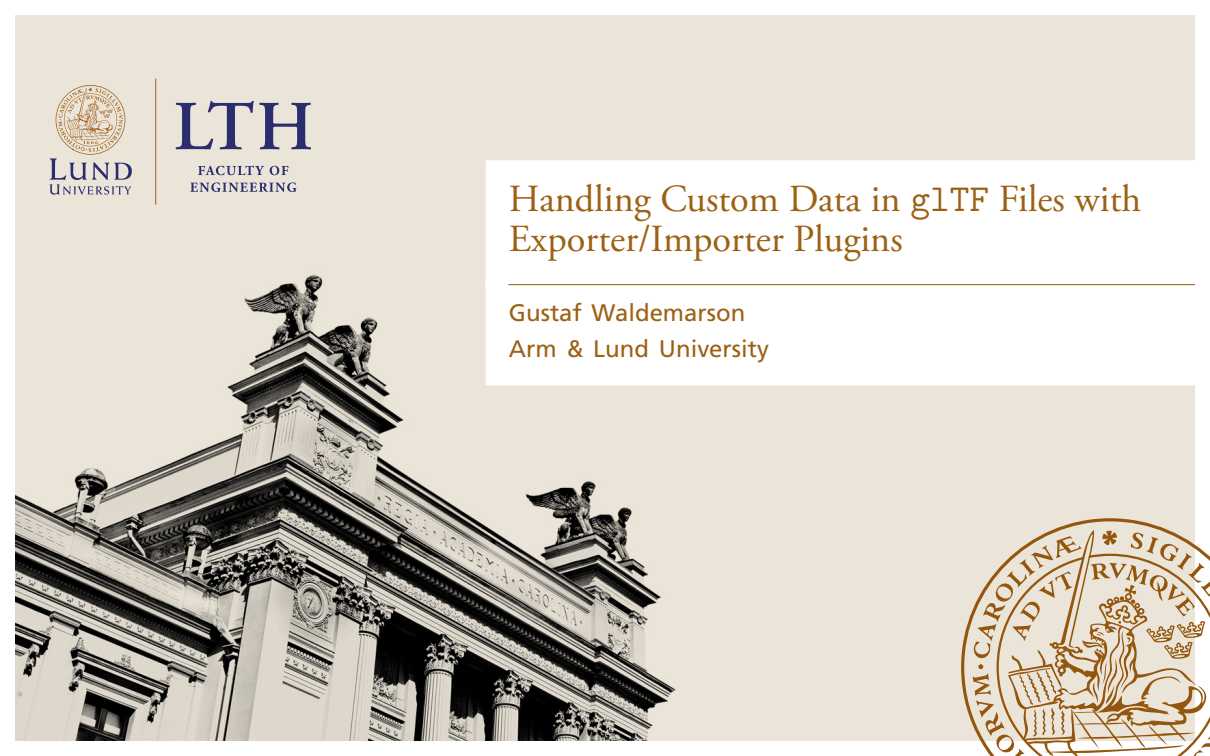


Handling Custom Data in glTF Files with Exporter/Importer Plugins

Gustaf Waldemarson
Arm & Lund University

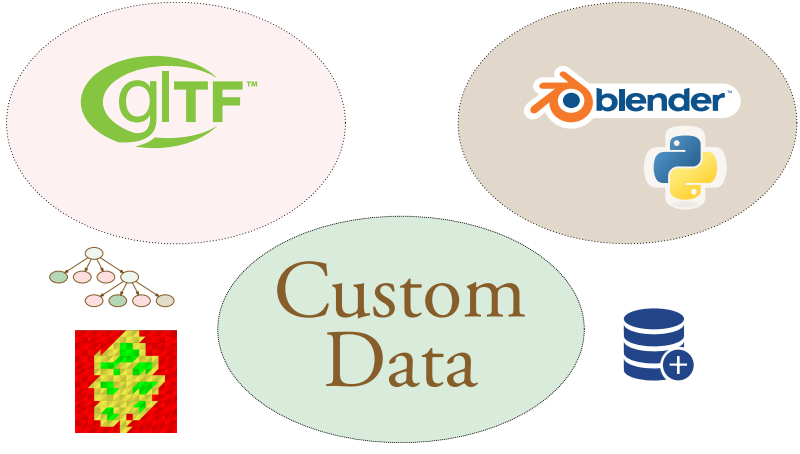
Who am I?

- Industrial PhD Student at the Lund University Graphics Group
- Software Engineer at Arm

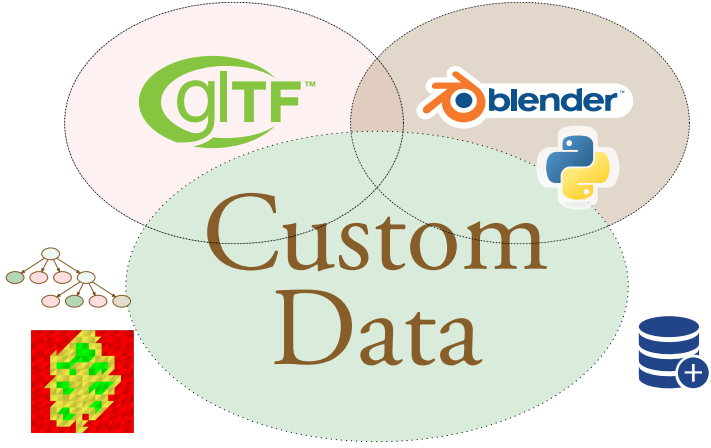
arm

- https://gustafwaldemarson.com

What is all this?



