Handover

DRIVER SIMULATOR HANDOVER

Thrill Capital – 2025 Semester 1

Licheng Wang

Company Introduction & Background

The Hamilton CBD Driving Simulator project is a Unity-based initiative aimed at creating a highly realistic urban driving simulation. The project is a collaborative effort that supports the organization's mission to innovate in digital environments and deliver cutting-edge tools for education, research, and urban development. The primary focus is on simulating the Hamilton Central Business District (CBD) in New Zealand, with an emphasis on visual accuracy and realistic traffic dynamics.

Business Context:

The organization is focused on leveraging technology to provide realistic urban simulation environments. The driving simulator directly addresses the need for a robust tool to:

- Facilitate education and training in urban navigation and driving.
- Serve as a testing ground for traffic AI systems and autonomous vehicle technologies.
- Offer engaging, research-driven solutions for urban planning and traffic management studies.

Business Objectives:

1. **Improve Realism:** Create detailed, accurate 3D models of the Hamilton CBD to provide an authentic driving experience.
2. **Enhance Traffic AI:** Develop dynamic and responsive traffic systems that replicate real-world driving scenarios.
3. **Ensure Scalability:** Design the simulator to allow easy expansion to include additional cities, features like pedestrian AI, and compatibility with emerging technologies such as VR.

The Hamilton CBD Driving Simulator is a vital step toward establishing the organization as a leader in digital urban simulations, offering significant value to education, research institutions, and technology developers. This project aligns with the organization's goals of innovation, scalability, and delivering high-impact solutions.

Prerequisites

This section provides a step-by-step guide on what is needed to work on the project files and ensures a smooth handover to anyone taking over the project.

1. Software Requirements

To ensure smooth development and collaboration, the following software versions and tools are required:

Unity

- **Version**: Unity **2022.3.7f1** (LTS version recommended for stability).
- **Why Use Unity?**
  - Provides a robust engine for 3D environments.
  - Offers built-in physics for realistic vehicle simulation.
  - Supports cross-platform development.
- **Installation**:
  - Download Unity Hub from Unity's official website.
  - Install Unity **2022.3.7f1** via Unity Hub:
    - Open Unity Hub.
    - Go to the **Installs** tab.
    - Select **Add Editor** and search for Unity **2022.3.7f1**.

Supporting Tools and Plugins

- **EasyRoads3D V3**:
  - This plugin is used for creating and editing road networks in Unity.
  - Ensure you have access to the same version used in the project. Licensing or installation might be required.
- **Road Marker**:

    - Used for road markings.
    - The plugin should already be included in the project, but confirm compatibility with your setup.
- **Texturing Tools**:
    - Tools like GIMP or Photoshop (I was using Photoshop)
- **Mobile Traffic System v1.3.11 by Gley**:
    - Integrated to simulate realistic vehicle behaviours and traffic flow, including waypoint-based AI navigation.

Blender

- **Version**: Blender **4.4**.
- **Installation**:
    - Download Blender from Blender's official website.
    - Install it with default settings for easy setup.

2. Setting Up the Workspace

Getting the Files

- Obtain the complete project folder from the shared link:
    - **Link**: Access the project files here.
    - **You need to get access to the files from Chan. Ask him to email you the link to get access to the files.**
- Ensure the folder contains:
    - Unity project files (.unity, Assets, Library, etc.)
    - Blender files (.blend)
    - External assets (e.g., textures, FBX models, plugin files).

Configuring Unity

- **Open the Project**:
    1. Open Unity Hub.
    2. Click **Open Project**.
    3. Navigate to the project folder and select it.
- **Resolve Missing Dependencies**:

- If prompted to download missing packages or plugins, accept and install them.
- Use the Unity Console to identify any errors related to missing scripts or assets.

Configuring Blender

- Open Blender.
- Load any .blend files included in the project.
- Ensure the BLOSM add-on is installed and active.

3. Essential Knowledge for Taking Over

This project assumes basic familiarity with Unity and Blender. However, if you're new to these tools:

- **Unity Basics**:
    - Learn about **Scenes**, **Game Objects**, **Components**, and **Prefabs**.
    - Understand **Navigation**, **Inspector Panel**, and **Hierarchy**.
- **Blender Basics**:
    - Understand **Object Mode** and **Edit Mode**.
    - Know how to import/export models (FBX for Unity).

Suggested Learning Resources:

- Unity Learn: https://learn.unity.com/
- Blender Tutorials: https://www.blender.org/support/tutorials/

Project Overview: A Beginner-Friendly Explanation

The **Driving Simulator Project** is a Unity-based application designed to simulate driving experiences in a detailed urban environment. The project integrates tools like EasyRoads3D, Road Marker, and Mobile Traffic System to create a realistic and interactive environment for users. Here's a breakdown of the project:

1. Purpose of the Project

- **Main Goal**: To create a functional and visually accurate driving simulator of Hamilton CBD, New Zealand.

- **Key Features**:
    - Realistic road networks and traffic behaviour.
    - Accurate building facades based on the real-world cityscape.
    - Customizable vehicle AI, pedestrian paths, and road markings.
- **Target Users**: Designed for education, research, or entertainment purposes, this project is adaptable for future enhancements like advanced traffic systems or VR integration.

2. Key Components

The project comprises several interconnected systems:

Road Network

- Created using **EasyRoads3D**, the road network mimics the layout of Hamilton CBD.
- Features include:
    - Smooth road curves and intersections.
    - Dynamic road types, such as highways, residential streets, and intersections.
    - Integration of sidewalks and pathways for pedestrians.

Traffic System

- Powered by **Mobile Traffic System**, the traffic AI handles:
    - Vehicle behaviour, such as lane following, acceleration, and deceleration.
    - Traffic lights and priority rules at intersections.
    - Configurable traffic density for various simulation scenarios.

Building Models

- The urban environment is built with:
    - Textures and models created or enhanced in **Blender** using **BLOSM** for importing OpenStreetMap data.
    - Low-poly models optimized for performance, with high-resolution textures applied to maintain realism.

Road Markings

- Markings created using **Road Marker** provide essential road details:

    - Lane dividers, crosswalks, and other standard road markings.
    - Custom patterns and configurations to match the Hamilton road style.

3. Workflow

The project's workflow combines 3D modelling, Unity development, and plugin integration:

- **Initial Environment Setup**:
    - Used **BLOSM** to import the Hamilton CBD map into Blender for initial modelling.
    - Refined road layouts and buildings in Blender.
- **Unity Integration**:
    - Imported Blender models into Unity.
    - Configured road networks, traffic systems, and road markings.
- **Testing and Iteration**:
    - Played and tested in Unity Editor to refine traffic behaviour and overall performance.
- **Texture creation in Photoshop or GIMP**
    - Single texture with multiple perspectives from target building is recommended as it easy to files' manage and UV mapping
    - Eliminate perspective for photos in Photoshop is more ideal in effect than in blender
- **Model in Blender and export with FBX**
    - FBX without embedded textures, together with textures contained inside `Model.fbm\` sub directory at the same directory is valid way to be utilized in unity

4. Current State of the Project

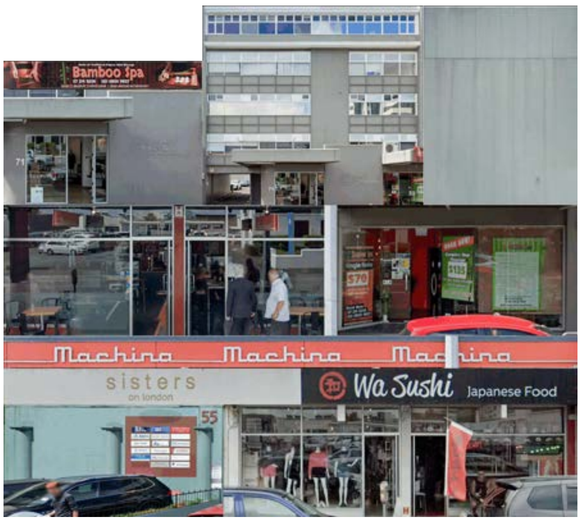
- The project is functional and includes the following:
    - Mostly working Driving AI and driving routes.
    - Upgraded textures in the CBD with less quality as you go out.
    - Some part of area is missing maps and building decoration.
- **Known Limitations**:

- Certain areas may lack details.
- Performance optimizations are still required for larger map areas.

5. Future Development Potential

- **Additional Features**:
    - Add pedestrian AI and detailed animations for a richer simulation.
    - Expand the map to include more areas of Hamilton or additional cities.
    - Expand the road network and driver AI.
- **Optimization Goals**:
    - Improve FPS by further optimizing models and scripts.
    - Use baked lighting and simplified shaders for better performance on low-end devices.
- **Enhancements**:
    - Introduce advanced traffic AI, including traffic jams and emergency vehicles.
    - Implement weather effects like rain or fog for a dynamic simulation.

Generate buildings Texture assets



Bamboo Spa Hamilton

Expected Texture be like

- Orthogonal perspective
- Packed into single Texture if feasible
- Lightened shadow if it looks too dark

Recommend Setting

- Recommend Resolution 4096*4096
- Well-organized

💡 **Why resolution of 4096*4096 is recommend**

1. **High Detail for Large Surfaces**

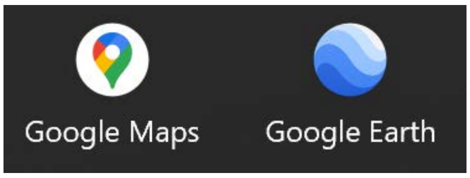
   Allows for crisp, detailed visuals without blurring or visible pixelation.

2. **GPU Optimization at better performance against size**

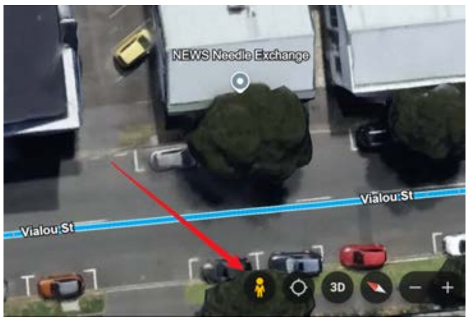
   GPUs are optimized to handle **power-of-two (POT)** texture sizes — dimensions like 256, 512, 1024, 2048, 4096, etc.

