Université de Limoges

FACULTÉ DES SCIENCES ET TECHNIQUES

master ISICG limoges

INDIGO PACKSHOTER v1.0

Olivier Dupont [o.dupont186@gmail.com]

10/01/2022

Table of contents

I. Indigo Packshoter

II. Target public

III. User workflow

IV. Upkeeping

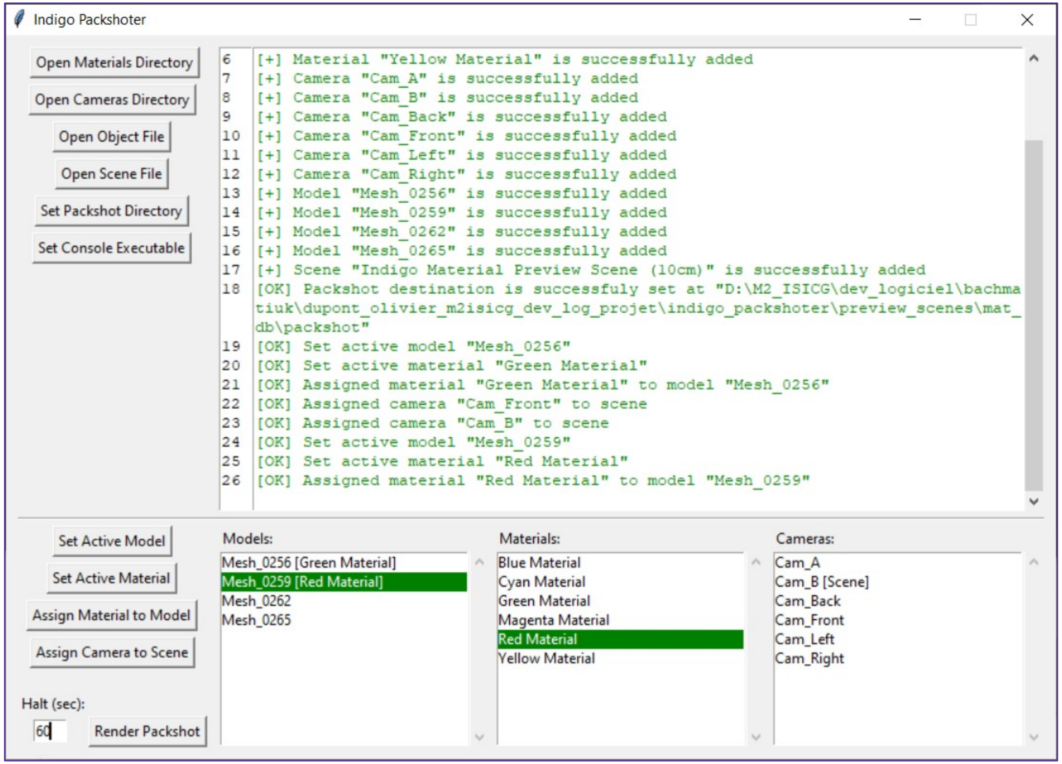
V. Implementation planning

I. Indigo Packshoter

This simple tool allows to automatically render images of an object in a Indigo Renderer scene by firstly injecting materials from a material database into a Indigo Renderer object.

A Packshot is basically an image.

Indigo Renderer is a 3D rendering software that allows to create photo-realistic images.



```
[+] Material "Yellow Material" is successfully added
[+] Camera "Cam_A" is successfully added
[+] Camera "Cam_B" is successfully added
[+] Camera "Cam_Back" is successfully added
[+] Camera "Cam_Front" is successfully added
[+] Camera "Cam_Left" is successfully added
[+] Camera "Cam_Right" is successfully added
[+] Model "Mesh_0256" is successfully added
[+] Model "Mesh_0259" is successfully added
[+] Model "Mesh_0262" is successfully added
[+] Model "Mesh_0265" is successfully added
[+] Scene "Indigo Material Preview Scene (10cm)" is successfully added
[OK] Packshot destination is successfuly set at "D:\M2_ISICG\dev_logiciel\bachmatiuk\dupont_olivier_m2isicg_dev_log_projet\indigo_packshoter\preview_scenes\mat_db\packshot"
[OK] Set active model "Mesh_0256"
[OK] Set active material "Green Material"
[OK] Assigned material "Green Material" to model "Mesh_0256"
[OK] Assigned camera "Cam_Front" to scene
[OK] Assigned camera "Cam_B" to scene
[OK] Set active model "Mesh_0259"
[OK] Set active material "Red Material"
[OK] Assigned material "Red Material" to model "Mesh_0259"
```

Models:

Mesh_0256 [Green Material]
Mesh_0259 [Red Material]
Mesh_0262
Mesh_0265

Materials:

Blue Material
Cyan Material
Green Material
Magenta Material
Red Material
Yellow Material

Cameras:

Cam_A
Cam_B [Scene]
Cam_Back
Cam_Front
Cam_Left
Cam_Right

Halt (sec): 60

II. Target public

The main targets of such tool are quite various. The users are definitely linked to computer graphics domain.

- Toys market

- Online bank of models

- etc.

In fact, fields where Packshots are needed, in a commercial aspect in general.

Packshots goals are to showcase a finished product through high quality photographies in order to put it in catalogues, websites or else.

III. User workflow: Prerequisites

The user is provided with a Windows executable, a preview database of materials files, and some cameras, object and scene files.

The executable allows the user to easily run the application.

The user has to install Indigo Renderer, in order to render images.