What is all this?



Agenda

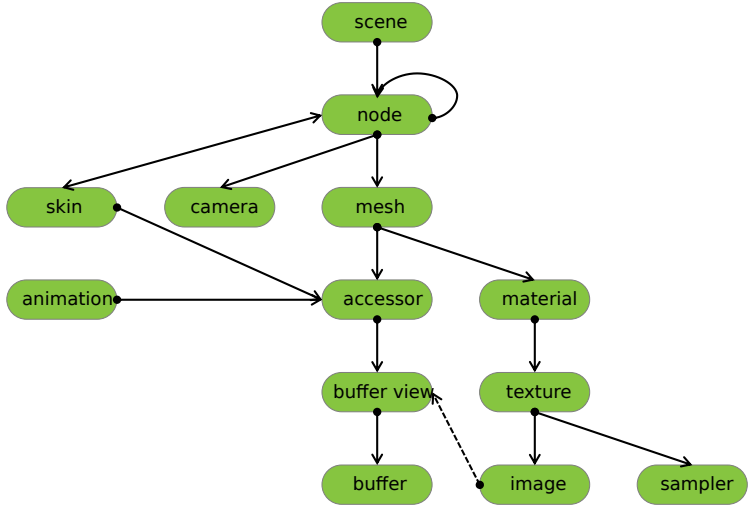
- `glTF`?
  - What is it?
  - Data-models
- The `glTF` Importer/Exporter Addon
  - `Blender`
  - `tinyglTF`
- Extension Mechanism
- Example Plugin: Watermarking
- Customization Functions / Hooks
- Micromap Generator Plugin
- Importers

What is glTF?

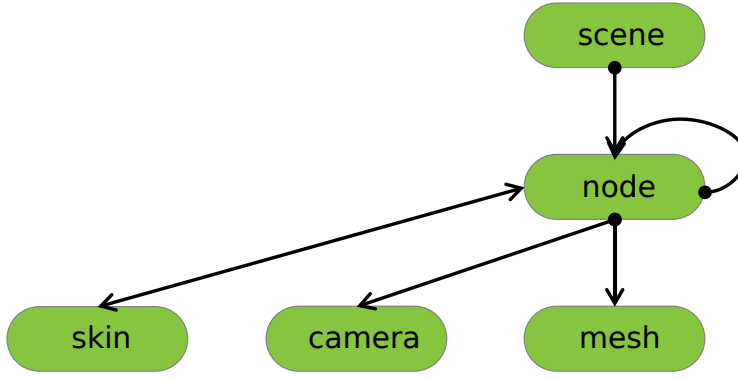
Types of glTF Files

- `GLTF_EMBEDDED` (`.gltf`)
- `GLTF_SEPARABLE` (`.gltf`, `.bin` + `textures`)
- `GLTF_BINARY` (`.glb`)

Overview

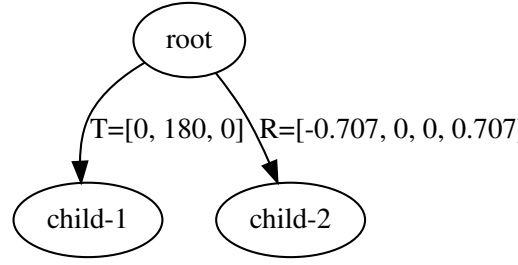

scene
node
skin
camera
mesh
animation
accessor
material
buffer view
texture
buffer
image
sampler

Scene Graph

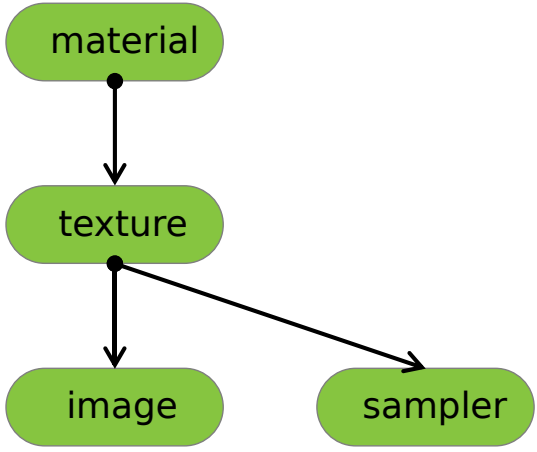


Scene Graph

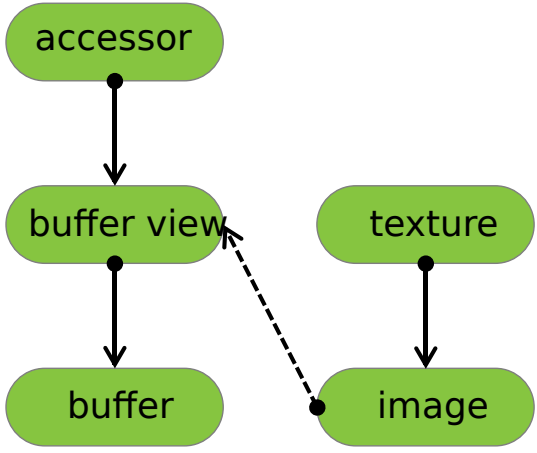
```
"scenes": [ { "nodes": [0] } ],
"nodes": [
    {
        "name": "root",
        "children": [1, 2]
    },
    {
        "name": "child-1",
        "translation": [ 0, 180, 0 ],
    },
    {
        "name": "child-2",
        "rotation": [-0.707, 0, 0, 0.707],
    }
]
```