1. Capture pictures using Street View in Google Map or Google Earth

- Select from Google Maps or Google Earth or both (Google Maps for copy building names, Google Earth for capturing screenshots)

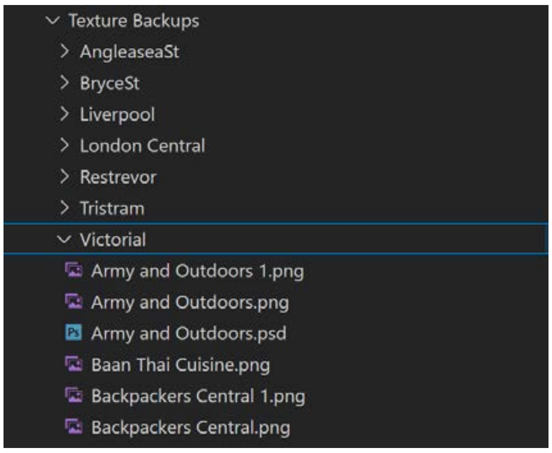


- Drag the icon into place where the viewer supposed to be



- Capture from target buildings at different perspective - to have multiple shots for a single building
  - Facing to the front as possible as you could

- Save these raw photos into directories for better organized File Structure



```
⌄ Texture Backups
  › AngleaseaSt
  › BryceSt
  › Liverpool
  › London Central
  › Restrevor
  › Tristram
  ⌄ Victorial
      Army and Outdoors 1.png
      Army and Outdoors.png
      Army and Outdoors.psd
      Baan Thai Cuisine.png
      Backpackers Central 1.png
      Backpackers Central.png
```

File Structure for Textures backup

- Divided by Location and Street name while saved into separate directory
- Images named by building's with sequential suffixes

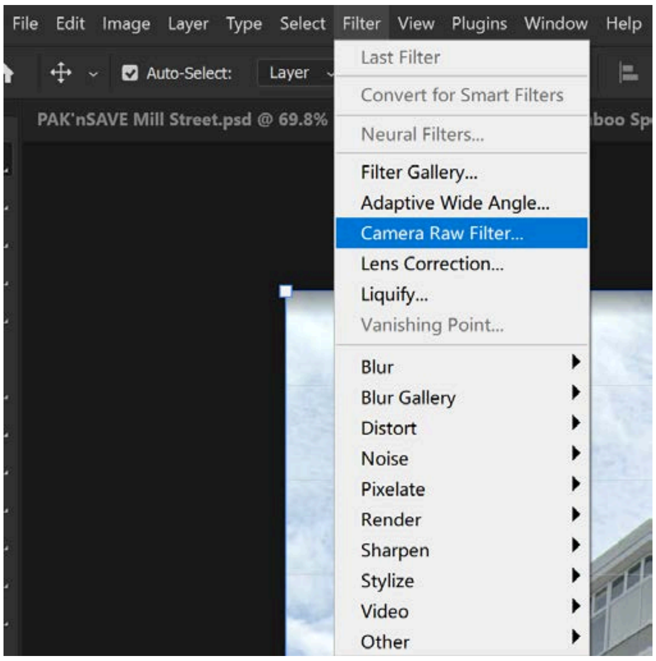
💡

Advantages and shortages to Google Map and Google Earth

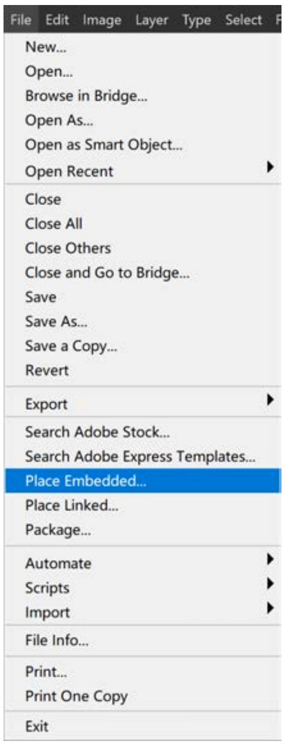
- Google maps is good at performance and convenience since its web driven
- Google Earth is relevantly heavy but benefit from its clean layout. No labels and signs appeared at Street view layout

2. Generate Packed Textures in Photoshop

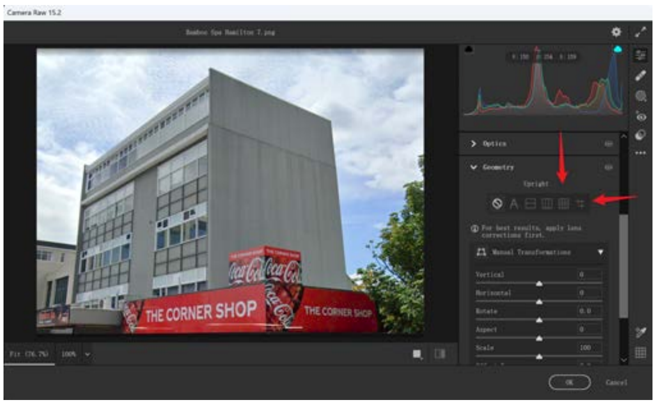
1. Place images into Photoshop using Dragging or Place Embedded
2. Adjust Geometry in Camera Raw, in order to be faced to layer. Try automatic first, otherwise by **Guided lines**
3. Adjust exposure shadows and other options for fine lightening: Shadows not being too dark since shadows are being calculated by software
4. Try to organize multiple images in default position. As the First floor placed at Top, Ground floor placed at Bottom, while banner placed at middle, so that teammates are able to recognize how textures are mapped to models (in later UV mapping)
5. Save project as `PSD|PSB` before export the image to `PNG` . Backup for futures potential adjustment



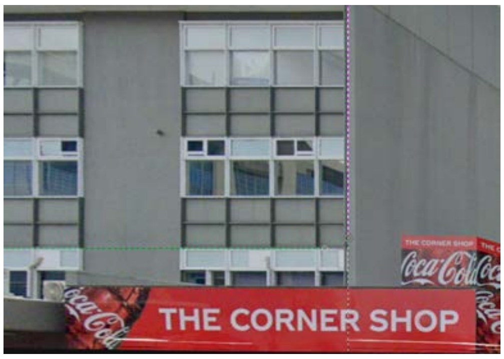
Enter Camera Raw



Place images into Photoshop



Adjust to upright image



Use guided lines

💡

For Best output, leave certain pixels between images

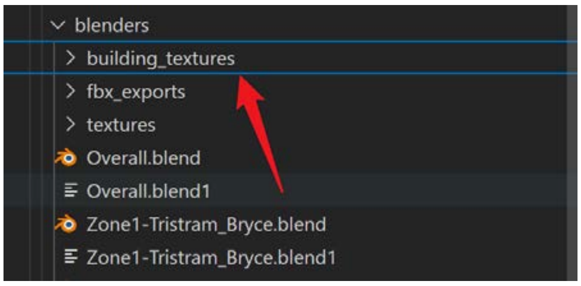
As to avoid bleeding in UV mapping

No interval between images may result in difficulty as marquee select UV, certain pixels may included where it not supposed to be

3. Finalize Packed Textures: Locate in directory and Downsizing

Texture Location is vital in UV mapping, if textures moved to other path or renamed, blender will failed to find image source (Images are not packed to blender project)

Relative path from blender



```
blenders
  building_textures
  fbx_exports
  textures
  Overall.blend
  Overall.blend1
  Zone1-Tristram_Bryce.blend
  Zone1-Tristram_Bryce.blend1
```

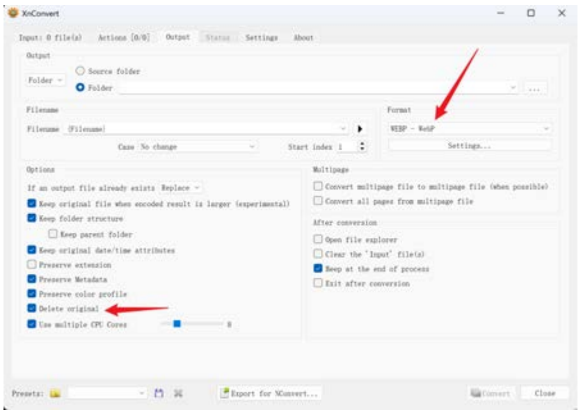
> **Textures locate at**
>
> `building_textures\`

- Do NOT rename or move textures after reference to project
- If rename is necessary (Like format changes), re-import image to Materials or modify references by Python scripts (Batch modification)

Downsize textures for better performance in blender

A 4K texture in 32-bit RGBA format takes up **about 67 MB** in VRAM *per texture* (uncompressed)

- By default, images exported from Photoshop as `PNG` or `JPG` which takes about 10 MiB for 4096×4096 resolution
- Use “XnConvert” to convert image to `WEBP` which persist image quality as well as considerable downsize (at usual 95% size decreased)
- WEBP DOESNOT support in Unity yet, but there are addons to manage it

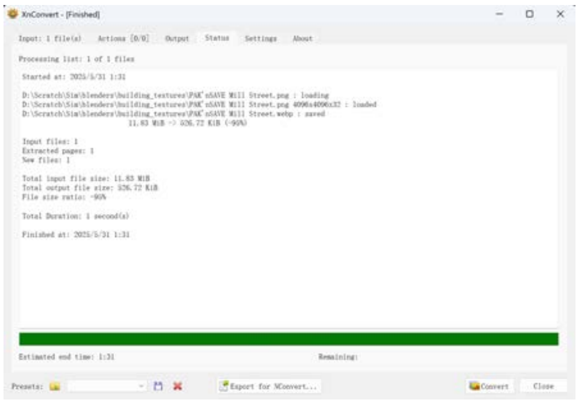


Output: Folder; ○ Source folder; ● Folder
Filename: {Filename}; Case: No change; Start index: 1
Format: WEBP - WebP; Settings...

Options
If an output file already exists: Replace
☑ Keep original file when encoded result is larger (experimental)
☑ Keep folder structure
☐ Keep parent folder
☑ Keep original date/time attributes
☐ Preserve extension
☑ Preserve Metadata
☑ Preserve color profile
☑ Delete original
☑ Use multiple CPU Cores 8

Multipage
☐ Convert multipage file to multipage file (when possible)
☐ Convert all pages from multipage file

After conversion
☐ Open file explorer
☐ Clear the 'Input' file(s)
☑ Beep at the end of process
☐ Exit after conversion

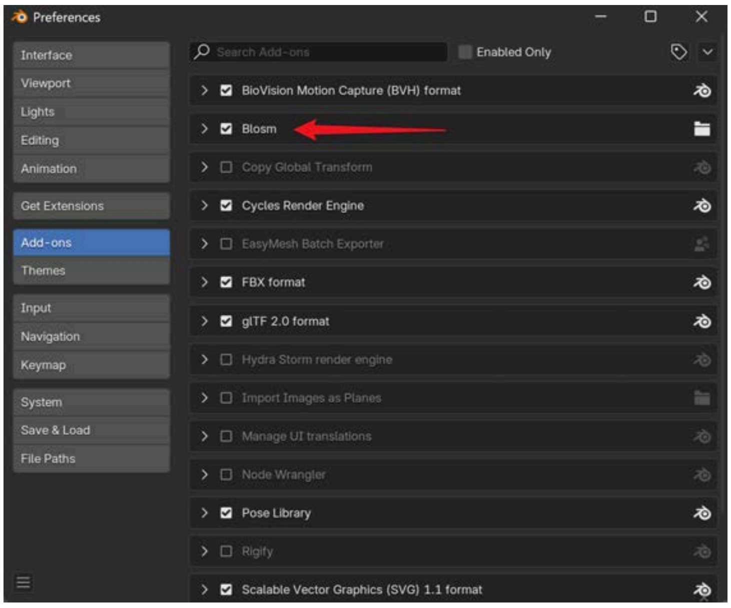


```
Processing list: 1 of 1 files

Started at: 2025/5/31 1:31

D:\Scratch\Sim\blenders\building_textures\PAK'nSAVE Mill Street.png : loading
D:\Scratch\Sim\blenders\building_textures\PAK'nSAVE Mill Street.png 4096x4096x32 : loaded
D:\Scratch\Sim\blenders\building_textures\PAK'nSAVE Mill Street.webp : saved
                        11.83 MiB -> 526.72 KiB (-96%)

Input files: 1
Extracted pages: 1
New files: 1

Total input file size: 11.83 MiB
Total output file size: 526.72 KiB
File size ratio: -96%

Total Duration: 1 second(s)

Finished at: 2025/5/31 1:31

Estimated end time: 1:31                    Remaining:
```