OS Windows

Program EXE

Material files DB (.igm)

Cameras, object and scene files (.igs)

A version of Indigo Renderer installed

III. User workflow: How To

Open a file system dialog box to choose the Indigo material DB directory

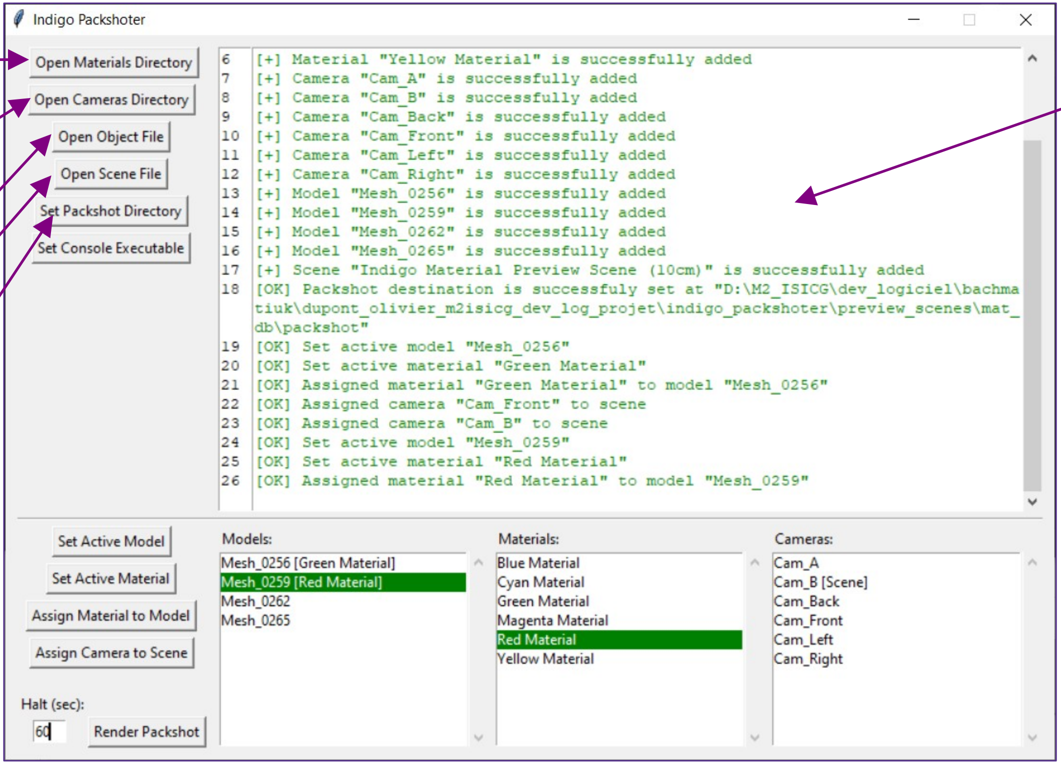
Open a file system dialog box to choose the Indigo cameras directory

Open a file system dialog box to choose the Indigo object file

Open a file system dialog box to choose the Indigo scene file

Open a file system dialog box to choose the Packshot destination

Logs of all operations



```
[+] Material "Yellow Material" is successfully added
[+] Camera "Cam_A" is successfully added
[+] Camera "Cam_B" is successfully added
[+] Camera "Cam_Back" is successfully added
[+] Camera "Cam_Front" is successfully added
[+] Camera "Cam_Left" is successfully added
[+] Camera "Cam_Right" is successfully added
[+] Model "Mesh_0256" is successfully added
[+] Model "Mesh_0259" is successfully added
[+] Model "Mesh_0262" is successfully added
[+] Model "Mesh_0265" is successfully added
[+] Scene "Indigo Material Preview Scene (10cm)" is successfully added
[OK] Packshot destination is successfuly set at "D:\M2_ISICG\dev_logiciel\bachma
tiuk\dupont_olivier_m2isicg_dev_log_projet\indigo_packshoter\preview_scenes\mat_
db\packshot"
[OK] Set active model "Mesh_0256"
[OK] Set active material "Green Material"
[OK] Assigned material "Green Material" to model "Mesh_0256"
[OK] Assigned camera "Cam_Front" to scene
[OK] Assigned camera "Cam_B" to scene
[OK] Set active model "Mesh_0259"
[OK] Set active material "Red Material"
[OK] Assigned material "Red Material" to model "Mesh_0259"
```

Models:
- Mesh_0256 [Green Material]
- Mesh_0259 [Red Material]
- Mesh_0262
- Mesh_0265

Materials:
- Blue Material
- Cyan Material
- Green Material
- Magenta Material
- Red Material
- Yellow Material