Shading Model



Data Storage



Buffers, Views and Accessors

`Accessors`

- What to access
- How to access
- `BufferView`

`Texture`

- `Image`

`Image`

- `BufferView`
- `URI`

`BufferView`

- `byteStride`
- `byteLength`
- `byteOffset`

`Buffer`

- `byteLength`
- `URI`

Buffers, Views and Accessors

Inside Blender…

- `Accessor` ⇒ `Accessor`
- `Texture` ⇒ `Texture`
- `Image` ⇒ `Image`
- `BufferView` ⇒ `BinaryData`
- `Buffer` ⇒ `BinaryData`
- `Image` ⇒ `ImageData`

`https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html`
`https://www.khronos.org/files/gltf20-reference-guide.pdf`

The Blender glTF I/O Addon

tinygltf

```
template<class T>
std::vector<T> gltf_access(const tinygltf::Model &gltf,
                           const tinygltf::Accessor &acc)
{
    auto &bv = gltf.bufferViews[acc.bufferView];
    auto &b = gltf.buffers[bv.buffer];
    // ~100 lines to extract the data.
}
```

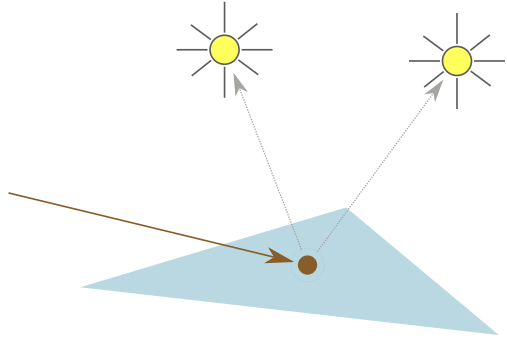
The Blender glTF I/O Addon

Blender

```
import io_scene_gltf2.io.exp.gltf2_io_binary_data as exp
import io_scene_gltf2.io.imp.gltf2_io_binary_data as imp
indices = [0, 1, 2]
binary_data = exp.BinaryData(bytes(indices))
decoded = imp.BinaryData.decode_accessor_internal(binary_data)
```

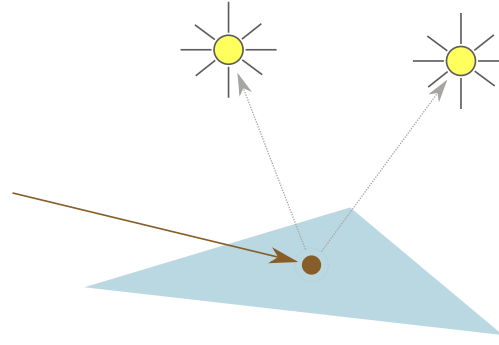
The Extension Mechanism

```
{
  "extensionsUsed":[
    "KHR_lights_punctual",
    "KHR_materials_transmission"
  ],
  "extensionsRequired":[
    "KHR_lights_punctual"
  ],
  "extensions":{
    "KHR_lights_punctual":{
      "lights":[]
    }
  }
}
```



The Extension Mechanism

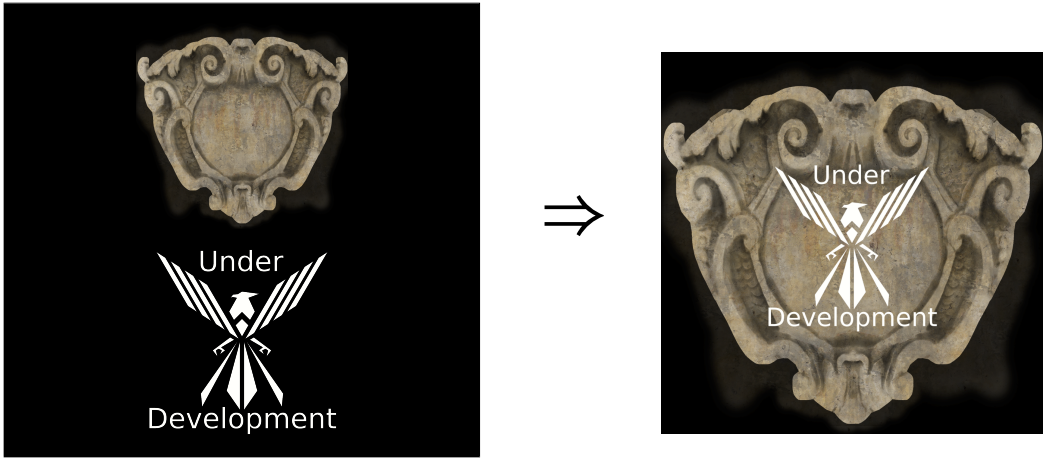
```
"nodes": [
    {
        "extensions":{
            "KHR_lights_punctual":{
                "light":0
            }
        },
        "name":"Point",
        "rotation":[],
        "translation":[]
    },
]
```



