a more immersive and comprehensive design review and allows viewers to virtually inhabit the proposed space, promoting better understanding and decision-making.

Our results underscore the potential of Blender as a comprehensive architectural visualisation tool that offers the possibility of high-quality photorealistic renderings and VR-ready experiences. This whitepaper, therefore, provides valuable insights for architects, designers, and stakeholders in using Blender and VR technology for visualisation and design evaluation.

2. Processing pipeline

The procedure for creating faithful visualisations from architectural data in VR or in the form of photorealistic video can be described via the simplified workflow depicted in Fig. 1. 3D architectural data is imported into Blender and processed for either VR or video purposes. In the case of VR, the Blender environment is directly used to provide the VR experience. The video presenting the original architectural design is rendered using HPC resources.

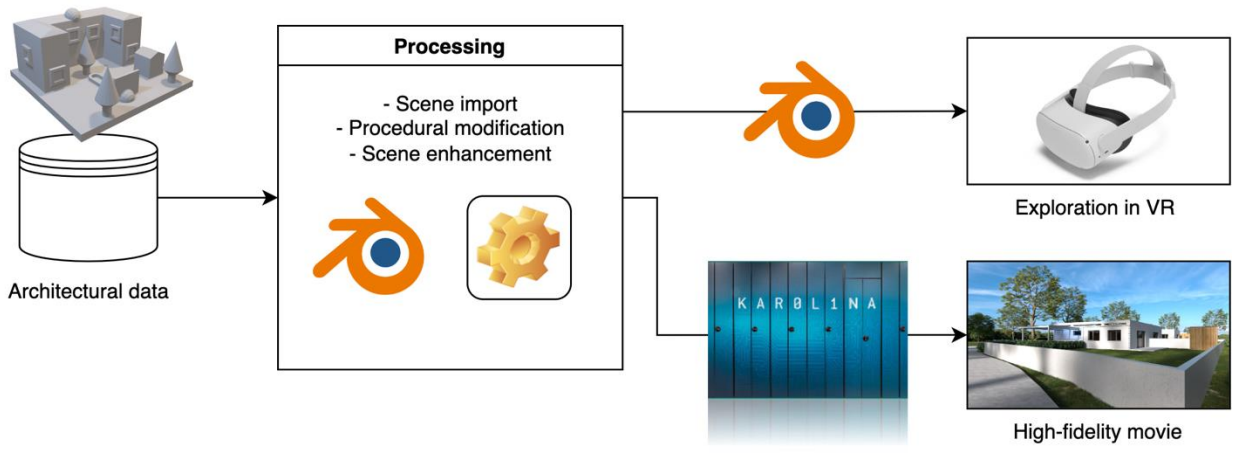


Fig. 1. Processing workflow applied to architectural data.

In the further text, we describe the main processing steps done in Blender to provide a 3D scene ready for exploration in VR or ready for video rendering.

2.1.Importing data

We have selected the FBX format to transfer data from architectural software to Blender. FBX is a proprietary file format (.fbx) created by Autodesk. It's a widely used format for interchanging 3D models, animations, and scenes between different 3D software. The FBX format has been used because it can store complex data, such as geometry, materials, textures, lighting, and animations, all within a single file.

The list of operations regarding the FBX file import into Blender:

- Go to File: In the top left corner of the screen, click on the "File" menu.
- Import FBX: Hover over "Import" to expand the submenu, and then click on "FBX (.fbx)".
- Choose the File: A file browser will appear. Navigate to the location of the FBX file, select it, and click on the "Import FBX" button in the top-right corner.
- Adjust Import Settings (Optional): On the right side of the file browser, there are several settings that we can adjust before importing the FBX file. These include options for scaling, rotation, importing materials and textures, and more. We can adjust these settings to match the requirements of the project.
- Import the FBX file in the current Blender scene.

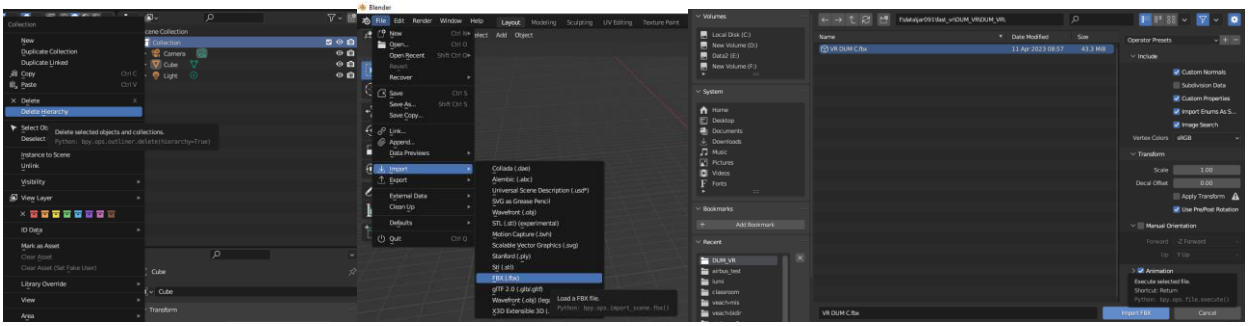


Fig. 2. Importing FBX file in Blender

2.2.Fix mesh

After importing all 3d objects, it is necessary to fix the mesh. This is especially important before baking the light into the textures as described later but it is a good practice recommended to be done after importing a 3D data. Blender provides a range of tools for fixing and refining 3D meshes. We have performed the following repair steps to fix the mesh to reach the visualisation goals. The provided steps can either be performed using Blender UI, or it is often more beneficial to write a simple Python script that performs the requested action using Blender operators. Created scripts can either be run from Blender's Python Console or Blender's Text Editor.

- Fix normal by auto-smooth
- Removal of duplicate vertices
- Separate the mesh by material
- Hide mesh with glass material (**only before texture baking**)
- Fix normal by manual/eye check

- Fix polygon by manual/eye check

2.2.1. Set auto-smooth

Normals define the orientation of a surface, and incorrect normals can cause a mesh to render incorrectly. Blender's auto-smooth feature can be used to correct the normals of a mesh automatically. After importing all 3d objects, it is necessary to fix the normals using auto-smooth operators.