Cameras:
- Cam_A
- Cam_B [Scene]
- Cam_Back
- Cam_Front
- Cam_Left
- Cam_Right

Halt (sec): 60

III. User workflow: How To (2)

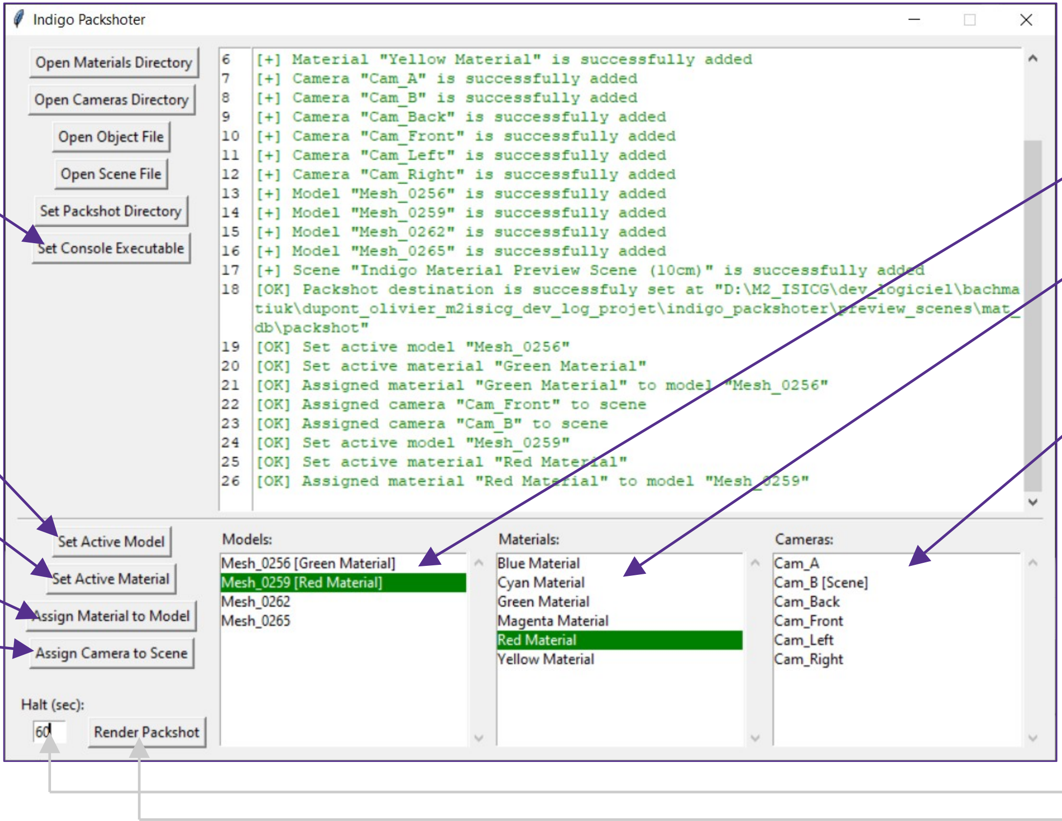
Open a file system dialog box to choose the Indigo Renderer executable for last step of render

Set active (in green) the selected model

Set active (in green) the selected material

Assign an active material to an active model

Assign a selected camera to the scene



```
[+] Material "Yellow Material" is successfully added
[+] Camera "Cam_A" is successfully added
[+] Camera "Cam_B" is successfully added
[+] Camera "Cam_Back" is successfully added
[+] Camera "Cam_Front" is successfully added
[+] Camera "Cam_Left" is successfully added
[+] Camera "Cam_Right" is successfully added
[+] Model "Mesh_0256" is successfully added
[+] Model "Mesh_0259" is successfully added
[+] Model "Mesh_0262" is successfully added
[+] Model "Mesh_0265" is successfully added
[+] Scene "Indigo Material Preview Scene (10cm)" is successfully added
[OK] Packshot destination is successfuly set at "D:\M2_ISICG\dev_logiciel\bachma
tiuk\dupont_olivier_m2isicg_dev_log_projet\indigo_packshoter\preview_scenes\mat_
db\packshot"
[OK] Set active model "Mesh_0256"
[OK] Set active material "Green Material"
[OK] Assigned material "Green Material" to model "Mesh_0256"
[OK] Assigned camera "Cam_Front" to scene
[OK] Assigned camera "Cam_B" to scene
[OK] Set active model "Mesh_0259"
[OK] Set active material "Red Material"
[OK] Assigned material "Red Material" to model "Mesh_0259"
```

Models:
- Mesh_0256 [Green Material]
- Mesh_0259 [Red Material]
- Mesh_0262
- Mesh_0265

Materials:
- Blue Material
- Cyan Material
- Green Material
- Magenta Material
- Red Material
- Yellow Material

Cameras:
- Cam_A
- Cam_B [Scene]
- Cam_Back
- Cam_Front
- Cam_Left
- Cam_Right

Halt (sec): 60

List of Indigo object models

List of Indigo materials from the DB

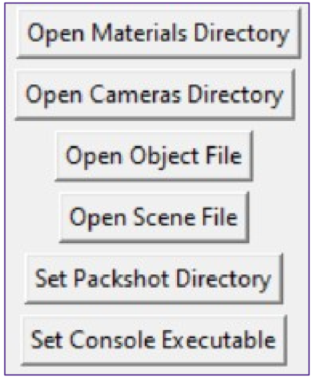
List of Indigo cameras

Number in seconds of time spend in rendering

Render an image with an assigned camera

III. User workflow: How To (3)

Step 1

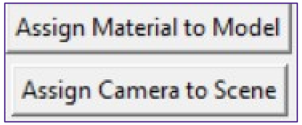


Open all necessary contents

Set the Packshot destination

Open the Indigo Renderer console .exe

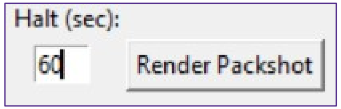
Step 2



Assign to each model a material

Assign a camera to the scene

Step 3


Halt (sec): 60

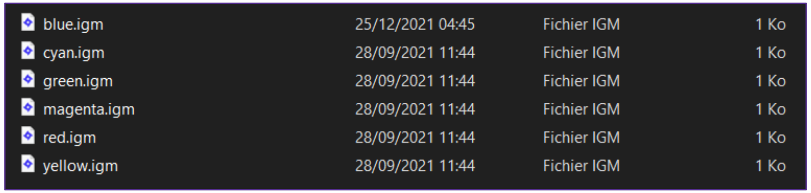
Set a Halt (limited to 9999s)