https://github.com/KhronosGroup/glTF/tree/main/extensions

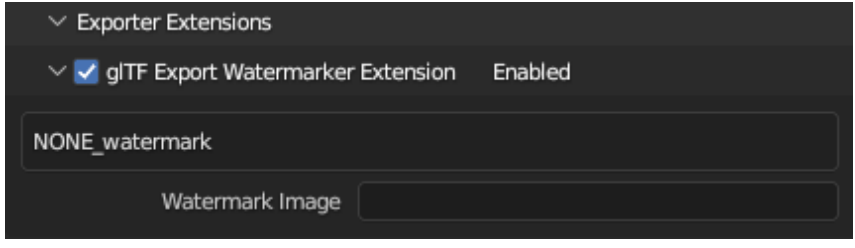
Texture Watermarking

A First Use-Case



Extension Plugins – 2

Panel



Extension Plugins – 2

bl_info


```
bl_info = {
    "name": "glTF Export Watermarker Extension",
    "category": "Import-Export",
    "version": (1, 0, 0),
    "blender": (3, 0, 0),
    "location": "File > Export > glTF 2.0",
    "description": "Watermark any exported image texture.",
    "author": "Gustaf Waldemarson",
}
```

https://wiki.blender.org/wiki/Process/Addons/Guidelines/metainfo

Extension Plugins – 3

Properties


```
class WatermarkingExtensionProperties(bpy.types.PropertyGroup):

    enabled: bpy.props.BoolProperty(
        name="glTF Export Watermarker Extension",
    )
    watermark: bpy.props.StringProperty(
        name="Watermark image",
    )
```

Extension Plugins – 4


```
class GLTF_PT_UserExtensionWatermarkingPanel(bpy.types.Panel):
    bl_space_type = 'FILE_BROWSER'
    bl_region_type = 'TOOL_PROPS'
    bl_label = 'Enabled'
    bl_parent_id = 'GLTF_PT_export_user_extensions'
    bl_options = {'DEFAULT_CLOSED'}
```

Extension Plugins – 5


```
def draw_header(self, context):
    props = bpy.context.scene.WatermarkingExtensionProperties
    self.layout.prop(props, 'enabled')

def draw(self, context):
    layout = self.layout
    layout.use_property_split = True
    layout.use_property_decorate = False
    props = bpy.context.scene.WatermarkingExtensionProperties
    layout.active = props.enabled
    box = layout.box()
    box.label(text="NONE_watermark")
    layout.prop(props, "watermark", text="Watermark Image")
```

Extension Plugins – 6

Registration


```
def register():
    bpy.utils.register_class(WatermarkingExtensionProperties)
    prop = bpy.props.PointerProperty(type=WatermarkingExtensionProperties)
    bpy.types.Scene.WatermarkingExtensionProperties = prop


def register_panel():
    try:
        bpy.utils.register_class(GLTF_PT_UserExtensionWatermarkingPanel)
    except Exception:
        pass
    return unregister_panel
```

Extension Plugins – 7

Un-Registration


```
def unregister_panel():
    try:
        bpy.utils.unregister_class(GLTF_PT_UserExtensionWatermarkingPanel)
    except Exception:
        pass


def unregister():
    unregister_panel()
    bpy.utils.unregister_class(WatermarkingExtensionProperties)
    del bpy.types.Scene.WatermarkingExtensionProperties
```

Extension Plugins – 8

The Exporter Class


```
class glTF2ExportUserExtension:
  def __init__(self):
    from io_scene_gltf2.io.com.gltf2_io_extensions import Extension
    self.props = bpy.context.scene.WatermarkingExtensionProperties

  def gather_image_hook(self,
                        gltf2_image,
                        blender_shader_sockets,
                        export_settings):
    watermark(gltf2_image)
```

Extension Plugins – 9

Watermarking

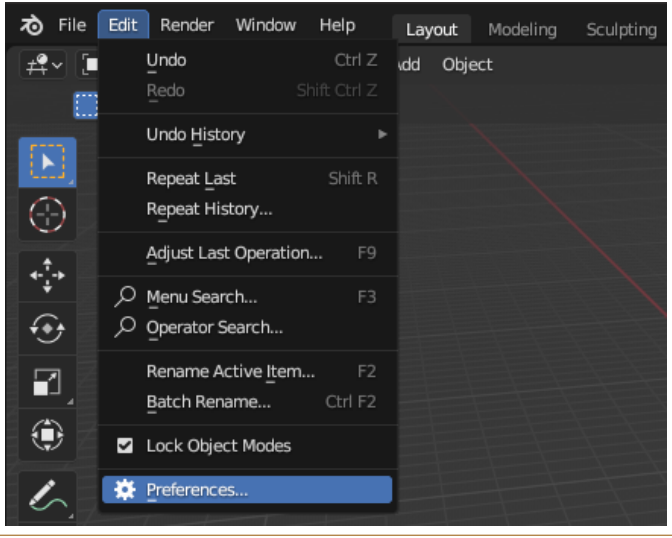

```
def watermark(self, gltf_image):
    img = img2numpy(gltf_image)
    img = watermark_internal(img, mark)
    data = img2memory(gltf_image.mime_type, img)
    mime = gltf_image.mime_type
    if gltf_image.uri:
        import io_scene_gltf2.io.exp.gltf2_io_image_data as gltf
        name = gltf_image.uri.name
        gltf_image.uri = gltf.ImageData(data, mime, name)
    else:
        import io_scene_gltf2.io.exp.gltf2_io_binary_data as gltf
        gltf_image.buffer_view = gltf.BinaryData(data)
```