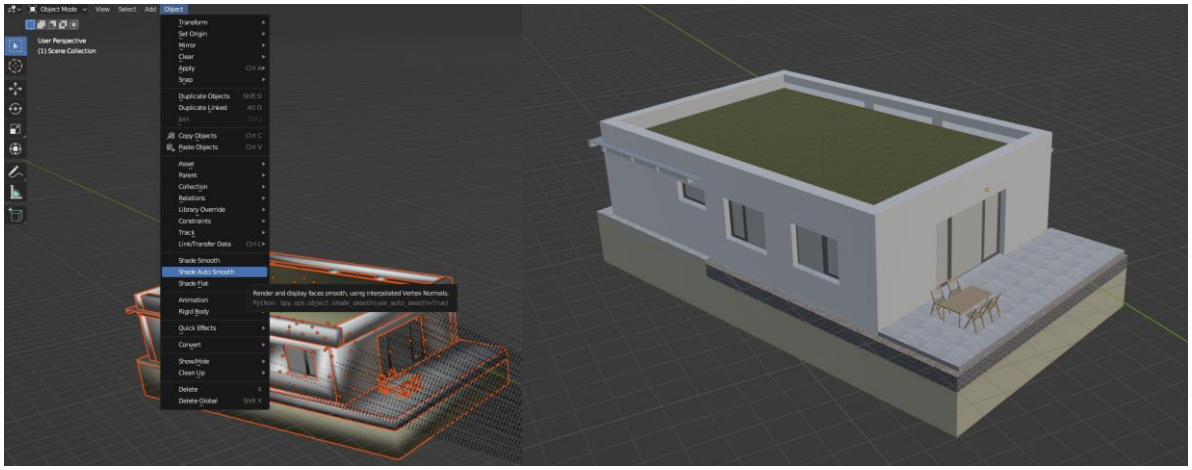


Fig. 3. Fixing normals by auto-smooth feature

2.2.2. Remove doubles and separate mesh by material

Duplicate vertices can cause a range of problems, from rendering issues to inaccurate physics simulations. To remove duplicate vertices, first, go into Edit mode, then select all vertices. Go to Mesh > Clean Up > Merge by Distance. This will remove vertices that are very close together, effectively removing duplicates.

Here is an example of a script for removing duplicate vertices:

```
for ob in bpy.data.objects:
    if ob.type == 'MESH':
        bpy.context.view_layer.objects.active = ob
        bpy.ops.object.mode_set(mode='EDIT')
        bpy.ops.mesh.select_all(action='SELECT')
        bpy.ops.mesh.remove_doubles()
        bpy.ops.object.mode_set(mode='OBJECT')
```

2.2.3. Separate mesh by material

If the mesh uses multiple materials and we want to separate it into different objects based on these materials, we can do so in Edit mode. Select the object and enter Edit mode. Go to Mesh > Separate > By Material. Each material will now be a separate object.

Here is an example of a script for separating objects:

```
for ob in bpy.data.objects:
    if ob.type == 'MESH':
        bpy.context.view_layer.objects.active = ob
        bpy.ops.object.mode_set(mode='EDIT')
        bpy.ops.mesh.select_all(action='SELECT')
        bpy.ops.mesh.separate(type='MATERIAL')
        bpy.ops.object.mode_set(mode='OBJECT')
```

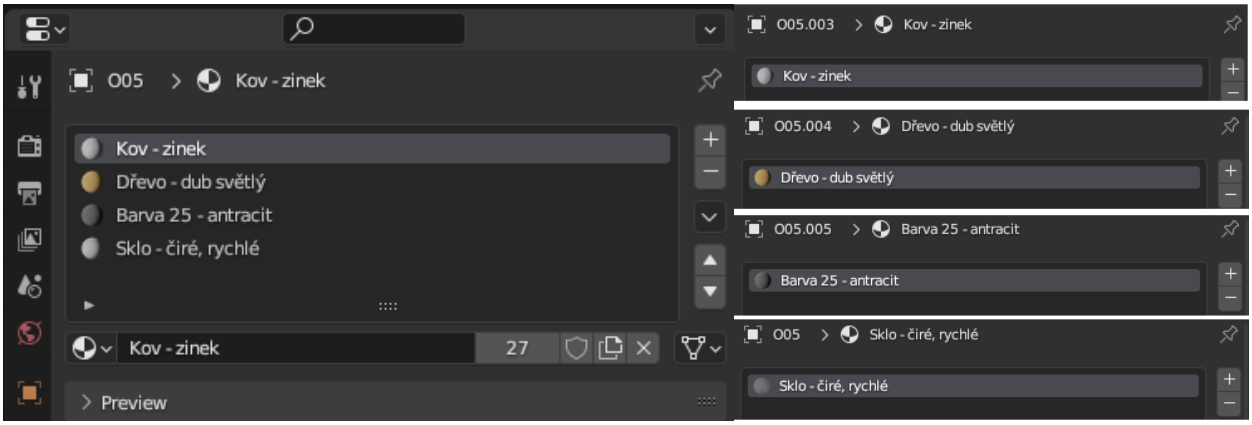


Fig. 4. An example of separating the mesh by material (left-before, right-after)

2.2.4. Hide mesh with glass material

As the baking of materials will be explained later, here we only point out that it is a specific operation that was used to enhance the visual quality of the scene for VR purposes. In Blender, baking transparent materials can cause problems, so we hid all the geometry with the glass material. In the VR scene, it was a part of the doors and windows. Here is an example of a script for hiding objects:

```
for ob in bpy.data.objects:
    if ob.type == 'MESH':
        for mat in ob.material_slots:
            mat_name = mat.name.lower()
            if 'lampa' in mat_name:
                continue
            if 'sklo' in mat_name:
                ob.hide_render = True
                ob.hide_set(True)
```

2.2.5. Manual/eye check

Sometimes, it is necessary to adjust the normals on the geometry manually. To do this, select the object and enter Edit mode. In the viewport, go to Viewport Overlays and enable Face Orientation.