Finally render

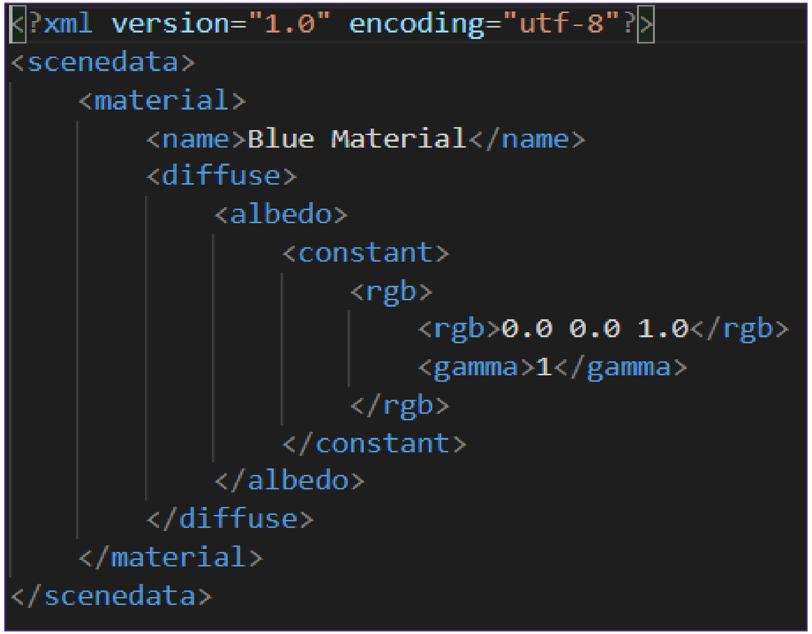
III. User workflow: Provided material database

The material database is basically a directory that contains one or multiples Indigo materials (.igm).

A Indigo material file is a XML file. The <material> tag must be encapsulated in a <scenedata> tag to be detected by the application.



blue.igm	25/12/2021 04:45	Fichier IGM	1 Ko
cyan.igm	28/09/2021 11:44	Fichier IGM	1 Ko
green.igm	28/09/2021 11:44	Fichier IGM	1 Ko
magenta.igm	28/09/2021 11:44	Fichier IGM	1 Ko
red.igm	28/09/2021 11:44	Fichier IGM	1 Ko
yellow.igm	28/09/2021 11:44	Fichier IGM	1 Ko



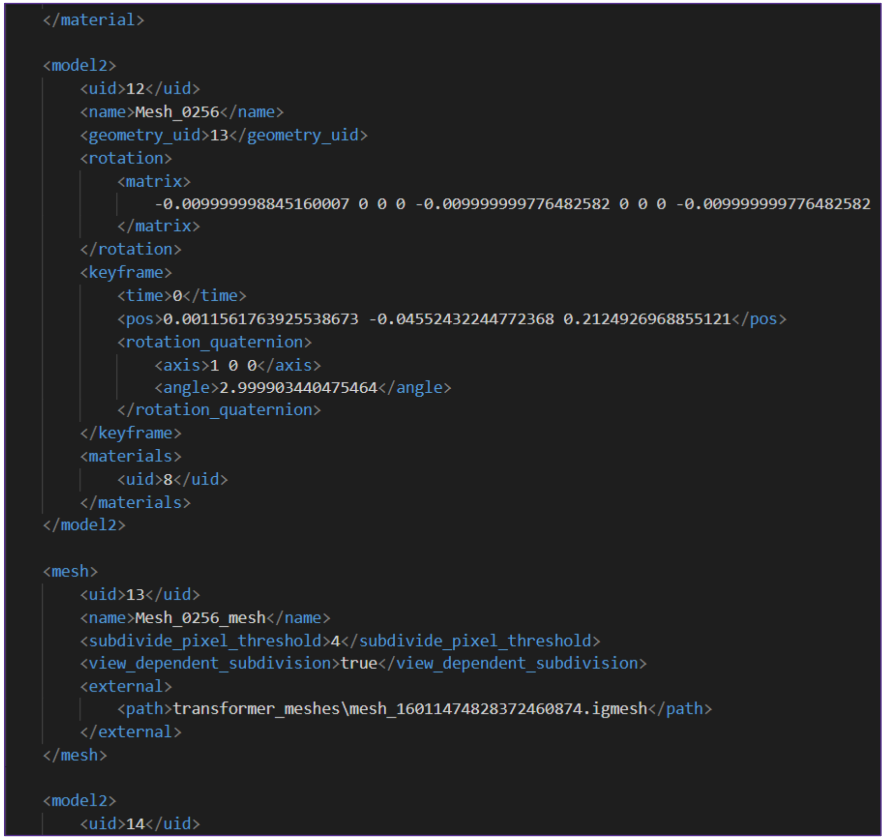
```xml
<?xml version="1.0" encoding="utf-8"?>
<scenedata>
    <material>
        <name>Blue Material</name>
        <diffuse>
            <albedo>
                <constant>
                    <rgb>
                        <rgb>0.0 0.0 1.0</rgb>
                        <gamma>1</gamma>
                    </rgb>
                </constant>
            </albedo>
        </diffuse>
    </material>
</scenedata>
```

III. User workflow: Provided others files

The others files provided are Indigo cameras, Indigo object and Indigo scene, all in the .igs extension.

Like Indigo materials, Indigo cameras, Indigo object and Indigo scene files are XML files. All tags must also be encapsulated in a <scenedata> tag to be detected by the application. The exception is for the Indigo scene which must have all being encapsulated in a <scene> tag.

Theses files can be generated from Indigo Renderer.