Extension Plugins – 10

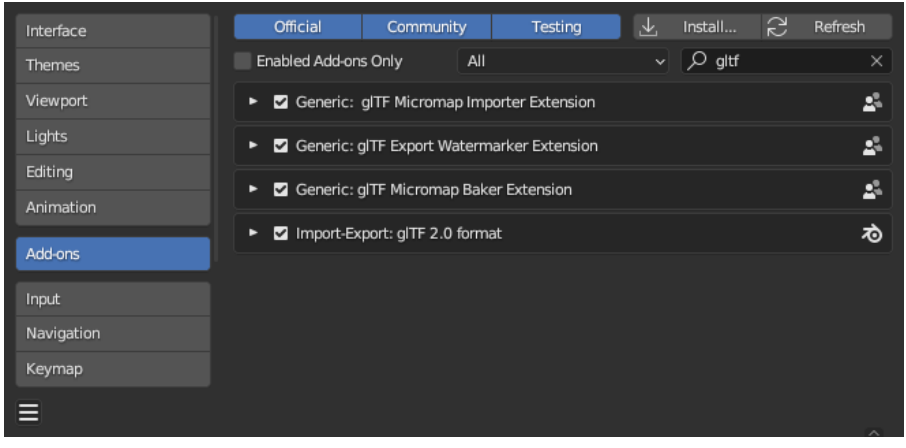
The Extension Class


```
def gather_node_hook(self, gltf_obj, bl_object, export_settings):
    key = extension_name
    if self.properties.enabled:
        if gltf_obj.extensions is None:
            gltf_obj.extensions = {}
        gltf_obj.extensions[key] = self.Extension(
            name=key,
            extension={"float": self.properties.float_property},
            required=False,
        )
```

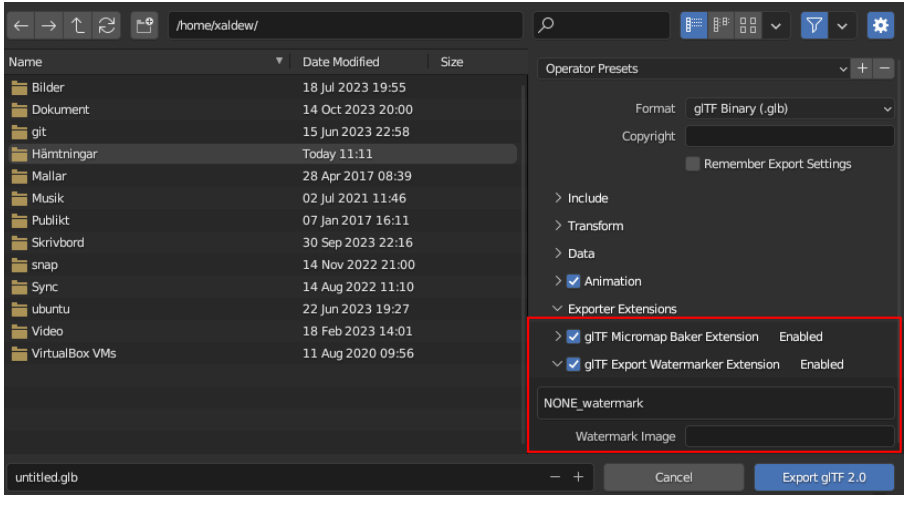
Installing and Enabling Plugins



Installing and Enabling Plugins



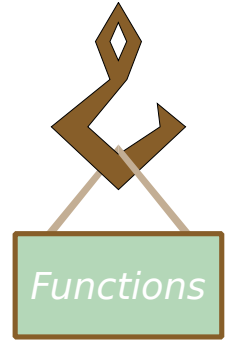
Installing and Enabling Plugins



Hooks

Available Customization Methods

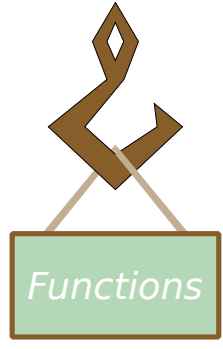
```
gather_animation_hook(...)
gather_animation_channel_hook(...)
gather_animation_channel_target_hook(...)
gather_animation_sampler_hook(...)
gather_asset_hook(...)
gather_camera_hook(...)
gather_gltf_extensions_hook(...)
gather_image_hook(...)
gather_joint_hook(...)
gather_material_hook(...)
gather_material_pbr_metallic_roughness_hook(...)
gather_material_unlit_hook(...)
gather_mesh_hook(...)
gather_node_hook(...)
gather_node_name_hook(...)
gather_sampler_hook(...)
# ...
gather_scene_hook(...)
```



Available Customization Methods

My Overrides

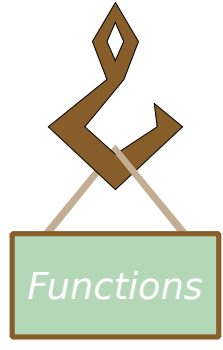
```
gather_image_hook(...)
gather_mesh_hook(...)
gather_primitive_hook(...)
```