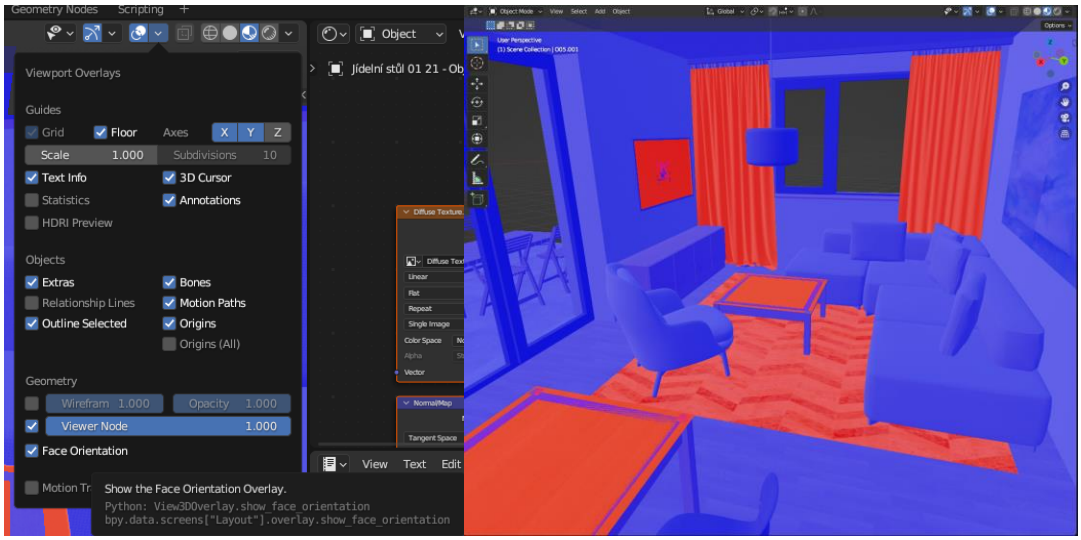


Fig. 5. Switching on the face orientation in the overlays

Fixing polygons manually involves examining the mesh for problems (like non-planar polygons, self-intersections, or flipped normals) and using Blender's mesh editing tools to correct them. Tools like Grab, Rotate, Scale, and Extrude can be used to adjust vertices, edges, and faces. The Knife tool (shortcut K) can be used to add new edges, and Mesh > Clean Up > Make Planar Faces can be used to fix non-planar polygons. The specific operation performed on the mesh depends on the issues we are encountering with the mesh.

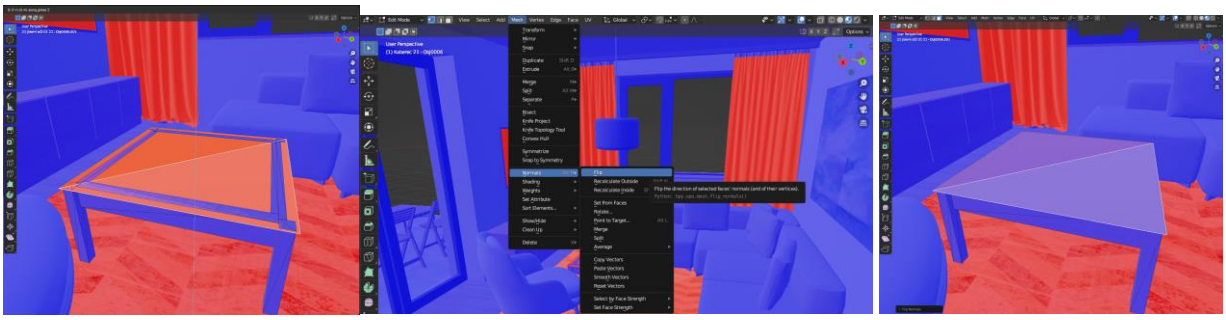


Fig. 6. The example of manually fixing the mesh: select faces of the table, move selected faces up, and fix the orientation of normals

2.3.Add lights to the scene

Lighting is an important part of the scene enhancement that increases the level of realism. We have used two types of lights for lighting: the emissive material and HDR Environment Map.

2.3.1. Emissive material

In Blender's Cycles renderer, an emissive material is a material that emits light. This can be used to create objects that act as light sources, such as light bulbs, neon signs, or objects that glow.

Here's a step-by-step guide on how to create an object with an emissive material in Blender:

- Create or Select an Object: Start by creating a new object or select an existing object to that we want to apply the emissive material. We can do this by right-clicking on the object in the 3D viewport.
- Open the Material Properties Tab: With the object selected, click on the Material Properties tab in the Properties Editor.
- Create a New Material: Click on the 'New' button to create a new material for the selected object.
- Change the Surface Type: In the Surface section of the Material Properties tab, change the Surface type to 'Emission' using the dropdown menu. This changes the material to an emissive material.
- Adjust the Emission Settings: With 'Emission' selected as the surface type, we can now adjust the emission settings. 'Color' changes the colour of the light that the material emits, and 'Strength' changes how bright the light is. For example, we might choose a blue colour and a strength of 10 to create a bright blue light.

2.3.2. Fix the material of the emission object

Here is an example of a script for adjusting the lighting after importing from FBX format:

```
def set_light(name):
    mat = bpy.data.materials[name]
    mat.node_tree.nodes["Principled BSDF"].inputs[19].default_value = (1,1,1,1)
    mat.node_tree.nodes["Principled BSDF"].inputs[20].default_value = 300
    mat.node_tree.nodes["Principled BSDF"].inputs[21].default_value = 1


set_light("GDLM49_Color D05")
set_light("Sklo - lampa")
set_light("GDLM119_builtInMatLampBulb")
```

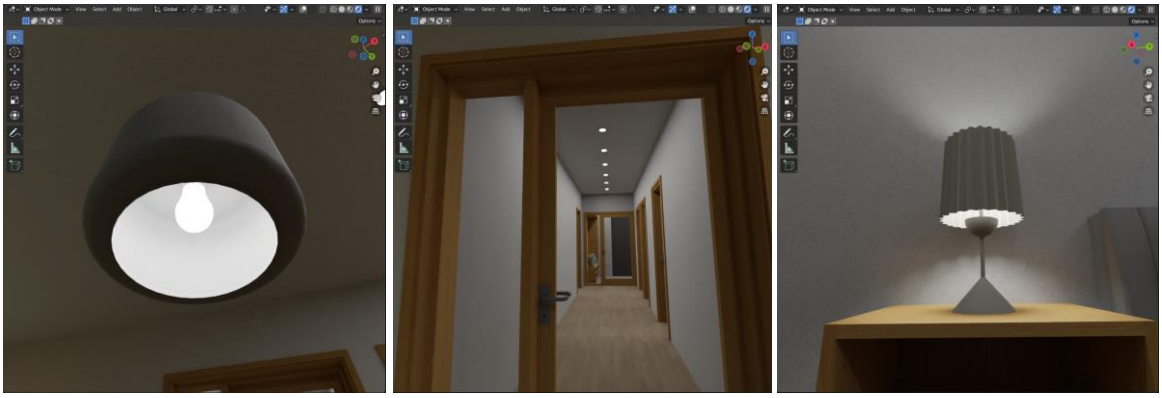


Fig. 7. The examples of emissive materials

2.3.3. HDR Environment Maps in Blender Cycles

An HDR (High Dynamic Range) Environment Map is a panoramic image that encodes a wide range of brightness levels, from deep shadows to bright sunlight. It is used to simulate realistic, dynamic lighting and reflections in a 3D scene. HDR environment maps are often used to create a natural-looking "global illumination" — the indirect light that bounces around a scene. This can greatly enhance the realism of the 3D renders in Blender's Cycles engine.