Modeling and UV mapping

1. Use BLOSM to import Google Tiles (buildings in area)

- Download BLOSM addons as ZIP file
- Import into blender and activate it
- Press N to open BLOSM options → Select position tuple from opening website → Paste to UI → Check options as picture then wait importing completion
- Unpack images to files, which create links to images files outsides blender project, hence reduce blender's memory size



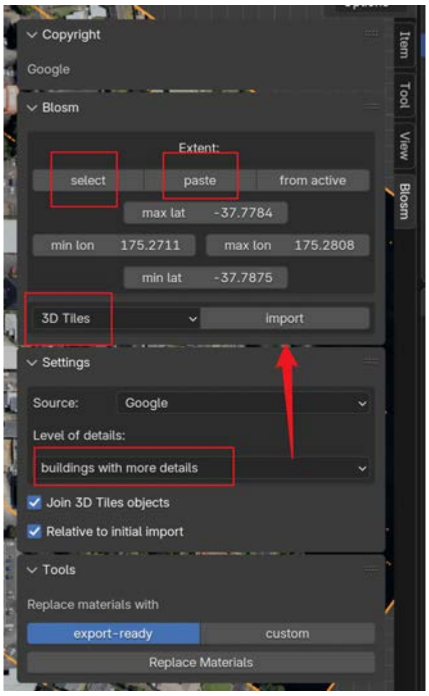
Preferences

Interface, Viewport, Lights, Editing, Animation, Get Extensions, Add-ons, Themes, Input, Navigation, Keymap, System, Save & Load, File Paths

Search Add-ons — Enabled Only

- [✓] BioVision Motion Capture (BVH) format
- [✓] Blosm
- [ ] Copy Global Transform
- [✓] Cycles Render Engine
- [ ] EasyMesh Batch Exporter
- [✓] FBX format
- [✓] glTF 2.0 format
- [ ] Hydra Storm render engine
- [ ] Import Images as Planes
- [ ] Manage UI translations
- [ ] Node Wrangler
- [✓] Pose Library
- [ ] Rigify
- [✓] Scalable Vector Graphics (SVG) 1.1 format

Activate BLOSM



Copyright: Google

Blosm

Extent: select | paste | from active

max lat -37.7784

min lon 175.2711 — max lon 175.2808

min lat -37.7875

3D Tiles — import

Settings

Source: Google

Level of details: buildings with more details

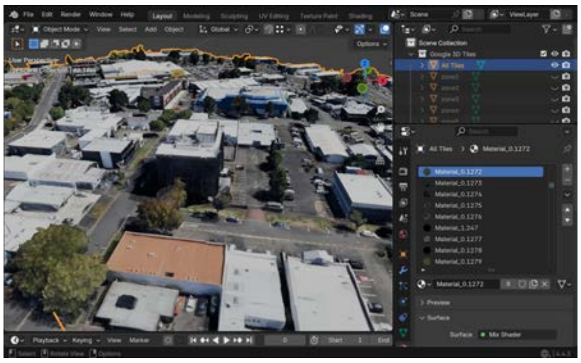
- [✓] Join 3D Tiles objects
- [✓] Relative to initial import

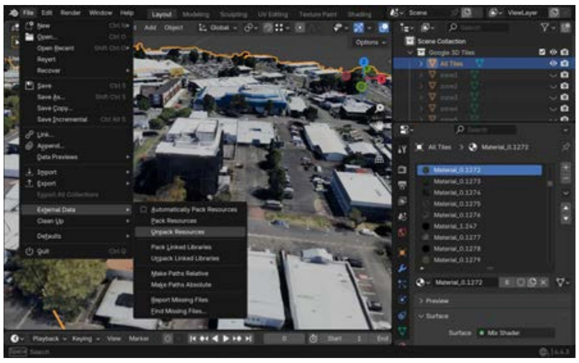
Tools

Replace materials with: export-ready | custom

Replace Materials

Import Google Tiles





💡

Benefits of Unpacking Images from Blender

1. Better Asset Reusability

- **External images can be reused** across multiple scenes, Blender projects, or even game engines like Unity or Godot.
- No need to duplicate textures by copying `.blend` files.

---

2. Smaller `.blend` File Size

- Packed textures increase `.blend` file size significantly.
- Unpacking reduces size, making the file easier to share or version-control (e.g. on GitHub).

---

3. Improved Game Engine Compatibility

- Game engines (Unity, Godot, Unreal) cannot read textures packed into `.blend` files.
- Unpacked images are required for importing models and materials into engines.

2. Simplify Shading of Google 3D Tiles

By default, imported Google 3D use Mix shader which eventually lead to export failure on their texture (no matter textures embedded or referred to exclusive directory)

Use Python scripts to simplify their shader as Principle BDSF directly output at beginning of importation

Python: transform all materials to Principle BDSF

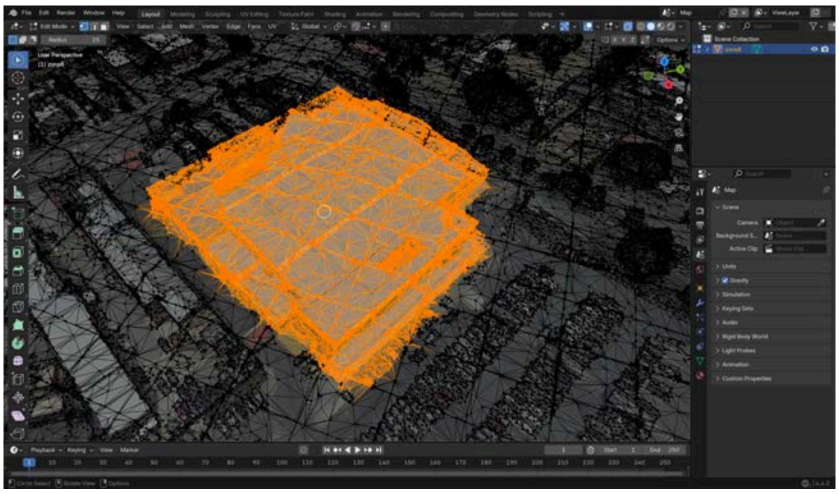
```
import bpy

processed_objects = 0
skipped_objects = 0
processed_materials = set()


```

```
# Iterate through all objects
for obj in bpy.data.objects:
    if obj.type == 'MESH' and obj.material_slots:
        for slot in obj.material_slots:
            material = slot.material
            if material and material.use_nodes:
                # Skip already processed materials
                if material.name in processed_materials:
                    continue

                # Check if the material already meets the condition (TEX_IMAGE → Bas
                already_valid = False
                for node in material.node_tree.nodes:
                    if node.type == 'BSDF_PRINCIPLED':
                        base_color_input = node.inputs['Base Color']
                        if base_color_input.is_linked:
                            from_node = base_color_input.links[0].from_node
                            if from_node.type == 'TEX_IMAGE':
                                already_valid = True
                                break

                if already_valid:
                    skipped_objects += 1
                    print(f"Skipped object: {obj.name}, material: {material.name} (already
                    continue

                # Create a new material
                new_material = bpy.data.materials.new(name=f"New_{material.name}")
                new_material.use_nodes = True
                new_nodes = new_material.node_tree.nodes
                new_links = new_material.node_tree.links

                # Clear default nodes
                for node in new_nodes:
                    new_nodes.remove(node)

                # Create new nodes
                bsdf = new_nodes.new(type='ShaderNodeBsdfPrincipled')
                bsdf.location = (0, 0)
                output = new_nodes.new(type='ShaderNodeOutputMaterial')
                output.location = (300, 0)
                new_links.new(bsdf.outputs['BSDF'], output.inputs['Surface'])

                # Search for image texture
                for node in material.node_tree.nodes:
                    if node.type == 'TEX_IMAGE' and node.image:
                        tex = new_nodes.new('ShaderNodeTexImage')
                        tex.image = node.image
                        tex.location = (-300, 0)
                        new_links.new(tex.outputs['Color'], bsdf.inputs['Base Color'])
                        break

                slot.material = new_material
                processed_materials.add(material.name)
                processed_objects += 1
                print(f"Processed object: {obj.name}, material: {material.name} → {new

print(f"\n✅ Done. Total processed objects: {processed_objects}, skipped: {skipp
bpy.ops.wm.save_mainfile()
```

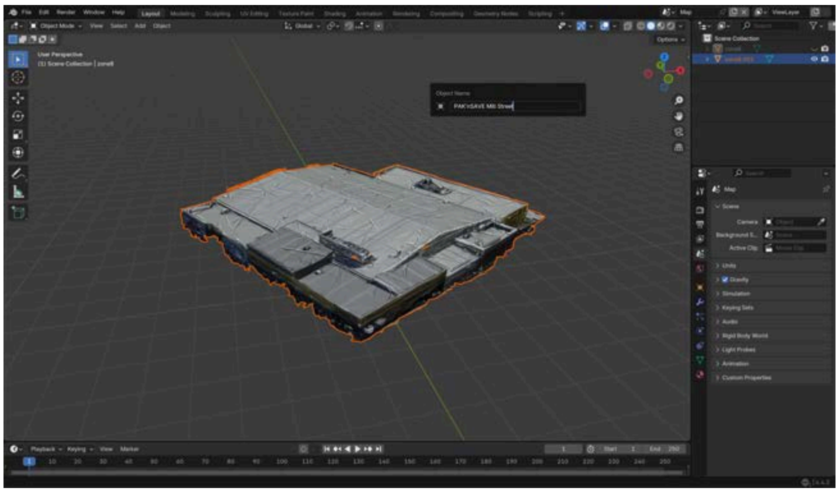
3. Modeling Walls and Crucial components in Blender