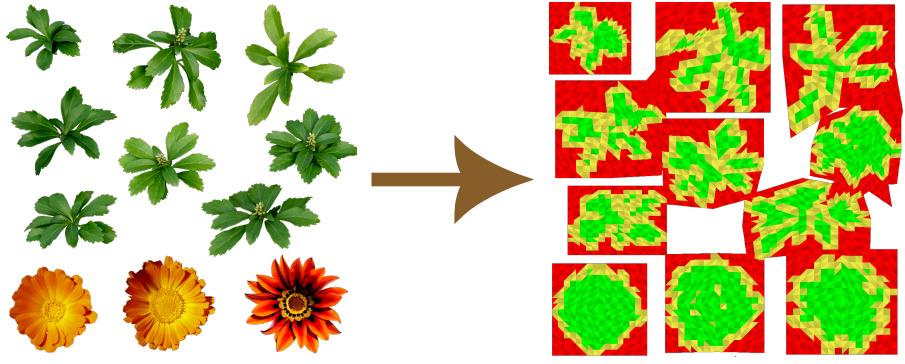
Available Customization Methods

General Structure

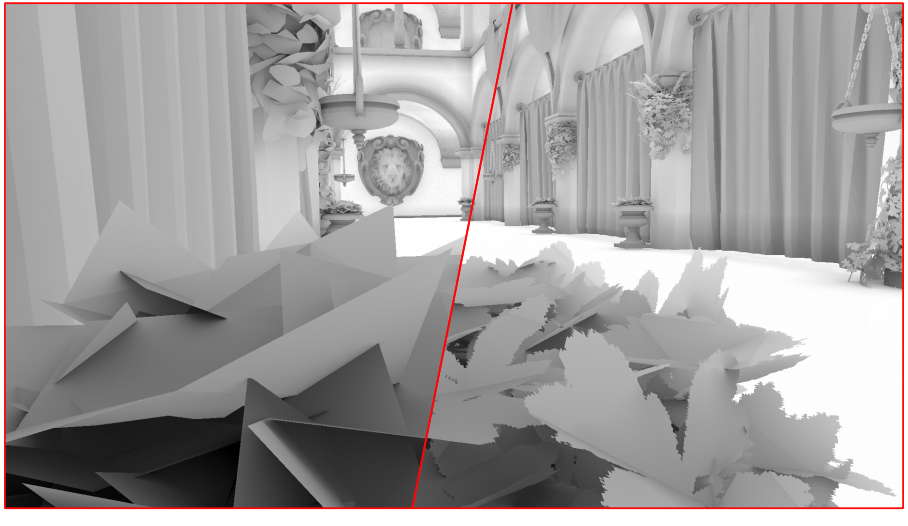
```
gather_mesh_hook(self,
                 gltf2_mesh,
                 blender_mesh,
                 blender_object,
                 vertex_groups,
                 modifiers,
                 materials,
                 export_settings)
```