2.3.4. Understanding HDR Environment Maps

Traditional digital images use 8 bits per colour channel, allowing for 256 different brightness levels. This is often insufficient to capture the full range of brightness we see in the real world, from the dark shadow under a car to the bright sun in the sky.

HDR images, on the other hand, can contain many more brightness levels, potentially up to billions. This allows them to capture real-world lighting conditions much more accurately. When used as environment maps, they can provide detailed, accurate lighting and reflections that can make the 3D renders come to life.

2.3.5. How to Use HDR Environment Maps in Blender Cycles

To use the HDR maps in Blender you should follow these steps:

- Download or Create an HDR Environment Map: There are many sources of HDR environment maps online, both free and paid. We can also create our own using specialized software and a 360-degree camera (e.g. using Blender). Or we can find the few HDR maps in “blender-3.5.1-windows-x64\3.5\datafiles\studiolights\world*.exr”.
- Open the World Settings: In Blender, go to the World Properties tab. This is where we will apply the HDR environment map.
- Use Nodes: Click on the "Use Nodes" checkbox. This allows us to use a node-based setup for the world environment, which is required to use an HDR environment map.
- Add Environment Texture Node: Switch to the Shader Editor and make sure we're editing the 'World' shader network. Add an "Environment Texture" node.
- Open HDR Image: Click 'Open' on the Environment Texture node and select the HDR image file.
- Adjust Strength: We can adjust the strength of the lighting in the Background node. Higher values will result in brighter light.

- Rotate the Environment Map: If we want to rotate the environment map, we can add a 'Mapping' node and a 'Texture Coordinate' node. Connect the 'Generated' output of the Texture Coordinate node to the 'Vector' input of the Mapping node, and then connect the 'Vector' output of the Mapping node to the 'Vector' input of the Environment Texture node. We can then rotate the environment map using the rotation values in the Mapping node.
- The HDR environment map will now light the scene, and any reflective materials will reflect it. The brightness of the HDR image will directly affect the lighting in the scene, so we may need to adjust the material settings to get the look we want.

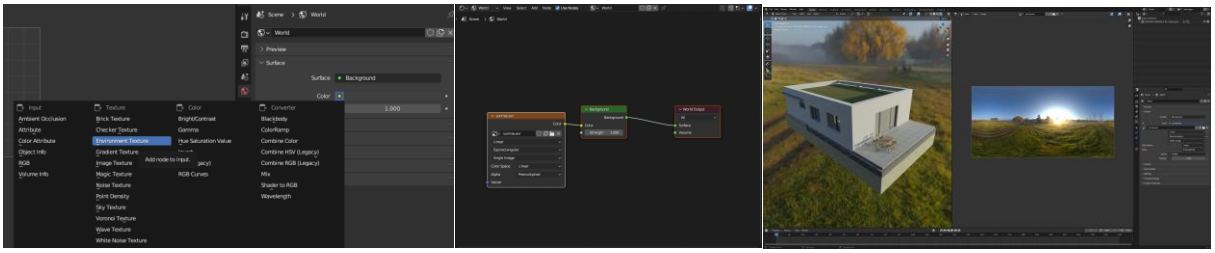


Fig. 8. Add environmental light

2.4.Setting camera

The camera is an important element of the scene, through which the user experience in the VR or rendered visual representation of the scene is provided. For VR exploration, the camera setup is driven by the connected VR headset, more on this is described in the VR section. The camera setup is highly important for the creation of video footage. Multiple approaches are possible in Blender to animate the camera.

Here is a list of selected actions to setup a camera for video footage:

- Add a camera to the scene. Use the add menu in 3D Viewport

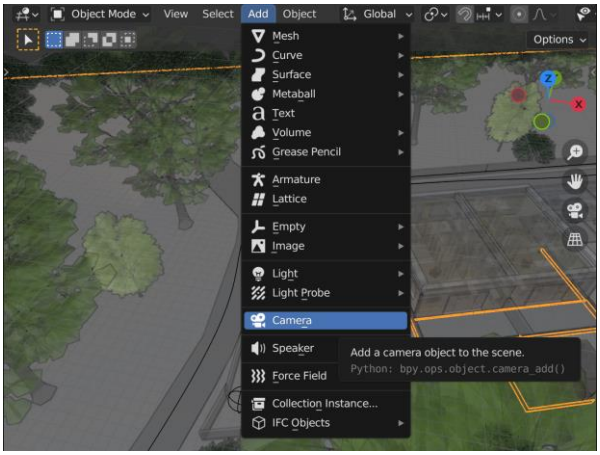


Fig. 9. Adding camera

- Select the camera. Use the Scene Properties of the Properties Editor to set the active camera that will be used for rendering.
- Set rendering resolution and a frame range. Use the Properties Editor and the Output Properties menu.

- Create a camera path. Create a curve using the Add Curve and constrain the camera to follow the curve by the Follow path constrain.