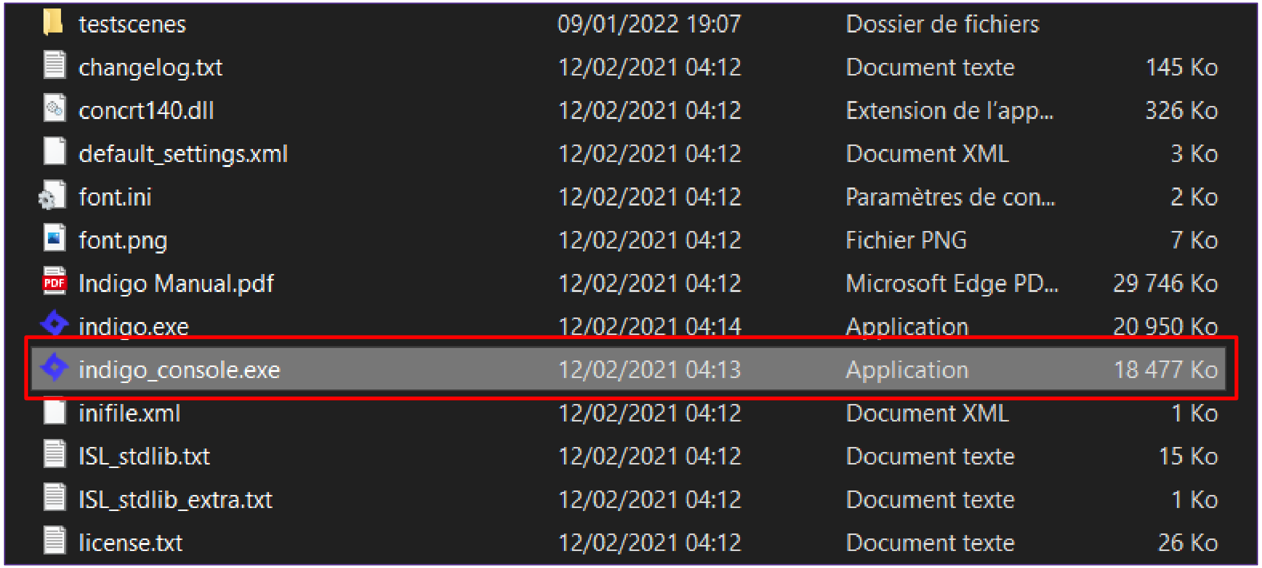


```xml
</material>

<model2>
    <uid>12</uid>
    <name>Mesh_0256</name>
    <geometry_uid>13</geometry_uid>
    <rotation>
        <matrix>
            -0.009999998845160007 0 0 0 -0.009999999776482582 0 0 0 -0.009999999776482582
        </matrix>
    </rotation>
    <keyframe>
        <time>0</time>
        <pos>0.0011561763925538673 -0.04552432244772368 0.2124926968855121</pos>
        <rotation_quaternion>
            <axis>1 0 0</axis>
            <angle>2.999903440475464</angle>
        </rotation_quaternion>
    </keyframe>
    <materials>
        <uid>8</uid>
    </materials>
</model2>

<mesh>
    <uid>13</uid>
    <name>Mesh_0256_mesh</name>
    <subdivide_pixel_threshold>4</subdivide_pixel_threshold>
    <view_dependent_subdivision>true</view_dependent_subdivision>
    <external>
        <path>transformer_meshes\mesh_16011474828372460874.igmesh</path>
    </external>
</mesh>

<model2>
    <uid>14</uid>
```

III. User workflow: Indigo Render Console Executable

When installing Indigo Renderer, the user must has a **indigo_console.exe** in the installation repertory.

This executable must be given to the application in order to render images.