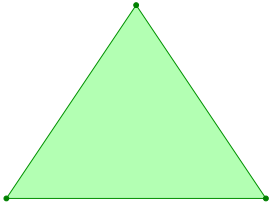
Micromaps – VK_EXT_opacity_micromap

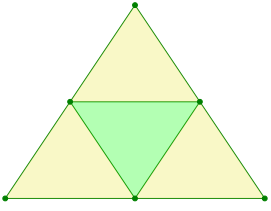


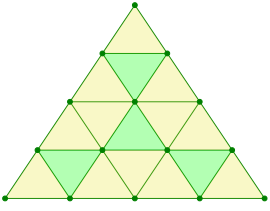
Micromaps – VK_EXT_opacity_micromap

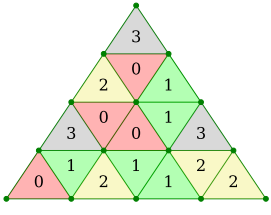


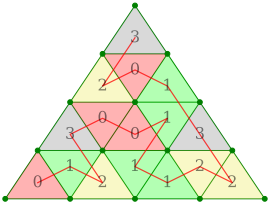
Micromaps – VK_EXT_opacity_micromap

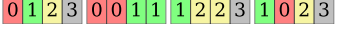






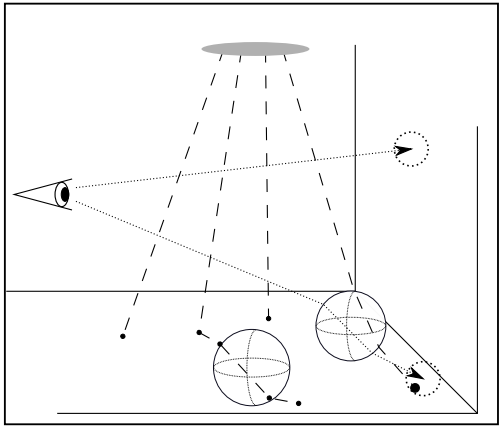




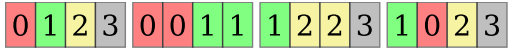


Opacity Micromaps

0 Transparent
1 Opaque
2 Unknown (Transparent)
3 Unknown (Opaque)



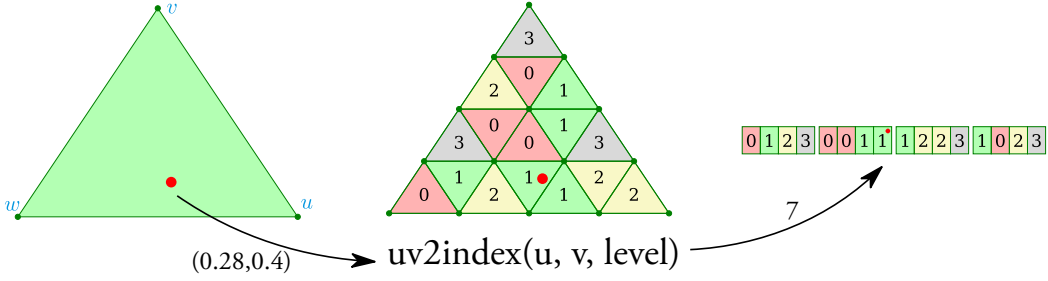
Opacity Micromaps



⇒

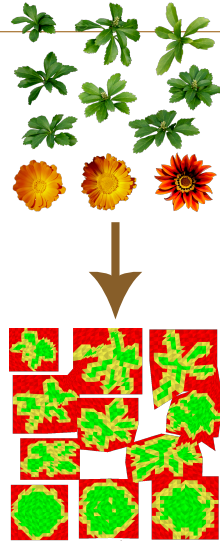
```
0b10100110 10100110 10100110 10100110
```

Micromaps – VK_EXT_opacity_micromap



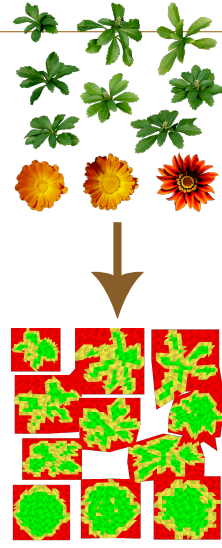
Opacity Micromaps Exporter

```
for triangle in scene:
    if tri.alpha_map:
        generate_micromaps(tri.alpha_map)
```

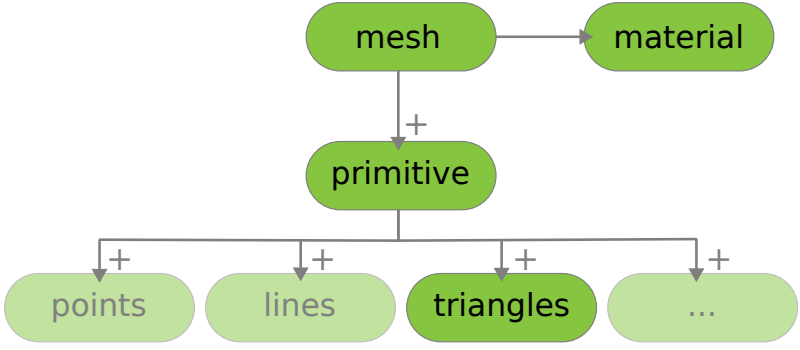


Opacity Micromaps Exporter

```
"meshes":[
{
  "name":"Plane.001",
  "primitives":[
    {
      "extensions":{
        "NONE_opacity_micromap":{
          "level":2,
          "mode":"4state",
          "micromaps":[
            "0b00101010000000001010101010101000",
            "0b00000000100010101010100000100000"
          ]
        }
      }
```

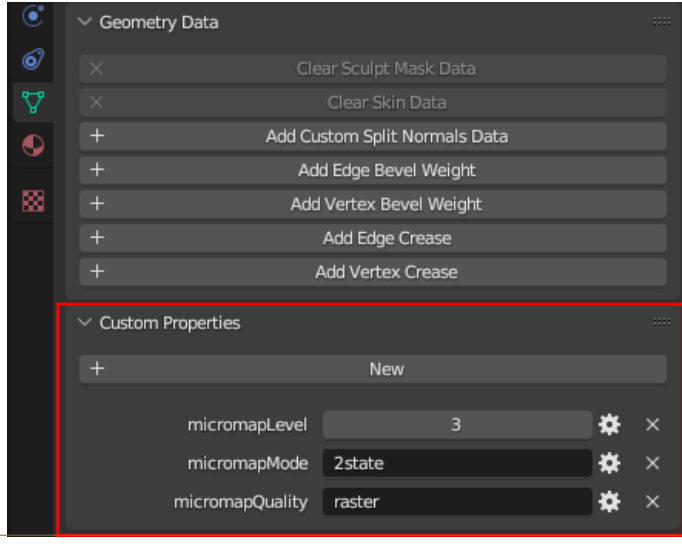


Opacity Micromaps Exporter



Opacity Micromaps Exporter

Control Attributes



Opacity Micromaps Exporter

```
for mesh in gltf_scene:
    for primitive in mesh:
        alpha_map = mesh.material.baseColorTexture
        if alpha_map:
            indices = gltf_access(primitive.indices)
            tex_coords = gltf_access(primitive.tex_coords)
            primitive.extension = micromaps_extension(...)
```

Opacity Micromaps Exporter

```
def gather_mesh_hook(gltf_mesh, bl_mesh, ...):
    for primitive in gltf_mesh:
        alpha_map = gltf_mesh.material.baseColorTexture
        if alpha_map:
            indices = gltf_access(primitive.indices)
            tex_coords = gltf_access(primitive.tex_coords)
            primitive.extension = micromaps_extension(...)
```