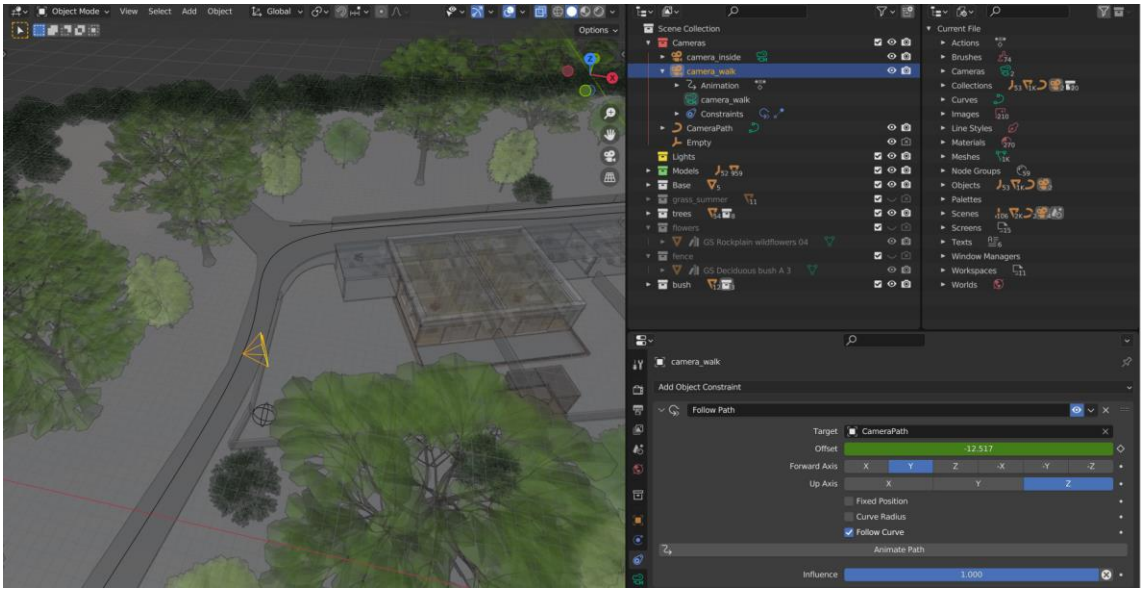


Fig. 10. Using constraints and making the camera follow a specific path

- Use keying in the Timeline Editor to interpolate specific camera movements e.g. rotation between the keyframes during the translation of the camera

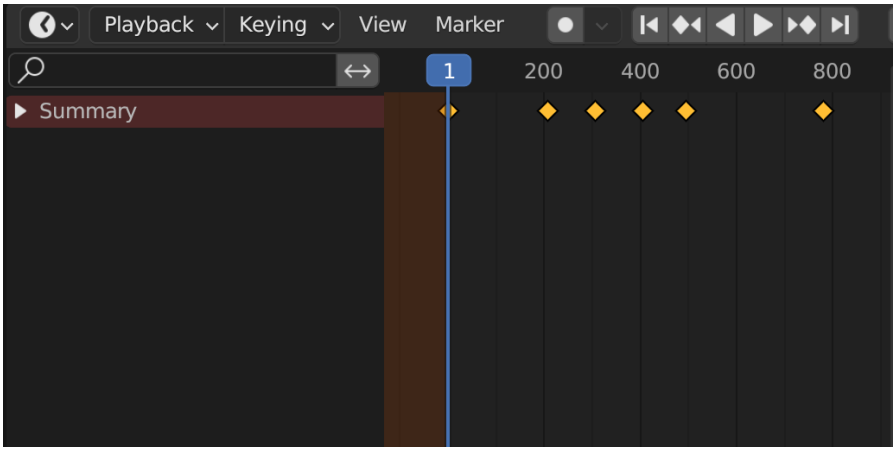


Fig. 10. Use of keyframes in the Timeline Editor

2.5.Enhancing the scene

The exterior of the scene had to be modified to achieve a higher level of realism. This involved populating the scene with vegetation. The original simplified vegetation was replaced by full 3D models of similar type, e.g., grass, bushes, and trees. For this, we have used a parametric tool of Blender called Geometry Nodes (GN). It can be used highly effectively, e.g. for object addition or replacement on the large scale of the scene. The example below shows the use of GN, which takes an old model of the hedgerow and replaces it with a more detailed 3D version of it.

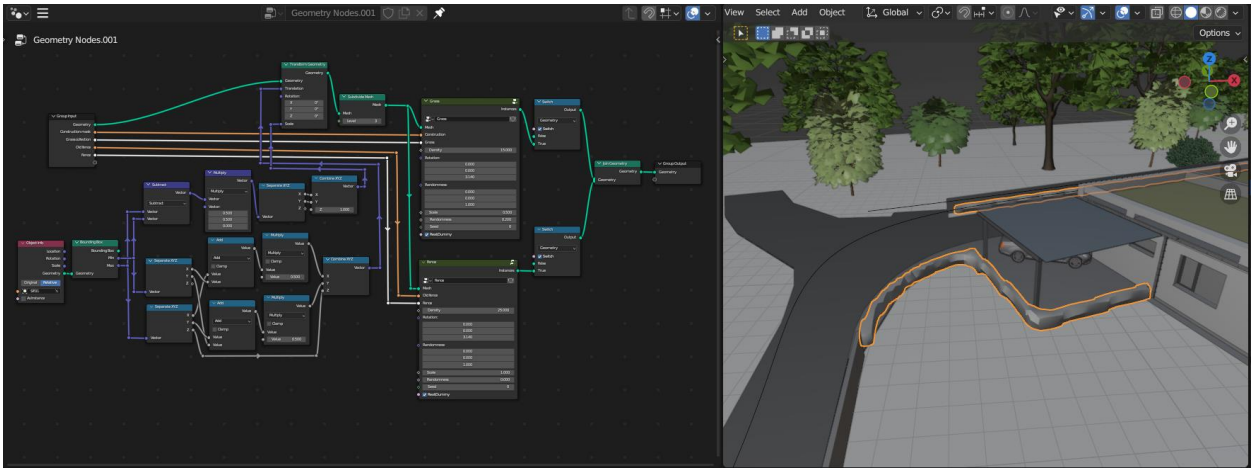


Fig. 11. Application of Geometry Nodes to automatically replace low detail of a hedgerow with its high detail version

2.6.Rendering

To convert a view of the scene into an image, that captures all the requested details such as proper materials, lighting, shadows, etc. a rendering process is applied. One of Blender's renderers called Cycles, provides a powerful path-tracing method. The path-tracing algorithm can achieve photorealistic quality. It simulates the way light behaves in the real world. It's a type of Monte Carlo method used to solve the rendering equation. In simple terms, the rendering equation describes how light interacts with surfaces. For each point in a scene, it considers both the light emitted by the point itself (like a light source) and the light reflected by the point from other light sources.

2.6.1. Rendering using Rendering as a Service (RaaS)

We have used our extension to Blender software, which can provide remote path tracing on an HPC cluster directly from a user's computer equipped with Blender. It is a part of the Rendering as a Service concept and from the user's computer, it is accessible via BHeappe [2] add-on.

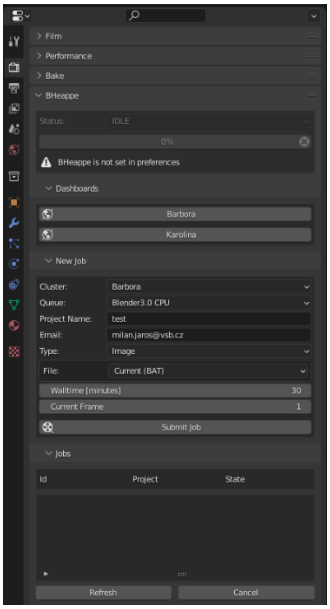


Fig. 12. BHeappe add-on as the interface to RaaS

We have rendered all frames of the architectural visualization using BHeappe and RaaS and then assembled the frames into video footage.