1. Separate buildings from 3D Tiles as individual objects

Toggle X-Ray in edit mode, select vertices before separating

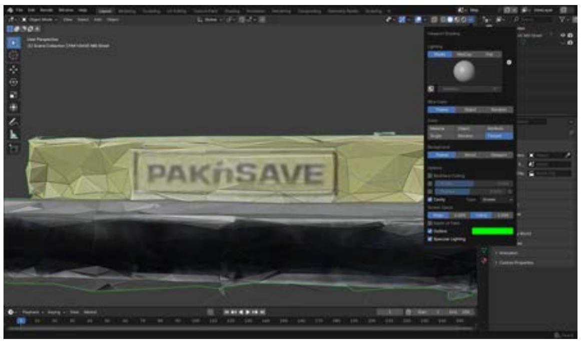


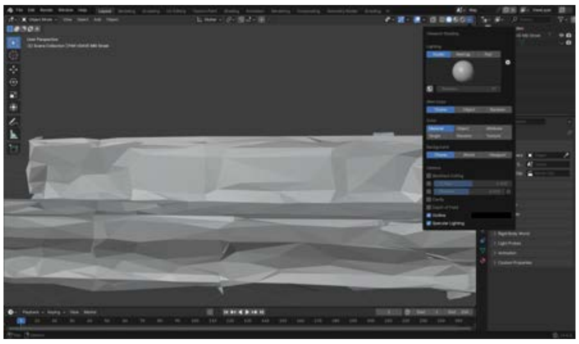
2. Rename objects


Object Name
PAK'nSAVE Mill Street

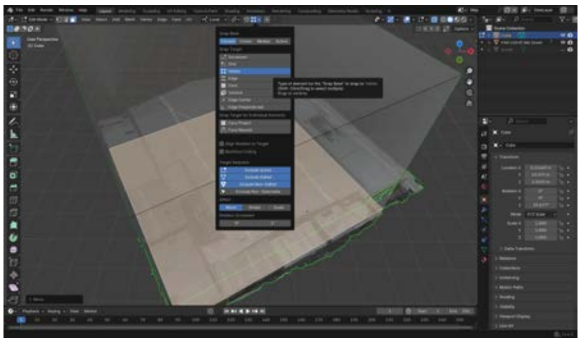
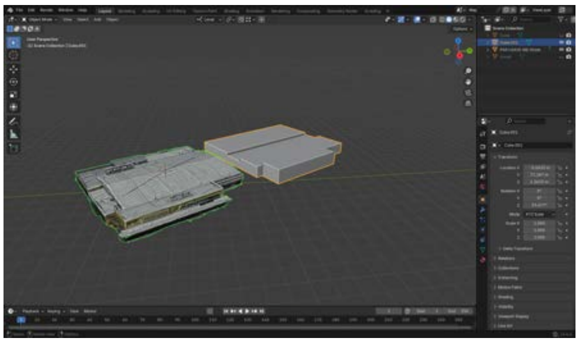
3. Improve Viewport shading in object mode

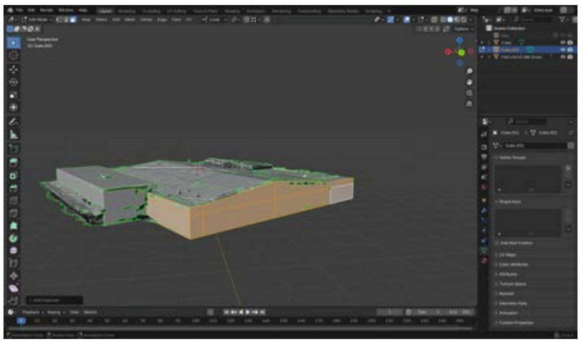
Better for observing textures in model

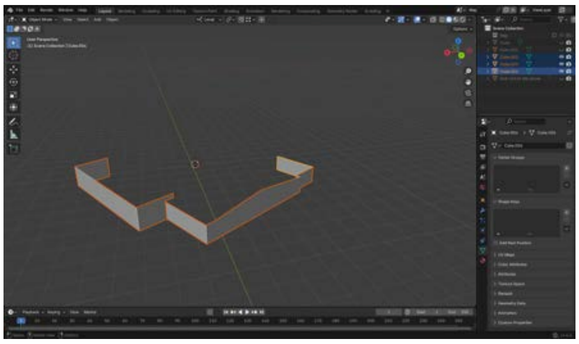




4. Create walls as it necessarily observed from street view



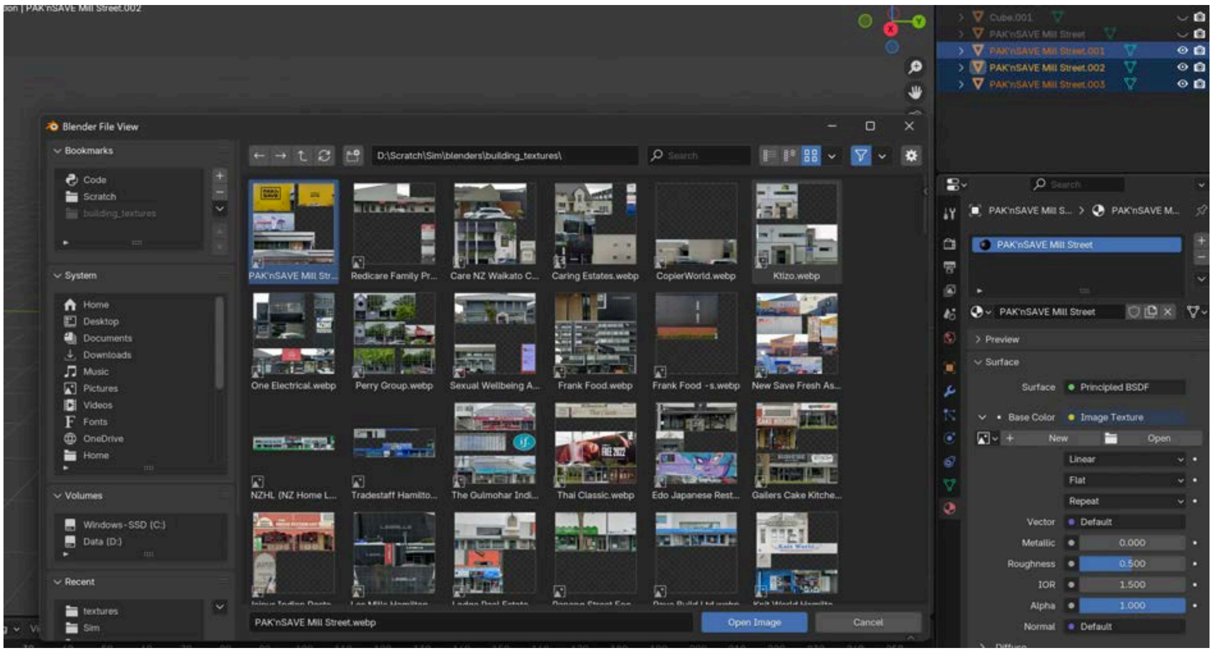


3. UV Map faces to textures

1. Create Materials linked to associated textures

If physical images get rename or removed, it cause texture missing for material

Use scripts to detect or rename path of image asset to relink correct texture



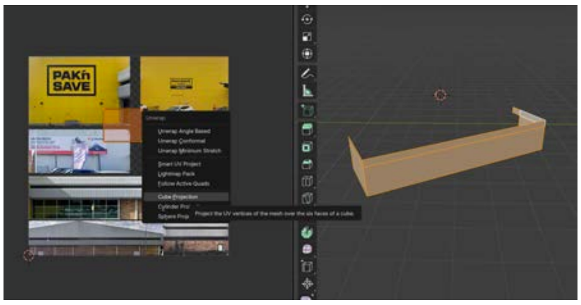
D:\Scratch\Sim\blenders\building_textures\

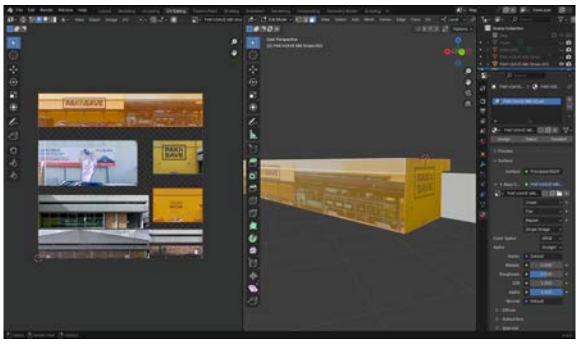
PAK'nSAVE Mill Street.webp

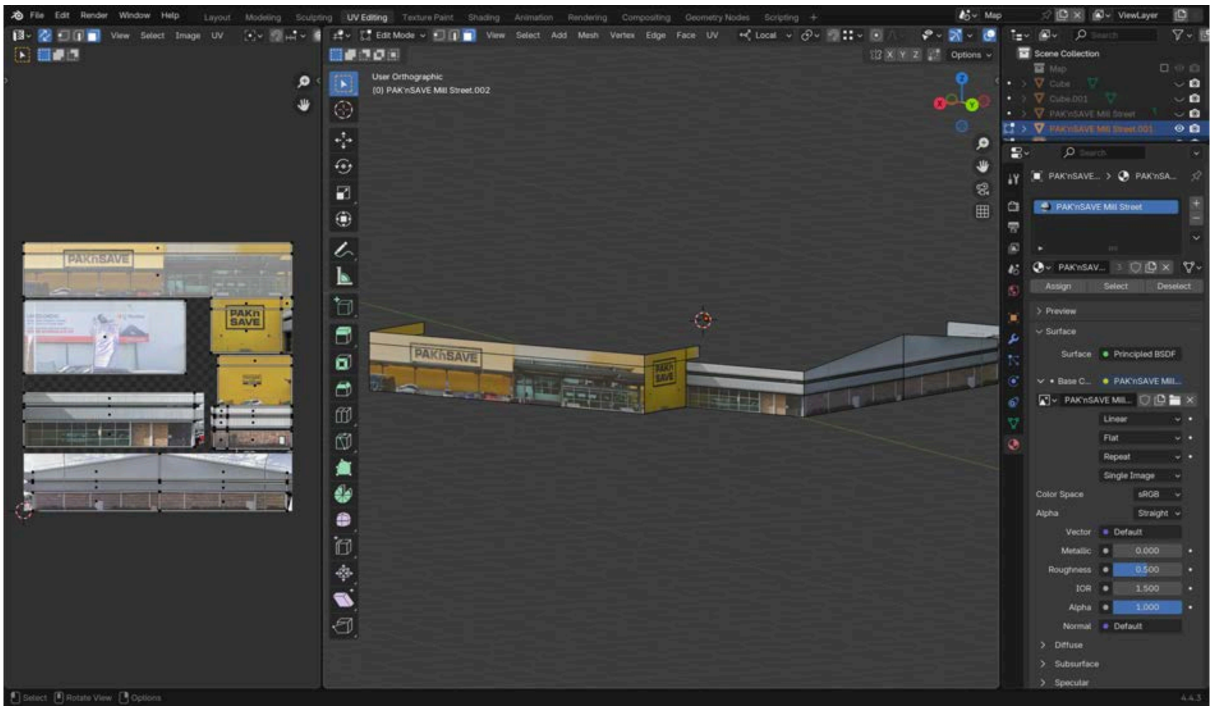
PAK'nSAVE Mill Street

Surface: Principled BSDF
Base Color: Image Texture
Linear
Flat
Repeat
Vector: Default
Metallic: 0.000
Roughness: 0.500
IOR: 1.500
Alpha: 1.000
Normal: Default