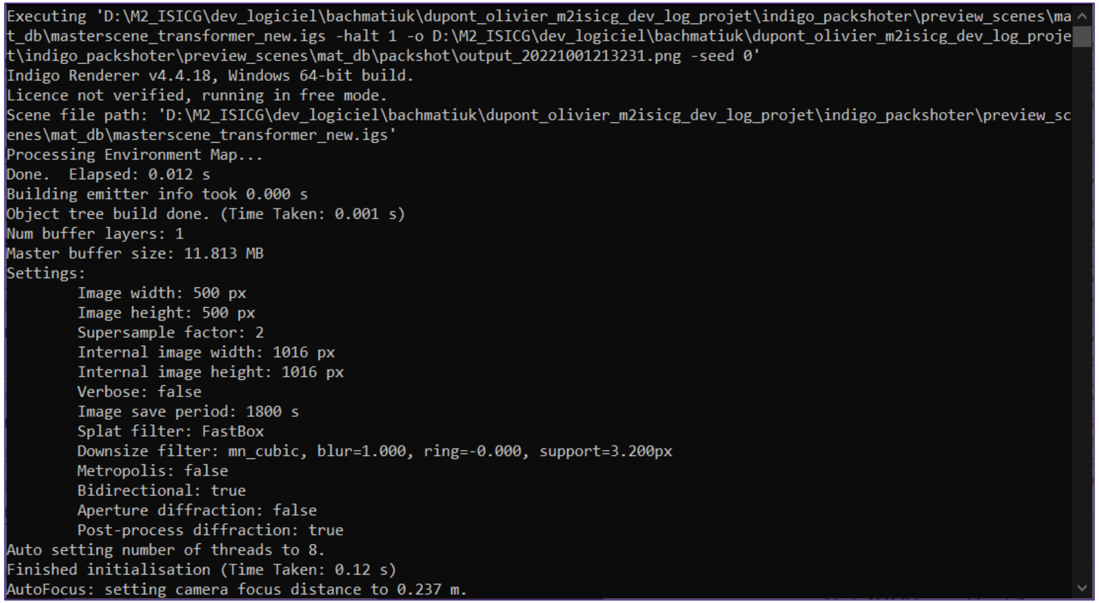


testscenes	09/01/2022 19:07	Dossier de fichiers	
changelog.txt	12/02/2021 04:12	Document texte	145 Ko
concrt140.dll	12/02/2021 04:12	Extension de l'app...	326 Ko
default_settings.xml	12/02/2021 04:12	Document XML	3 Ko
font.ini	12/02/2021 04:12	Paramètres de con...	2 Ko
font.png	12/02/2021 04:12	Fichier PNG	7 Ko
Indigo Manual.pdf	12/02/2021 04:12	Microsoft Edge PD...	29 746 Ko
indigo.exe	12/02/2021 04:14	Application	20 950 Ko
indigo_console.exe	12/02/2021 04:13	Application	18 477 Ko
inifile.xml	12/02/2021 04:12	Document XML	1 Ko
ISL_stdlib.txt	12/02/2021 04:12	Document texte	15 Ko
ISL_stdlib_extra.txt	12/02/2021 04:12	Document texte	1 Ko
license.txt	12/02/2021 04:12	Document texte	26 Ko

III. User workflow: Render terminal

When a render is started, a terminal is visible to inform the user about the process of rendering.

The render is finished when « Stopped. » is written in the terminal. Output image is sent to the Packshot directory previously set.

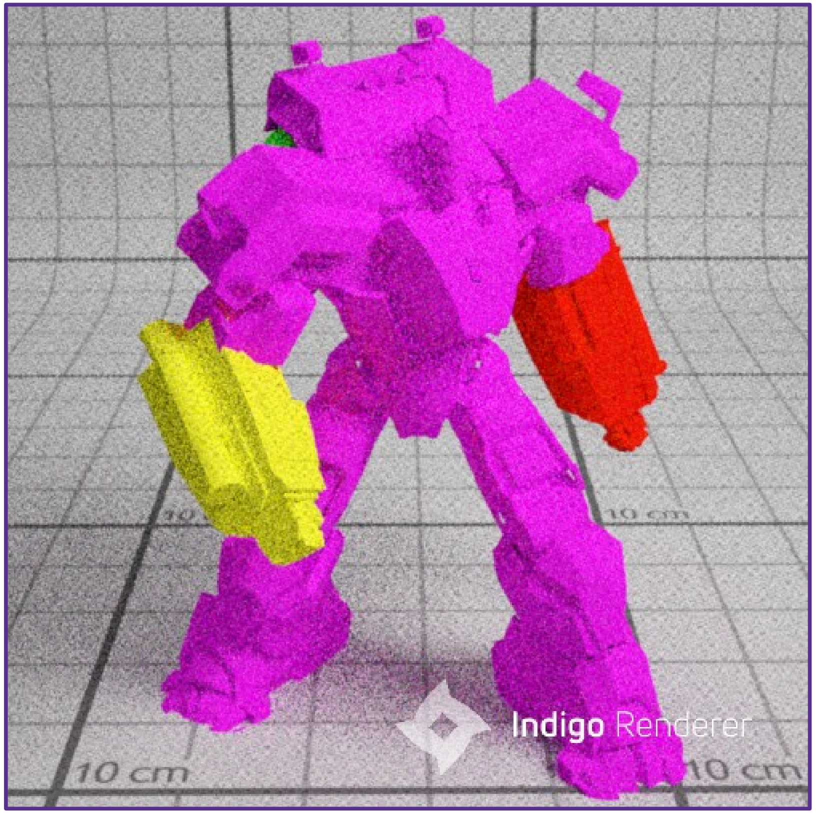


```
Executing 'D:\M2_ISICG\dev_logiciel\bachmatiuk\dupont_olivier_m2isicg_dev_log_projet\indigo_packshoter\preview_scenes\ma
t_db\masterscene_transformer_new.igs -halt 1 -o D:\M2_ISICG\dev_logiciel\bachmatiuk\dupont_olivier_m2isicg_dev_log_proje
t\indigo_packshoter\preview_scenes\mat_db\packshot\output_20221001213231.png -seed 0'
Indigo Renderer v4.4.18, Windows 64-bit build.
Licence not verified, running in free mode.
Scene file path: 'D:\M2_ISICG\dev_logiciel\bachmatiuk\dupont_olivier_m2isicg_dev_log_projet\indigo_packshoter\preview_sc
enes\mat_db\masterscene_transformer_new.igs'
Processing Environment Map...
Done.  Elapsed: 0.012 s
Building emitter info took 0.000 s
Object tree build done. (Time Taken: 0.001 s)
Num buffer layers: 1
Master buffer size: 11.813 MB
Settings:
        Image width: 500 px
        Image height: 500 px
        Supersample factor: 2
        Internal image width: 1016 px
        Internal image height: 1016 px
        Verbose: false
        Image save period: 1800 s
        Splat filter: FastBox
        Downsize filter: mn_cubic, blur=1.000, ring=-0.000, support=3.200px
        Metropolis: false
        Bidirectional: true
        Aperture diffraction: false
        Post-process diffraction: true
Auto setting number of threads to 8.
Finished initialisation (Time Taken: 0.12 s)
AutoFocus: setting camera focus distance to 0.237 m.
```

III. User workflow: Final result

By repeating rendering operation with same or others materials assigned and camera assigned, the user is able to build his Packshot.