2.6.2. Using path tracing for baking textures

Baking is a powerful feature in Blender that allows one to essentially 'bake' or record complex shading, lighting, and other properties of a material onto a texture. This can dramatically reduce the computational overhead of rendering, especially for VR.

When baking texture using path tracing, these are the main steps performed in Blender:

- **UV Mapping**: Each point on the 3D model is mapped to a corresponding point on a 2D texture map using a process called UV mapping.
- **Ray Casting**: For each point on the 2D texture map, rays are cast into the scene following the path-tracing algorithm. The rays simulate the possible paths that light could take from that point, considering direct lighting, indirect lighting, reflections, refractions, and shadows.
- **Colour Calculation**: The colour at each point on the 2D texture map is calculated based on the accumulated light along the traced rays. This calculation considers the material properties at each point on the 3D model.
- **Texture Storage**: The calculated colours are stored in the texture map, effectively 'baking' the lighting information into the texture.

2.6.3. Preparing Model for Baking

Before baking textures, we need to ensure that the model is ready. This includes:

- UV Mapping: The model needs to have UVs for the texture to be baked onto. UV mapping is a process that 'unwraps' a 3D object onto a 2D space. It determines how a texture wraps around the shape of the 3D object.
- Material Setup: We need to assign a material to the object. This material will contain the node setup that we will bake into a texture.

2.6.4. The Baking Process in Blender

Once the model is properly set up, follow these steps:

- Create a new image texture: In the UV/Image Editor, create a new image to which the bake will be written. We can name it and set the resolution as required.
- Create an Image Texture node: In the Shader Editor, add an Image Texture node to the material and select the new image we created. This node doesn't need to be connected to anything; it just needs to be selected.
- Set the Bake Type: In the Render Properties panel, switch to the Bake section. Here we can choose what we want to bake, like Diffuse, Glossy, Emission, Ambient Occlusion, or a full Render.
- Bake: Hit the 'Bake' button. Depending on the complexity of the scene and the computer's processing power, this might take some time. Once done, we will see the baked texture in the Image Texture node.

We had to perform these steps for each object in the scene.

We had to think about what to consider when baking:

- Resolution: Higher resolution textures will capture more detail but will take longer to bake and will use more memory and slower VR.
- Bake Types: Depending on what we need, we may bake different elements of the material. A full Render bake will include everything, while a Diffuse bake will just include the base colour without any shading. We used “Combined Bake Type” for all objects.
- Cycles Settings: Some Cycles settings can affect the bake. For instance, the number of light bounces in the render settings can affect a full Render or glossy bake. We had to turn off denoising when texture baking, it causes artefacts in some textures.

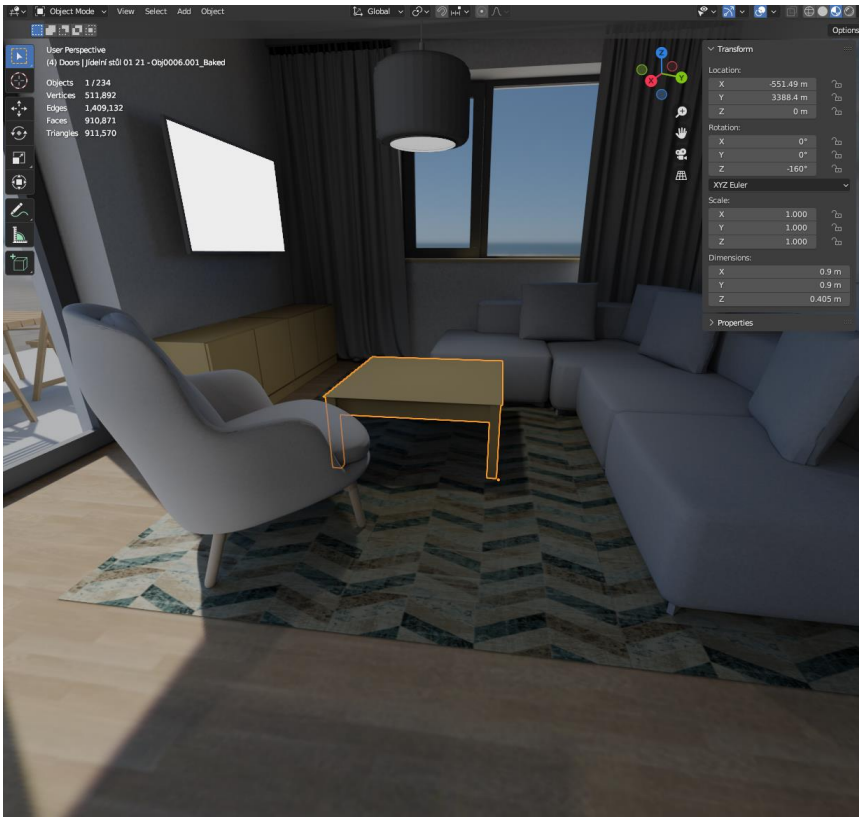


Fig. 13. The example of the final baked scene.

2.7.VR

For architecture and interior design, VR enables clients and designers to walk through the design virtually, giving a better sense of space and aesthetics. In the film and animation industry, directors can pre-visualize scenes and set designs more effectively. In industrial design and engineering, it can aid in the ergonomic evaluation of product designs.

We have chosen Blender to create a VR-ready model and to provide a VR experience. Mainly because Blender has consistently been at the forefront of integrating emerging technologies and with its built-in VR support, Blender has created a more immersive and intuitive way for artists to design, and model.

2.7.1. Preparing the scene for VR

Before the start of using the baked model in VR, it was necessary to perform other actions: repair of transparent materials such as glass, and repair of the object representing the door for which we plan to create close/open actions.