2. UV Editing for each face as to show popper images in all position

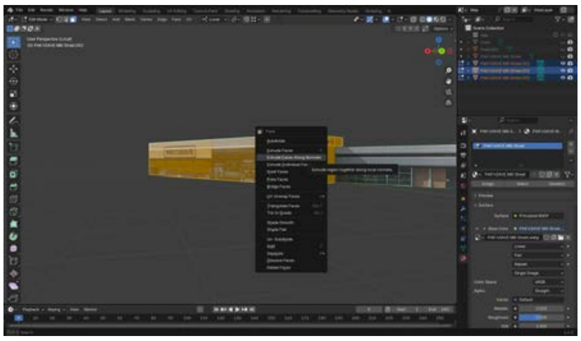


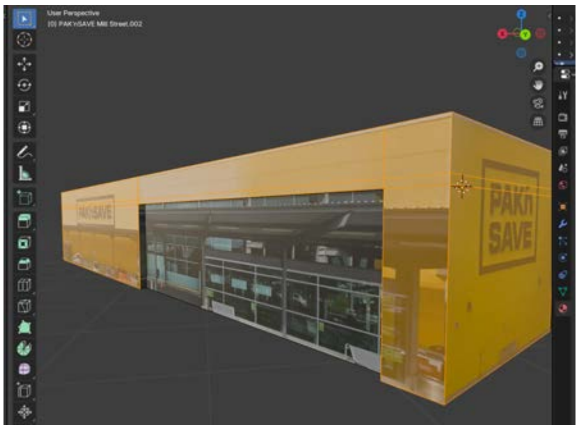


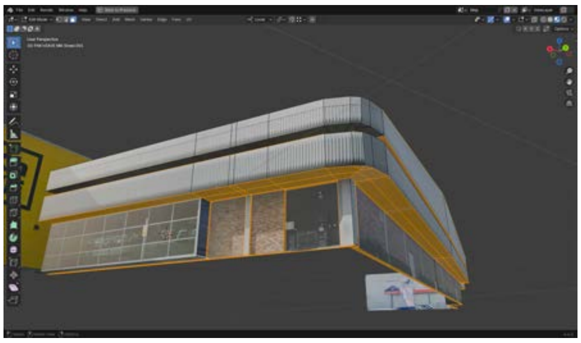
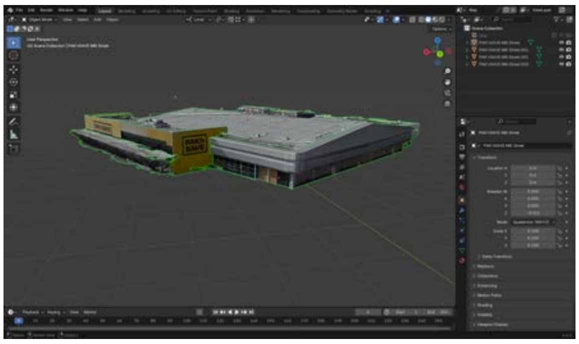


Surface: Principled BSDF
Base Color: PAK'nSAVE Mill...
PAK'nSAVE Mill...
Linear
Flat
Repeat
Single Image
Color Space: sRGB
Alpha: Straight
Vector: Default
Metallic: 0.000
Roughness: 0.500
IOR: 1.500
Alpha: 1.000
Normal: Default

4. Extrude faces to gain thickness to imitate real geometry

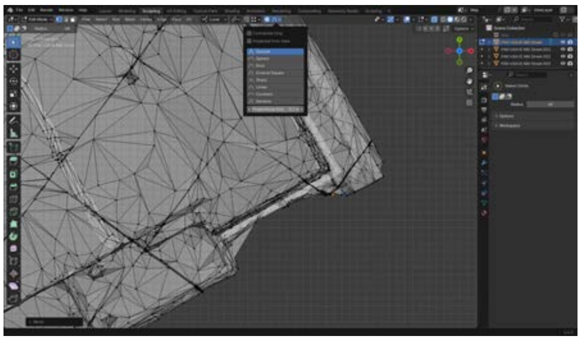
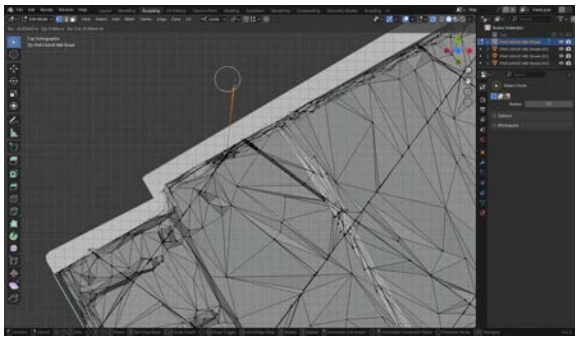




Simulate thickness of walls and roofs, which help realistic model in the game

5. Finalize modeling by remove outstanding vertices

Proportional option checked and drag vertices to flatten or remove outlet is a common practice. While use sculpture function is another approach

Export Objects to FBX

1. Organize objects hierarchy as each building has Top-levels objects and components as children

We are exporting object to FBX file, which act as prefab object in unity.

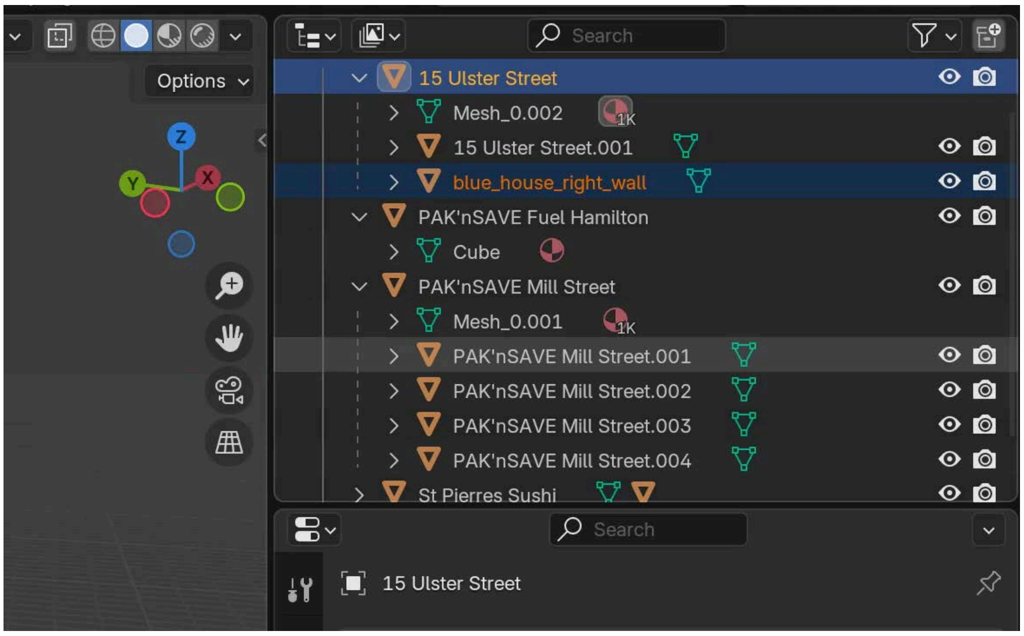
1. Separated zones as one prefab in unity

   TODO: set only top-level object as zone with children of all walls and roofs

   Advantage: adjust zone together

2. Individual walls as prefab

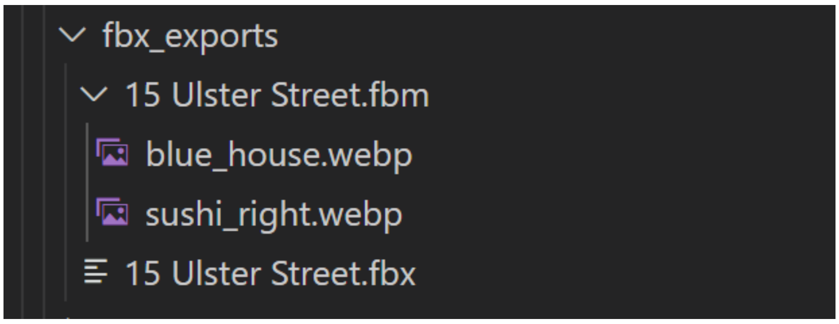
   Advantage: moderate walls individually



2. Use Scripts to export all Top-level objects with linked images in structures that Unity support

Expected textures file structure be like

- **FBX** files for models with textures linked to directory under the same parent
- Textures locate under directory with linked FBX's name with suffix `.fbm`



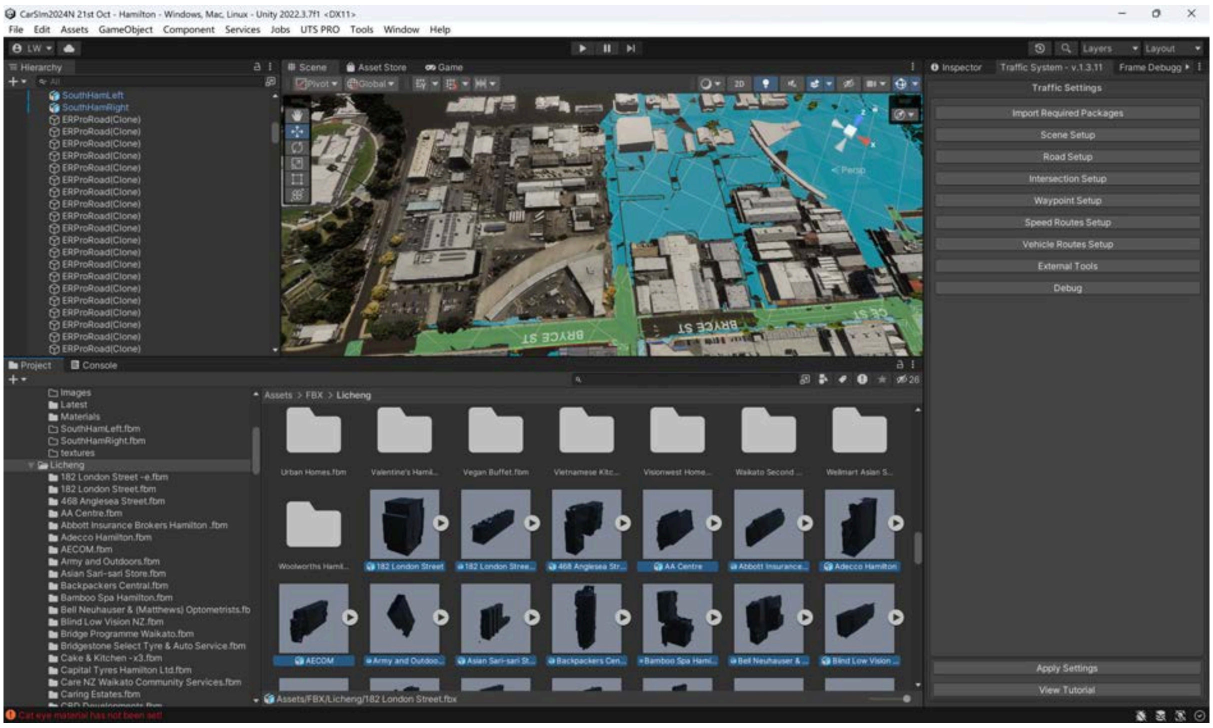
```
fbx_exports
  15 Ulster Street.fbm
    blue_house.webp
    sushi_right.webp
  15 Ulster Street.fbx
```