Here’s an example of result of a Transformer object with differents materials assigned and a choosen camera:



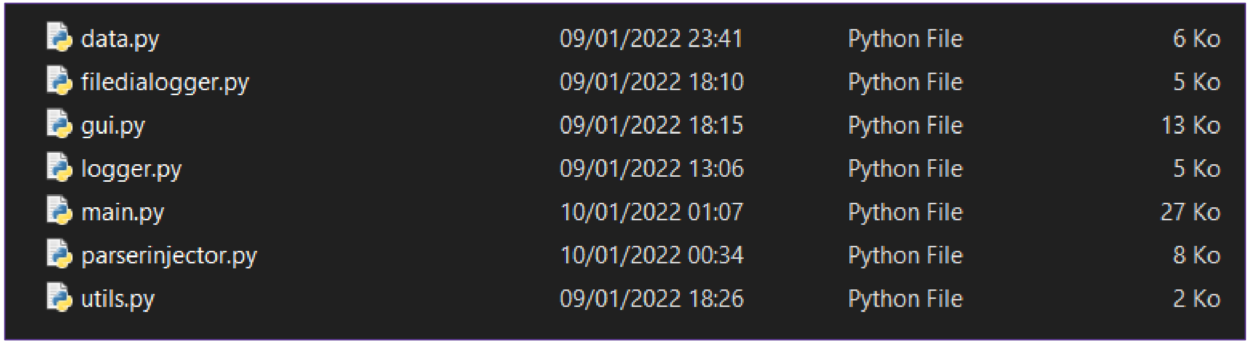
IV. Upkeeping

- Programming Language: Python 3.9
- Dependencies: (image on the bottom-right of comprehensive list of dependencies)
- External dependencies:

  - **xml.etree.ElementTree**: Python module to parse XML files

  - tkinter: Python module to build graphical interfaces

  - and Python standards modules

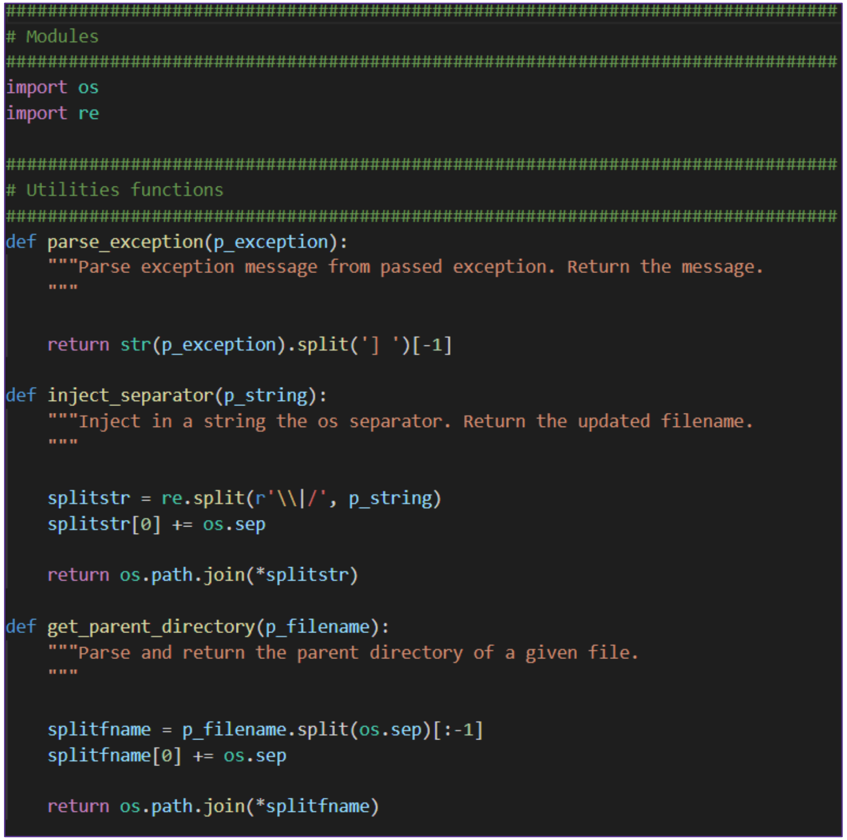


data.py	09/01/2022 23:41	Python File	6 Ko
filedialogger.py	09/01/2022 18:10	Python File	5 Ko
gui.py	09/01/2022 18:15	Python File	13 Ko
logger.py	09/01/2022 13:06	Python File	5 Ko
main.py	10/01/2022 01:07	Python File	27 Ko
parserinjector.py	10/01/2022 00:34	Python File	8 Ko
utils.py	09/01/2022 18:26	Python File	2 Ko

IV. Upkeeping: Utils

The file « utils.py » contains all utilities functions for the application.

Files paths are treated no matter OS is used.



```python
##################################################################################
# Modules
##################################################################################
import os
import re

##################################################################################
# Utilities functions
##################################################################################
def parse_exception(p_exception):
    """Parse exception message from passed exception. Return the message.
    """

    return str(p_exception).split('] ')[-1]

def inject_separator(p_string):
    """Inject in a string the os separator. Return the updated filename.
    """

    splitstr = re.split(r'\\|/', p_string)
    splitstr[0] += os.sep

    return os.path.join(*splitstr)

def get_parent_directory(p_filename):
    """Parse and return the parent directory of a given file.
    """

    splitfname = p_filename.split(os.sep)[:-1]
    splitfname[0] += os.sep

    return os.path.join(*splitfname)
```