2.7.1.1. Show mesh with glass material

Here is an example of a script for showing objects:

```
for ob in bpy.data.objects:
    if ob.type == 'MESH':
        for mat in ob.material_slots:
            mat_name = mat.name.lower()
            if 'sklo' in mat_name:
                ob.hide_render = False
                ob.hide_set(False)
```

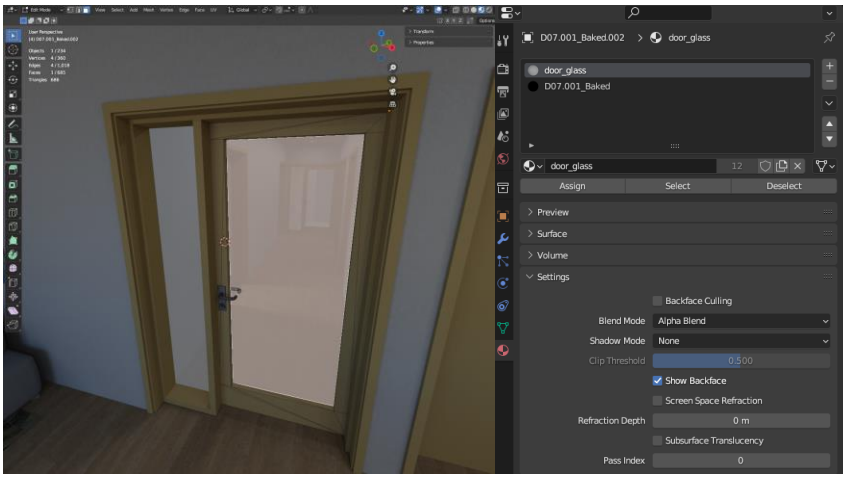
2.7.1.2. Setup blend mode for glass material

Blend Mode is a setting in Blender's Material properties that controls how the alpha channel of the material is used when the object is rendered, specifically in Blender's Eevee and Workbench render engines (Blender's renderers with the real-time capability). This is especially useful for creating effects such as transparency, translucency, or additive blending.

To set up Alpha Blend in Blender's Eevee renderer, follow these steps:

- Select the object and go to the Material Properties tab in the Properties Editor.
- Go to the Settings section at the very bottom of the Material Properties tab. Here, we will find options specific to the Eevee and Workbench render engines.
- Set the Blend Mode to 'Alpha Blend'. This makes the material use the alpha channel of its base colour to create transparency.

For the alpha channel to have an effect, we had to also set the Alpha value of the Base Color in the Principled BSDF shader. We can do this in the Shader Editor by using an RGB node with alpha for the colour input, or by using a texture with an alpha channel.



Fig, 14. Setup of blend material for VR

2.7.1.3. Fix door for VR

We have to fix the door using the "Join selected objects into an active object" operator. It is used for merging two or more objects into a single object. The "active" object is the last one selected, typically highlighted in a brighter colour than the other selected objects.

This operator is especially useful when we want to treat multiple separate meshes as a single unit, for example, for applying transformations or modifiers to all parts of a complex model simultaneously.

Here's how to use the operator:

- Select the Objects: In Object Mode, select the objects that we want to join. We can do this by holding down the Shift key and right-clicking on each object. The order of selection matters here: the last object we select will be the "active" object that all other selected objects are joined into.
- Join the Objects: With the objects selected, we go to the Object menu at the top of the screen and select "Join". All of the selected objects are now merged into the active object.
- After joining, the objects meshes will be combined into a single mesh, but their materials and vertex groups will remain separate. In Edit Mode, we can still edit the vertices, edges, and faces of each part of the joined object independently. If we need to separate the objects again later, we can do so by going into Edit Mode, selecting the part of the mesh we want to separate, and then using the "Separate" operator.

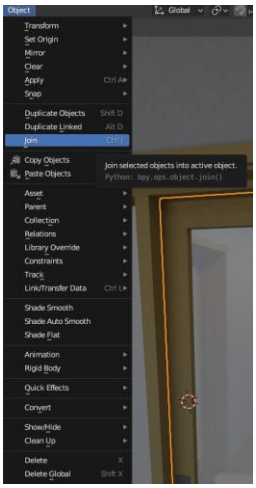


Fig. 15. Connection of individual door objects

For the correct rotation of the door, we also had to set the rotation point using “Origins to 3D Cursor”.

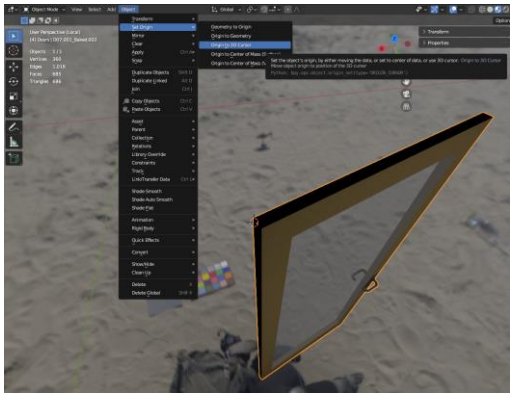


Fig. 16. Fix rotation point

2.8.Introducing the BHolodeck add-on

The BHolodeck add-on [3] is an extension of the VR Scene Inspection add-on. The VR Scene Inspection add-on integrates VR technology into Blender. It enables users to inspect their 3D models in a VR environment directly from Blender without exporting the model to a different software or format. The add-on uses the OpenXR API for rendering VR environments, which makes it compatible with any OpenXR-compliant VR hardware. Users can use the add-on with VR headsets like Oculus Rift, HTC Vive, Windows Mixed Reality, and others.

OpenXR is an open standard for virtual reality (VR) and augmented reality (AR) platforms, also collectively known as extended reality (XR). It was developed by the Khronos Group, an industry consortium dedicated to creating open standards for graphics, VR, and AR. OpenXR aims to unify the VR/AR landscape by providing a common interface between devices and applications. Before OpenXR, developers had to create different versions of their applications for each VR/AR platform, which was time-consuming and

fragmented the market. With OpenXR, a single application can work with any VR/AR device that supports the OpenXR standard.

The BHolodeck add-on functionalities go beyond just visual inspection. It allows users to create an action script that can manipulate objects, adjust lighting, perform measurements, and mark specific areas for review or modification. The user-friendly interface and intuitive controls make the BHolodeck add-on accessible for beginners while maintaining advanced features that cater to experienced designers.

2.8.1. Setting Up the BHolodeck add-on

The setup process for the BHolodeck add-on involves hardware and software requirements, installation, and configuration steps. To begin with, we will need Blender software, a VR headset, and the BHolodeck add-on installed on the system. Ensure the VR hardware is correctly set up and the system meets the necessary hardware requirements.

The setup process is as follows:

- Install the BHolodeck add-on in Blender via the User Preferences panel.
- Enable the add-on and set up the appropriate VR backend depending on the VR hardware.
- Open the scene we wish to inspect in VR.
- Click on the BHolodeck panel in the 3D viewport and find the start button to enable the VR session.
- We can navigate and interact with the 3D scene in the VR session using the VR hardware's controllers. BHolodeck uses the so-called trigger to move in the scene and trigger actions using predefined scripts.