Python automatically export top-level objects in specific scenes

```
import bpy
import os

# Export all top-level objects in the current scene
def export_current_top_level_objects():
    view_layer_objects = set(bpy.context.view_layer.objects)

    # Find all objects without a parent (i.e., "top-level parent objects")
```

```
top_level_objects = [
    obj for obj in bpy.context.scene.objects
    if obj.parent is None
    and obj.type not in {'CAMERA', 'LIGHT'}
    and obj in view_layer_objects  # ✅ Only process objects accessible in the cu
]

count = 0
# Export each parent object along with its child objects
for parent_obj in top_level_objects:
    # Deselect all objects
    bpy.ops.object.select_all(action='DESELECT')
    parent_obj.select_set(True)

    # Select all child objects
    for child in parent_obj.children_recursive:
        if child in view_layer_objects:
            child.select_set(True)

    # Set the active object as the parent object (required by some exporters)
    bpy.context.view_layer.objects.active = parent_obj

    # Define export path
    export_path = os.path.join(export_dir, f"{parent_obj.name}.fbx")

    # Perform FBX export
    bpy.ops.export_scene.fbx(
        filepath=export_path,
        use_selection=True,
        apply_unit_scale=True,
        bake_space_transform=True,
        object_types={'MESH', 'EMPTY'},
        mesh_smooth_type='OFF',
        axis_up='Z',
        global_scale=10.0, # Google tiles scaled to 0.1 on import, not not set to 1
        path_mode='COPY', # Copy texture files
        embed_textures=False, # Don't embed textures: will generate a .fbm folde
    )

    count += 1
    return count

# Set export directory
export_dir = bpy.path.abspath("//fbx_exports/")
if not os.path.exists(export_dir):
    os.makedirs(export_dir)

exported = 0
for scene in bpy.data.scenes:
    if not scene.name.startswith("Scene"):
        continue  # Skip scenes not in the target list

    bpy.context.window.scene = scene  # Switch to target scene
    exported += export_current_top_level_objects()

print(f"export {exported} objects")
```

3. Copy to unity assets directory and use models as prefabs

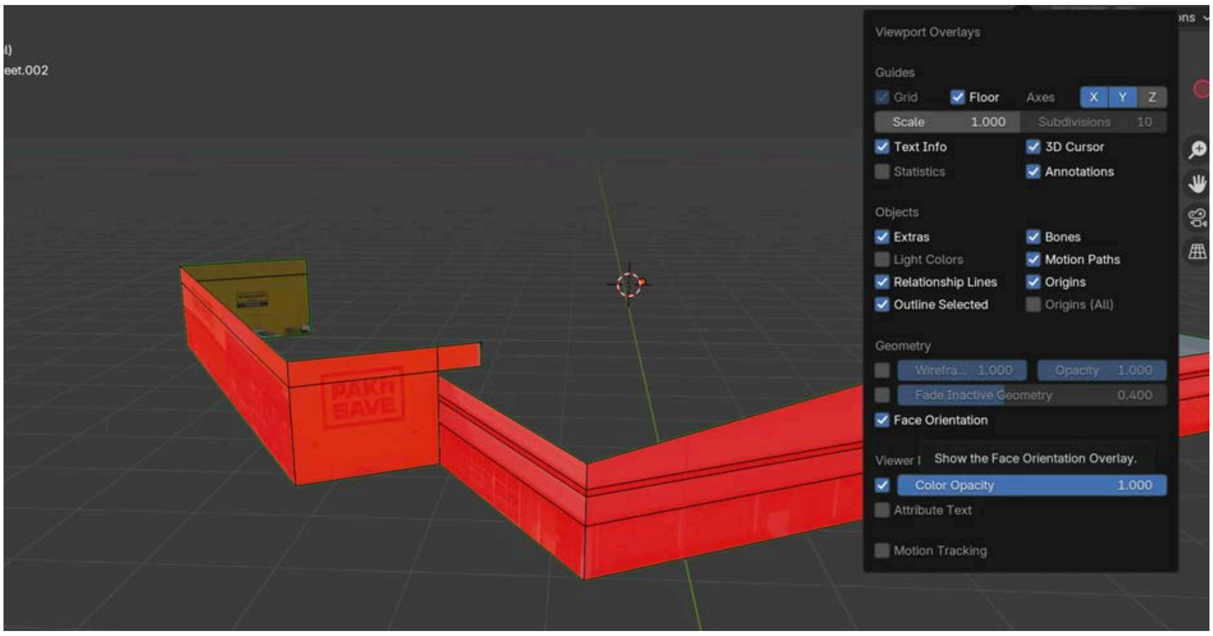


TROUBLESHOOTING

1. Unexpected Texture appearance (Transparent faces in Material Preview)

Cause: Faces orientation back to viewer

Fix: Spot back oriented face in viewport overlays, Select unexpected faces and enter SHIFT+N to turn over faces



Viewport Overlays

Guides

✓ Grid ✓ Floor Axes X Y Z

Scale 1.000 Subdivisions 10

✓ Text Info ✓ 3D Cursor

☐ Statistics ✓ Annotations

Objects

✓ Extras ✓ Bones

☐ Light Colors ✓ Motion Paths

✓ Relationship Lines ✓ Origins

✓ Outline Selected ☐ Origins (All)

Geometry

☐ Wirefra... 1.000 Opacity 1.000

☐ Fade Inactive Geometry 0.400

✓ Face Orientation

Viewer I Show the Face Orientation Overlay.

✓ Color Opacity 1.000

☐ Attribute Text

☐ Motion Tracking

2. Google tiles' texture are not included or embedded into FBX file as exported

Non-Principled BSDF shaders are excluded from texture baking, and their results will not be exported in the baked textures

Use python to adapt all Non-Principle BSDF to Principle BSDF

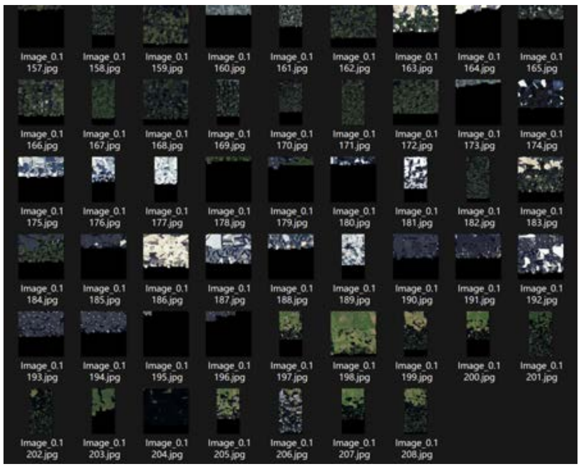
```
'''
resturcture all shaders to Principled BSDF
i.e. Google tiles by default get mix shader that unable to export with fbx, s
'''

import bpy

processed_objects = 0
skipped_objects = 0
processed_materials = set()

for obj in bpy.data.objects:
    if obj.type == 'MESH' and obj.material_slots:
        for slot in obj.material_slots:
            material = slot.material
            if material and material.use_nodes:
                if material.name in processed_materials:
                    continue

                already_valid = False
                for node in material.node_tree.nodes:
                    if node.type == 'BSDF_PRINCIPLED':
                        base_color_input = node.inputs['Base Color']
                        if base_color_input.is_linked:
                            from_node = base_color_input.links[0].from_node
                            if from_node.type == 'TEX_IMAGE':
                                already_valid = True
                                break

                if already_valid:
                    skipped_objects += 1
                    print(f"Skipped object: {obj.name}, material: {material.name} (
                    continue
                new_material = bpy.data.materials.new(name=f"New_{material.r
                new_material.use_nodes = True
                new_nodes = new_material.node_tree.nodes
                new_links = new_material.node_tree.links

                for node in new_nodes:
                    new_nodes.remove(node)

                bsdf = new_nodes.new(type='ShaderNodeBsdfPrincipled')
                bsdf.location = (0, 0)
                output = new_nodes.new(type='ShaderNodeOutputMaterial')
                output.location = (300, 0)
                new_links.new(bsdf.outputs['BSDF'], output.inputs['Surface'])

                for node in material.node_tree.nodes:
                    if node.type == 'TEX_IMAGE' and node.image:
                        tex = new_nodes.new('ShaderNodeTexImage')
                        tex.image = node.image
                        tex.location = (-300, 0)
                        new_links.new(tex.outputs['Color'], bsdf.inputs['Base Color
                        break

                slot.material = new_material
                processed_materials.add(material.name)
                processed_objects += 1
                print(f"Processed object: {obj.name}, material: {material.name}

print(f"\n✅ Done. Total processed objects: {processed_objects}, skipped
bpy.ops.wm.save_mainfile()
```

3. Google Tiles by default create enormously much materials and textures for the map tiles which slow down computing and file management, how to improve the efficiency?

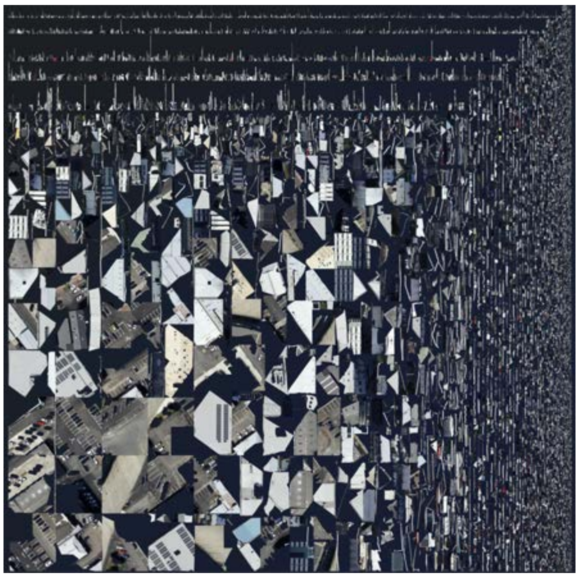
We can bake all textures into a single atlas texture e.g. bake 1000 textures of low resolution textures to a 4096 × 4096 high resolution texture

Normally small but many textures as image asset is work for both blender and unity rendering, but since our unity project utilized HDRP lit shader, small texture may corrupt in unity, therefore baking to a merged texture is necessary