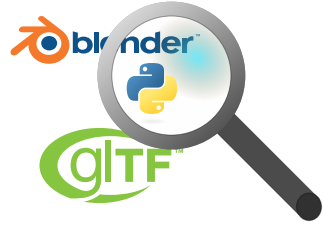
Multiprocessing

- No locals
- Avoid nested parallelism
- (Most) Arguments are *copied*
- Appropriate chunk-sizes

```
for work_item in chunk:
    run_task(work_item)

chk_sz = max(1, nitems // ncore)
```

Profiling Plugins

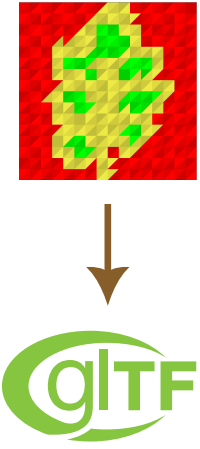
```
glTF2_pre_export_callback(export_settings)
glTF2_post_export_callback(export_settings)
```



Storing the Micromaps

JSON Storage

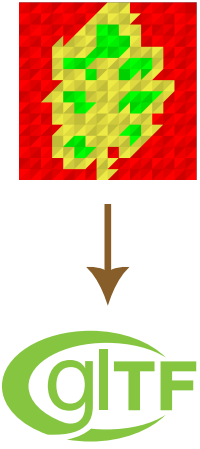
```
mm_strs = [str(mm) for mm in micromaps]
ext = {
    "level": micromaps.level,
    "mode": micromaps.mode,
    "format": "linear",
    "micromaps": mm_strs,
}
return Extension(extension=ext, ...)
```



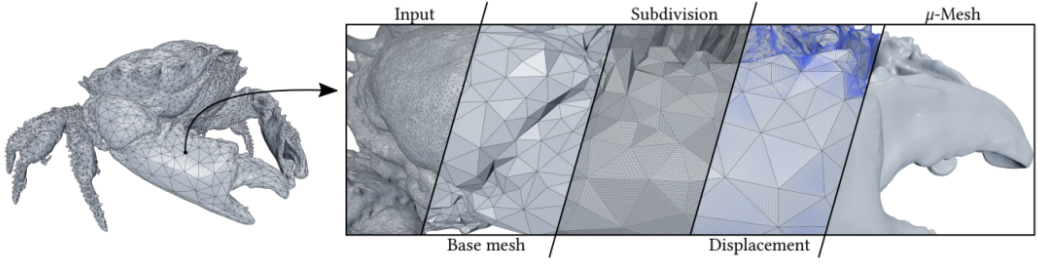
Storing the Micromaps

Buffer Storage

```
mm_bytes = pack_micromaps(micromaps)
ext = {
    "level": level,
    "mode": mode,
    "format": "buffer",
    "micromaps": BinaryData(mm_bytes),
}
return Extension(extension=ext, ...)
```



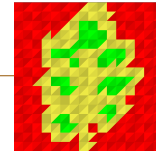
Displacement Micromaps / Other Micromaps



A. Maggiordomo, H. Moreton, and M. Tarini, "Micro-mesh construction," *ACM Trans. Graph.*, vol. 42, no. 4, Jul. 2023, ISSN: 0730-0301. DOI: 10.1145/3592440. [Online]. Available: https://doi.org/10.1145/3592440

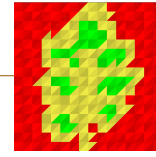
- `VK_NV_displacement_micromap`
- `NV_attribute_micromap`
- `NV_displacement_micromap`
- `NV_micromaps`
- `NV_micromap_tooling`
- `NV_opacity_micromap`

Importer Plugins

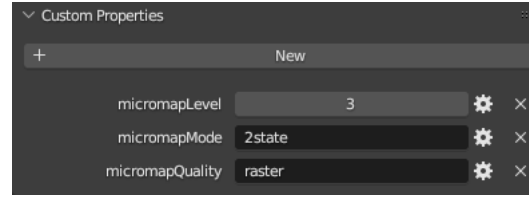



```
class glTF2ImportUserExtension:

    def __init__(self):
        from io_scene_gltf2.io.com.gltf2_io_extensions import Extension
        self.properties = bpy.context.scene.OmmImporterExtensionProperties
        self.extensions = [Extension(name="NONE_opacity_micromap",
                                     extension={},
                                     required=False)]

    def gather_import_mesh_after_hook(self, gltf2_mesh, blender_mesh, gltf):
        if self.properties.enabled:
            create_micromap_attributes(gltf2_mesh, blender_mesh)
```

Importer Plugins




```
mode, level, quality = find_attributes(gltf_mesh)
if level:
    bl_mesh["micromapLevel"] = level
if mode:
    bl_mesh["micromapMode"] = mode
if quality:
    bl_mesh["micromapQuality"] = quality
```



Conclusion and Summary

- What is `glTF`?
- The `glTF` Addon
- Creating Plugins
  - Watermarking
  - Micromaps
- Tips and Tricks
  - Profiling
  - Multiprocessing

Thanks for Listening

Questions

- Thanks for listening!
- Questions and Answers

The End