2.8.2. Adding actions to the scene

BHolodeck support action script written in Python. Here is an example of how to create actions for opening and closing doors in the scene. First, we need to set the positions of the closed doors and store these positions in the object property. Next, we need to rotate (open) individual doors and save these positions again in the object property.

2.8.3. Door closing angle adjustment

Here is an example of a script for door-closing angle adjustment:

```
def set_action(obj):
    obj["CLOSE"] = obj.rotation_euler[2]
    obj["STATE"] = "CLOSE"
set_action(bpy.data.objects["D03_Baked.001"])
set_action(bpy.data.objects["D06_Baked.001"])
set_action(bpy.data.objects["D07.001_Baked"])
set_action(bpy.data.objects["D07.002_Baked.001"])
set_action(bpy.data.objects["D07.003_Baked.001"])
```

```
set_action(bpy.data.objects["D07_Baked.001"])
set_action(bpy.data.objects["D08.001_Baked.001"])
set_action(bpy.data.objects["D08_Baked.001"])
set_action(bpy.data.objects["D37.001_Baked.001"])
set_action(bpy.data.objects["D37_Baked.001"])
set_action(bpy.data.objects["D36_Baked.002"])
```

2.8.4. Adjusting the angle for open doors after manually turning each door

Here is an example script for setting the door opening angle:

```
def set_action(obj):
    obj["OPEN"] = obj.rotation_euler[2]
    obj["STATE"] = "OPEN"
set_action(bpy.data.objects["D03_Baked.001"])
set_action(bpy.data.objects["D06_Baked.001"])
set_action(bpy.data.objects["D07.001_Baked"])
set_action(bpy.data.objects["D07.002_Baked.001"])
set_action(bpy.data.objects["D07.003_Baked.001"])
set_action(bpy.data.objects["D07_Baked.001"])
set_action(bpy.data.objects["D08.001_Baked.001"])
set_action(bpy.data.objects["D08_Baked.001"])
set_action(bpy.data.objects["D37.001_Baked.001"])
set_action(bpy.data.objects["D37_Baked.001"])
set_action(bpy.data.objects["D36_Baked.002"])
```

2.8.5. Setting action script for each door

Here is an example script for opening and closing doors using VR controllers. This script is assigned to each object using the VRObjectAction system, which is part of the BHolodeck add-on.

```
import bpy
obj = bpy.context.selected_objects[0]
if obj["STATE"] == "OPEN":
    obj.rotation_euler[2] = obj["CLOSE"]
```

```
    obj["STATE"] = "CLOSE"
else:
    obj.rotation_euler[2] = obj["OPEN"]
    obj["STATE"] = "OPEN"
```

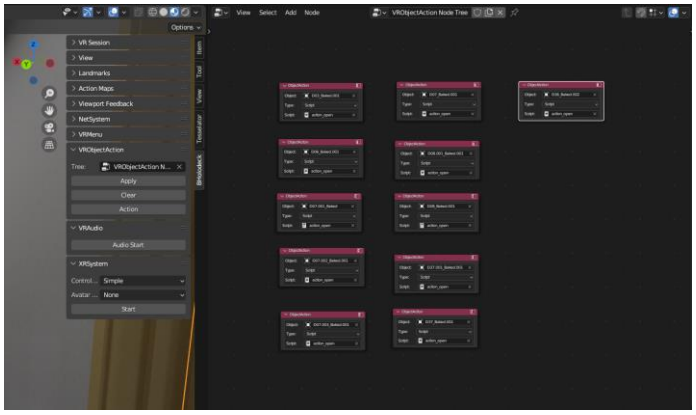


Fig. 17. The example of the VRObjectAction system

2.8.6. Using the BHolodeck add-on

The BHolodeck add-on provides a variety of features that allow for direct interaction with the 3D scene. Some of the key features include:

- Inspect Mode: Allows users to move around the scene and closely inspect objects.
- Manipulate Mode: Users can directly select, move, scale, or rotate objects using action scripts.
- VR Camera: Users can adjust the camera's position and settings while in the scene.

Click the 'Start' button in the BHolodeck tab to start the VR session. Put on the VR headset, and we should be able to look around the scene in 3D. Use the one VR controller to navigate and interact with the scene.

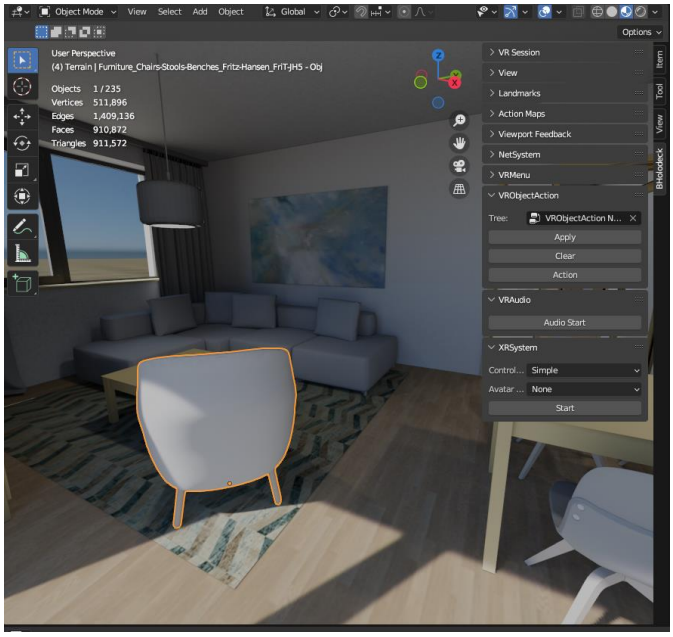


Fig. 18. Start button for starting of VR session

3. Conclusion

We have designed and tested processes to provide a high-fidelity architectural visualization in the form of video footage and as an interactive scene exploration in virtual reality (VR). We have used the 3D modelling software Blender to perform both tasks. An architectural model of a set of family houses has been provided as a use case. Our results have proven the potential of Blender as a comprehensive architectural visualization tool that offers the possibility of high-quality photorealistic rendering and VR-ready experience.

References

[1] Blender. https://www.blender.org

[2] BHeappe. https://code.it4i.cz/raas/bheappe

[3] BHolodeck. https://code.it4i.cz/blender/bholodeck

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 101101903. The JU receives support from the Digital Europe Programme and Germany, Bulgaria, Austria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, the Netherlands, Belgium, Luxembourg, Slovakia, Norway, Turkey, Republic of North Macedonia, Iceland, Montenegro, and Serbia. This project has received funding from the Ministry of Education, Youth and Sports of the Czech Republic.