For how to bake, follow demo videos on one drive project share

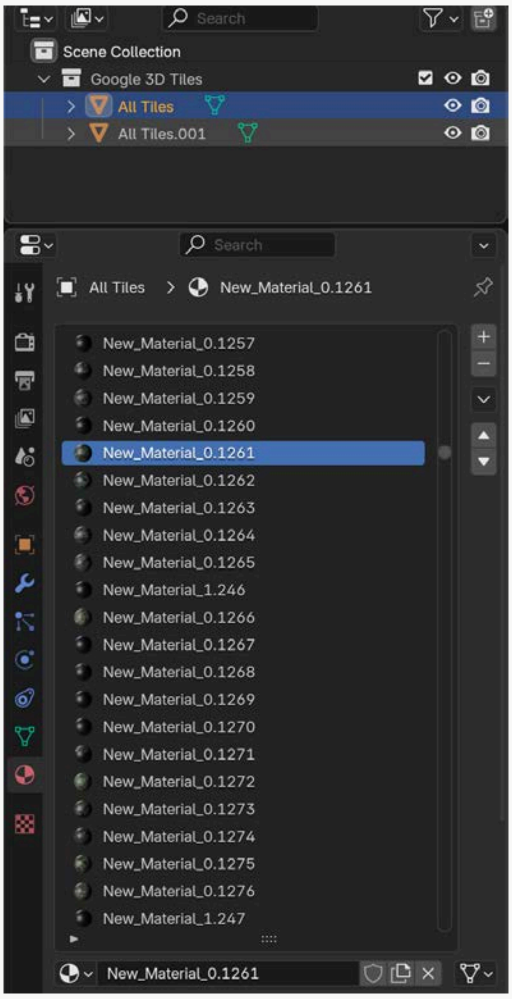


Lots of small textures created by BLOSM



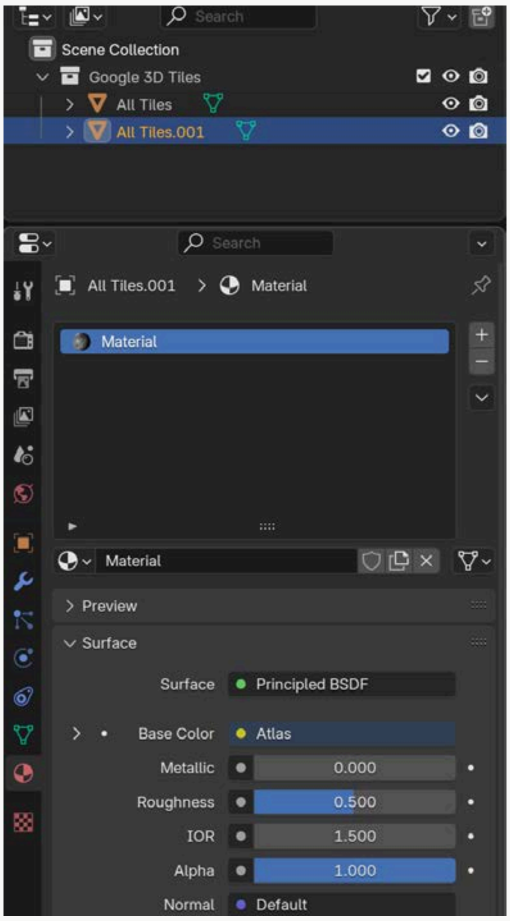
Baked texture belike

Baking supposed to have multiple textures replaced single texture

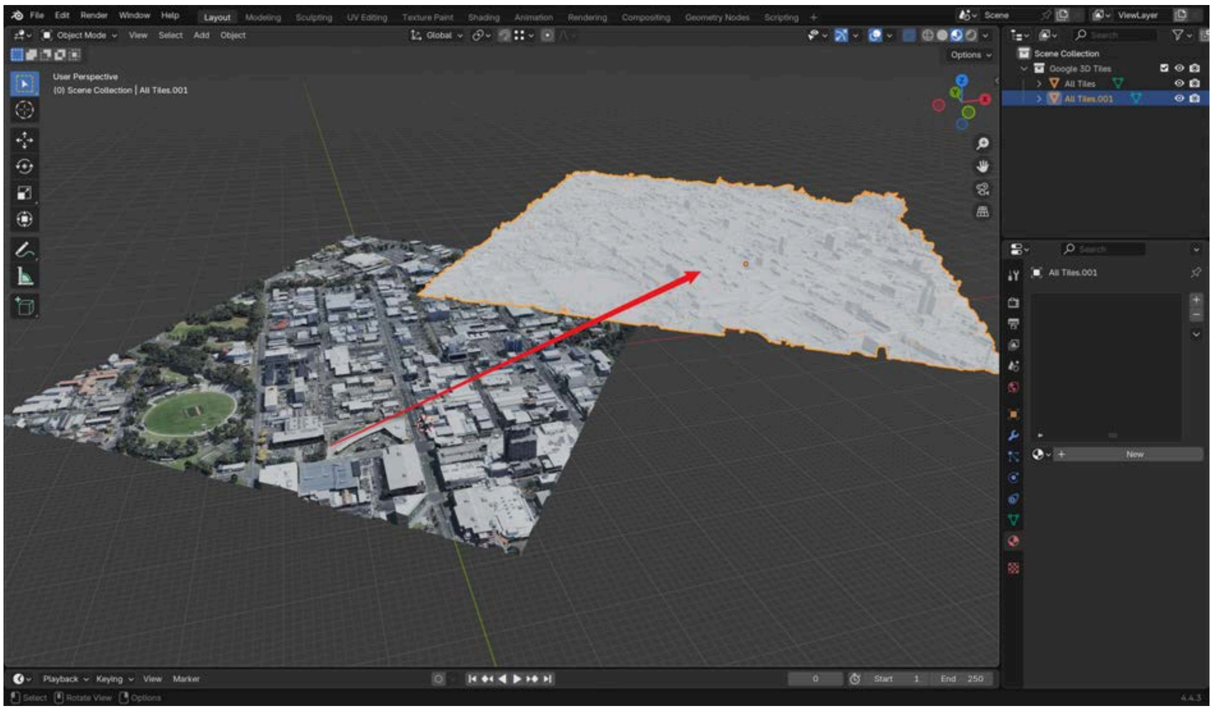


New_Material_0.1257
New_Material_0.1258
New_Material_0.1259
New_Material_0.1260
New_Material_0.1261
New_Material_0.1262
New_Material_0.1263
New_Material_0.1264
New_Material_0.1265
New_Material_1.246
New_Material_0.1266
New_Material_0.1267
New_Material_0.1268
New_Material_0.1269
New_Material_0.1270
New_Material_0.1271
New_Material_0.1272
New_Material_0.1273
New_Material_0.1274
New_Material_0.1275
New_Material_0.1276
New_Material_1.247

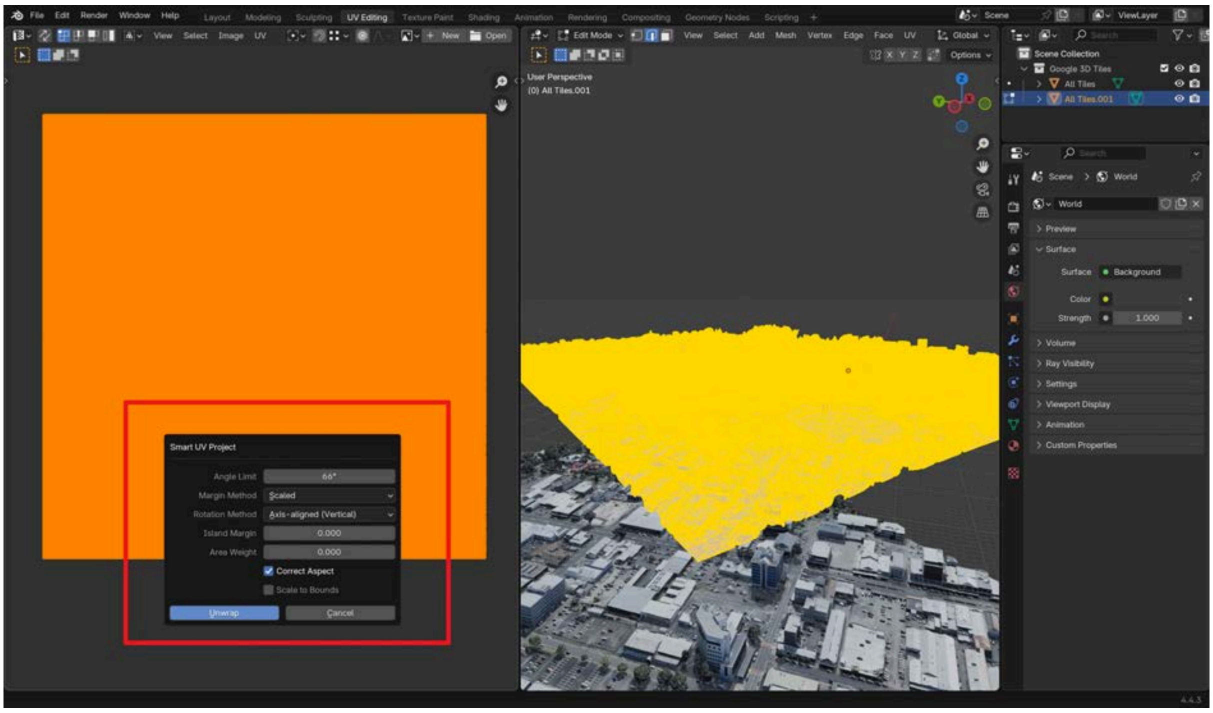
New_Material_0.1261



Surface: Principled BSDF
Base Color: Atlas
Metallic: 0.000
Roughness: 0.500
IOR: 1.500
Alpha: 1.000
Normal: Default



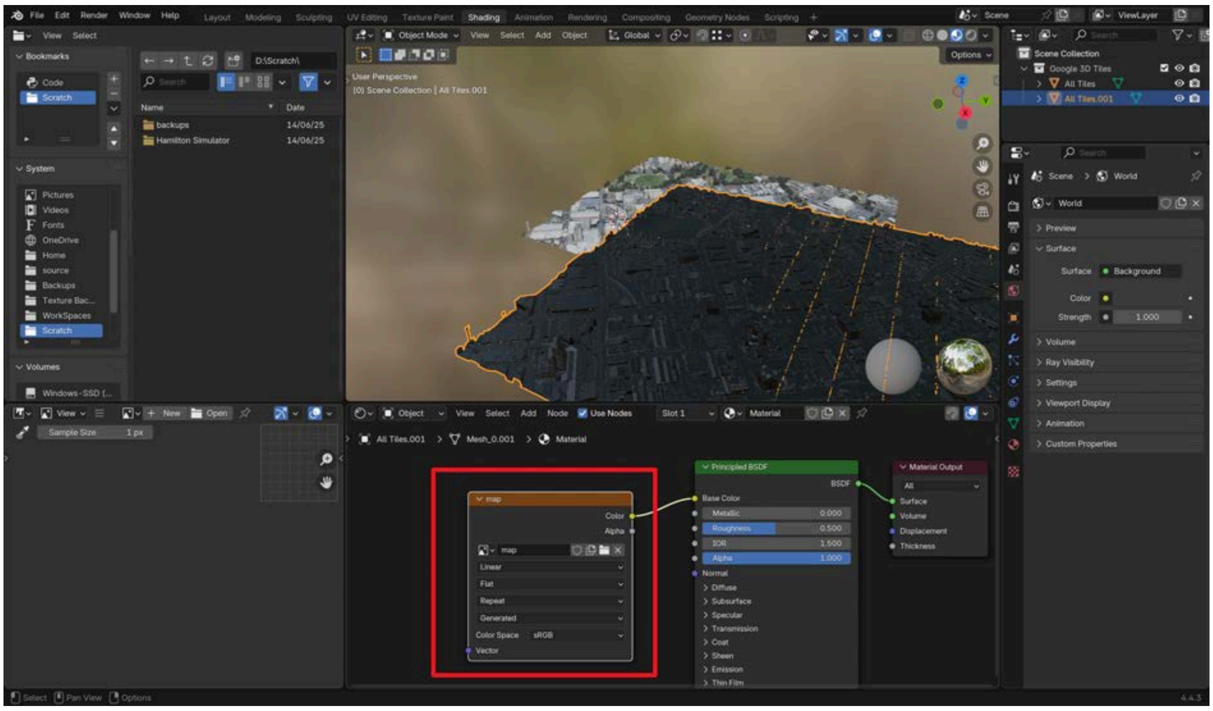
Remove all materials in each slot for New Google Tiles before baking



Smart UV Project

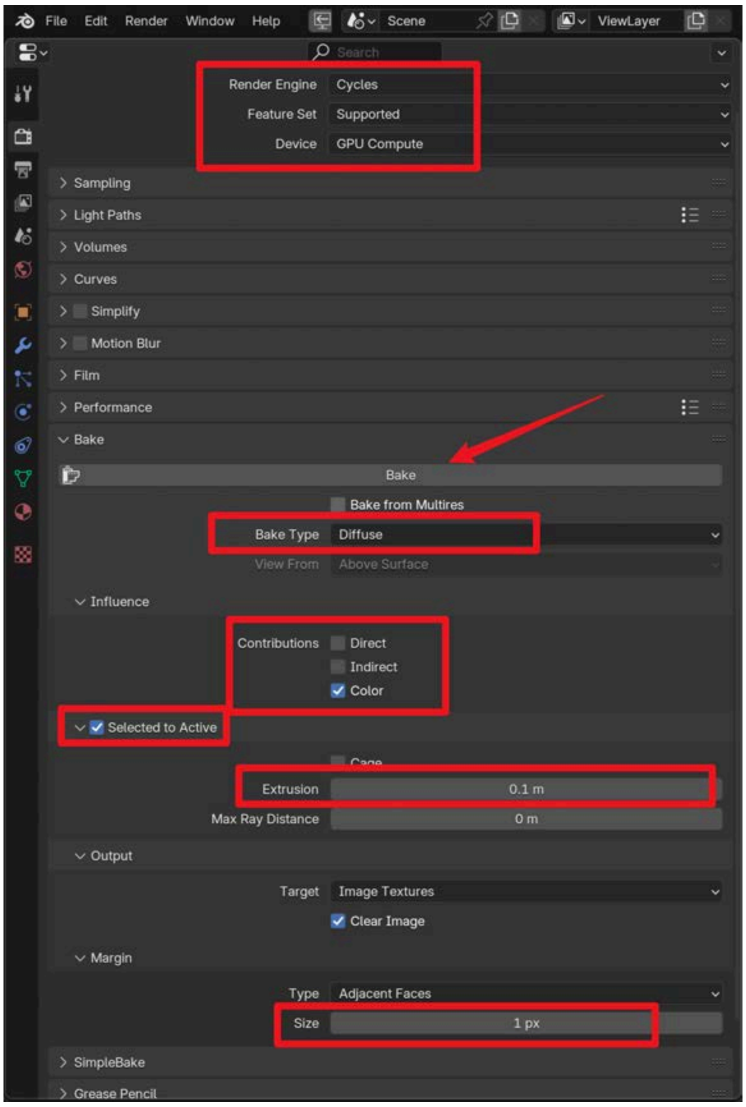
Angle Limit: 66°
Margin Method: Scaled
Rotation Method: Axis-aligned (Vertical)
Island Margin: 0.000
Area Weight: 0.000
[✓] Correct Aspect
[ ] Scale to Bounds

Smart UV set fro new Google Tile



Target image asset as destination of baking

🚨 **Make sure original Google Tile was selected and the new one is also selected and activated before baking**



Render Engine: Cycles
Feature Set: Supported
Device: GPU Compute

Sampling
Light Paths
Volumes
Curves
Simplify
Motion Blur
Film
Performance
Bake

Bake
Bake from Multires
Bake Type: Diffuse
View From: Above Surface

Influence
Contributions: Direct, Indirect, ✓ Color

✓ Selected to Active
Cage
Extrusion: 0.1 m
Max Ray Distance: 0 m

Output
Target: Image Textures
✓ Clear Image

Margin
Type: Adjacent Faces
Size: 1 px

SimpleBake
Grease Pencil

Bake settings

💡 **Use Python to remove all materials for Google Tiles before baking**

🚨 **Select object to process before running the scripts**

```
'''
clear materials in every slot for selected object
i.e. pre-processing for baking Google tiles in order to get the tile mapped
'''
import bpy

obj = bpy.context

for i in range(0,len(obj.object.material_slots)):
    obj.object.active_material_index = 1
    bpy.ops.object.material_slot_remove()

bpy.ops.object.mode_set(mode = 'EDIT')
bpy.ops.mesh.select_all(action = 'SELECT')
bpy.ops.object.material_slot_assign()
bpy.ops.object.mode_set(mode = 'OBJECT')
```

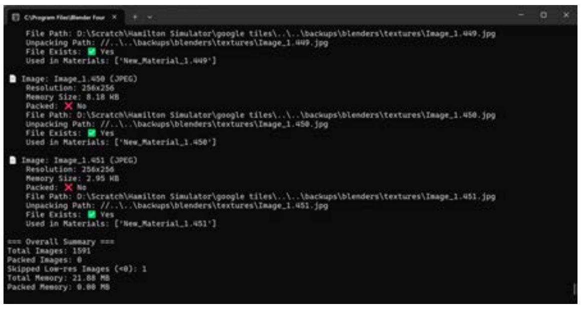
4. Take advantage of python to check all image assets stats in your blender

- You can manage your images with more confidence

```
'''
Report all image assets' stats and link states.

For each image, report:
   1. Physical path
   2. Unpacking path (if different from physical path)
   3. Packed or not
```

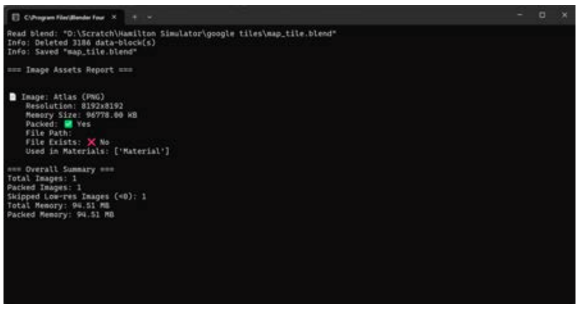
```
    4. Memory size
    5. Missing or not
    6. Used materials
'''

import bpy
import os

# Skip assets with resolution (smaller dimension) lower than this
SKIP_RESO = 0

def get_img_filesize(img: bpy.types.Image):
    """Get image file size from disk or packed data."""
    if img.packed_file is not None:
        return img.packed_file.size

    img_path = os.path.abspath(bpy.path.abspath(img.filepath))

    if not os.path.exists(img_path):
        print(f"⚠️ File not found: {img_path}")
        return 0

    try:
        return os.path.getsize(img_path)
    except Exception as e:
        print(f"❌ Failed to get file size: {img.name}, error: {e}")
        return 0

def find_used_materials(img: bpy.types.Image):
    """Find which materials are using this image."""
    used_in_materials = []
    for mat in bpy.data.materials:
        if not mat.use_nodes:
            continue
        for node in mat.node_tree.nodes:
            if node.type == 'TEX_IMAGE' and node.image == img:
                used_in_materials.append(mat.name)
    return used_in_materials

# Start reporting
```

```
print("\n=== Image Assets Report ===\n")

total_mem, count, packed, packed_mem = 0, 0, 0, 0
small_img_count = 0

for img in bpy.data.images:
    w, h = img.size

    if max(w, h) <= SKIP_RESO:
        small_img_count += 1
        continue

    mem = get_img_filesize(img)
    is_packed = img.packed_file is not None
    filepath = bpy.path.abspath(img.filepath)
    file_exists = os.path.exists(filepath)
    used_materials = find_used_materials(img)

    # Per-image report
    print(f"\n📄 Image: {img.name} ({img.file_format})")
    print(f"    Resolution: {w}x{h}")
    print(f"    Memory Size: {mem / 1024:.2f} KB")
    print(f"    Packed: {'✅ Yes' if is_packed else '❌ No'}")
    print(f"    File Path: {filepath}")
    if filepath != img.filepath_raw:
        print(f"    Unpacking Path: {img.filepath_raw}")
    print(f"    File Exists: {'✅ Yes' if file_exists else '❌ No'}")
    print(f"    Used in Materials: {used_materials if used_materials else '🚫 None'}"

    count += 1
    if is_packed:
        packed += 1
        packed_mem += mem
    total_mem += mem

# Summary
print("\n=== Overall Summary ===")
print(f"Total Images: {count}")
print(f"Packed Images: {packed}")
print(f"Skipped Low-res Images (<{SKIP_RESO}): {small_img_count}")
print(f"Total Memory: {total_mem / (1024 ** 2):.2f} MB")
print(f"Packed Memory: {packed_mem / (1024 ** 2):.2f} MB")
```



```
   File Path: D:\Scratch\Hamilton Simulator\google tiles\..\..\backups\blenders\textures\Image_1.449.jpg
   Unpacking Path: //..\..\backups\blenders\textures\Image_1.449.jpg
   File Exists: ✅ Yes
   Used in Materials: ['New_Material_1.449']

📄 Image: Image_1.450 (JPEG)
   Resolution: 256x256
   Memory Size: 8.18 KB
   Packed: ❌ No
   File Path: D:\Scratch\Hamilton Simulator\google tiles\..\..\backups\blenders\textures\Image_1.450.jpg
   Unpacking Path: //..\..\backups\blenders\textures\Image_1.450.jpg
   File Exists: ✅ Yes
   Used in Materials: ['New_Material_1.450']

📄 Image: Image_1.451 (JPEG)
   Resolution: 256x256
   Memory Size: 2.95 KB
   Packed: ❌ No
   File Path: D:\Scratch\Hamilton Simulator\google tiles\..\..\backups\blenders\textures\Image_1.451.jpg
   Unpacking Path: //..\..\backups\blenders\textures\Image_1.451.jpg
   File Exists: ✅ Yes
   Used in Materials: ['New_Material_1.451']

=== Overall Summary ===
Total Images: 1591
Packed Images: 0
Skipped Low-res Images (<0): 1
Total Memory: 21.88 MB
Packed Memory: 0.00 MB
```

Images stats before baking



```
Read blend: "D:\Scratch\Hamilton Simulator\google tiles\map_tile.blend"
Info: Deleted 3186 data-block(s)
Info: Saved "map_tile.blend"

=== Image Assets Report ===

📄 Image: Atlas (PNG)
   Resolution: 8192x8192
   Memory Size: 96778.00 KB
   Packed: ✅ Yes
   File Path:
   File Exists: ❌ No
   Used in Materials: ['Material']

=== Overall Summary ===
Total Images: 1
Packed Images: 1
Skipped Low-res Images (<0): 1
Total Memory: 94.51 MB
Packed Memory: 94.51 MB
```

Image stats after baking