IV. Upkeeping: Logger

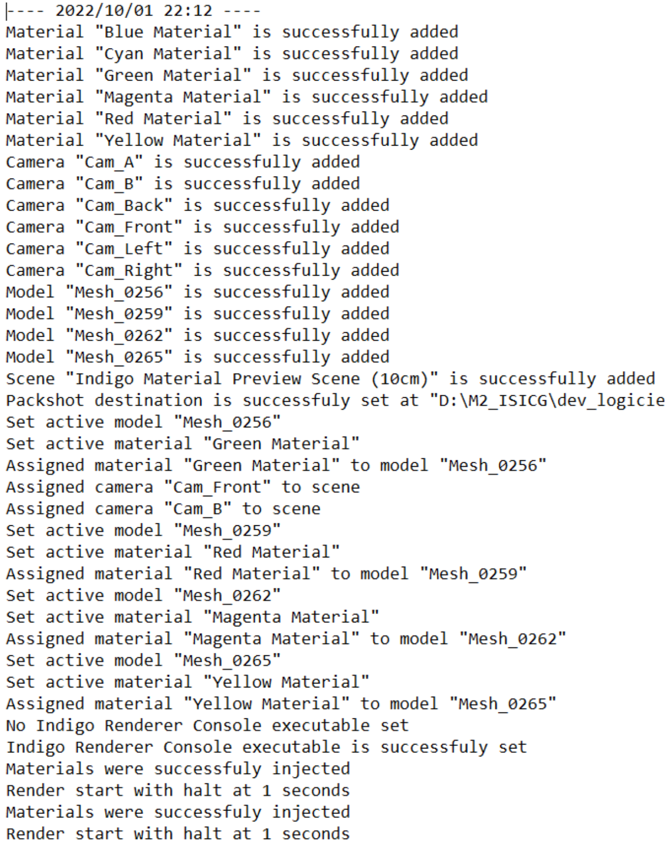
The file « logger.py » contains the entire definition of a **Logger**.

Logs are messages assigned with a type code **LogType** (**ERROR**, **WARNING**, **SUCCESS**, etc.) with allows to catch exceptions in order to treat them at any moment, or in the case of the application display them to the user for information on the multiples operations.

Loggers are basically stacks of logs.

When exited, the application is able to write logs in a file called « logs.txt », with proper date and time of last execution of the application, as shown on the right figure.

See documentation of classes for further information.



```
---- 2022/10/01 22:12 ----
Material "Blue Material" is successfully added
Material "Cyan Material" is successfully added
Material "Green Material" is successfully added
Material "Magenta Material" is successfully added
Material "Red Material" is successfully added
Material "Yellow Material" is successfully added
Camera "Cam_A" is successfully added
Camera "Cam_B" is successfully added
Camera "Cam_Back" is successfully added
Camera "Cam_Front" is successfully added
Camera "Cam_Left" is successfully added
Camera "Cam_Right" is successfully added
Model "Mesh_0256" is successfully added
Model "Mesh_0259" is successfully added
Model "Mesh_0262" is successfully added
Model "Mesh_0265" is successfully added
Scene "Indigo Material Preview Scene (10cm)" is successfully added
Packshot destination is successfuly set at "D:\M2_ISICG\dev_logicie
Set active model "Mesh_0256"
Set active material "Green Material"
Assigned material "Green Material" to model "Mesh_0256"
Assigned camera "Cam_Front" to scene
Assigned camera "Cam_B" to scene
Set active model "Mesh_0259"
Set active material "Red Material"
Assigned material "Red Material" to model "Mesh_0259"
Set active model "Mesh_0262"
Set active material "Magenta Material"
Assigned material "Magenta Material" to model "Mesh_0262"
Set active model "Mesh_0265"
Set active material "Yellow Material"
Assigned material "Yellow Material" to model "Mesh_0265"
No Indigo Renderer Console executable set
Indigo Renderer Console executable is successfuly set
Materials were successfuly injected
Render start with halt at 1 seconds
Materials were successfuly injected
Render start with halt at 1 seconds
```

IV. Upkeeping: FileDialogger

The file « filedialogger.py » contains the entire definition of a **FileDialogger**.

The **FileDialogger** is in charge of dialogging with the file system of the OS, and to control entries (right extension, empties directories, incorrect XML file, etc.).

See class documentation for further information.

IV. Upkeeping: Data

The file « data.py » contains all the definitions of data classes.

The base class is **XMLTree**, which contains a **Element** tree and parsed file name.

XMLTree has multiples derivated classes:

- **Material**: A **XMLTree** which contains a parsed Indigo material tree.

- **Camera**: A **XMLTree** which contains a parsed Indigo camera tree.

- **Model**: A **XMLTree** which contains a model tree from a parsed Indigo object, and an assigned **Material**.

- **Scene**: A **XMLTree** which contains a parsed Indigo scene tree, and an assigned **Camera**.

See documentation of classes for further information.

IV. Upkeeping: ParserInjector

The file « parserinjector.py » contains the entire definition of a **ParserInjector**.

The **ParserInjector** class is in charge of achieving all parsing and injecting operations of Indigo files (XML files).

See class documentation for further information.

IV. Upkeeping: GUI dependencies

The file «gui.py » contains all GUI dependencies classes:

- **GUILogger**:

A class that inherit from the **Logger** class. Its goal is to override the **log()** method in order to display logs to the user, within an attached text widget.

- **GUIFileDialogger**:

A class that inherit from the **FileDialogger** class. It is in charge of controlling Indigo files and directory entries from a file dialog box prompt to the user.

- **CustomListbox**:

A class that inherit from the **Listbox** Tkinter class. It is in charge of treating the consequences of user actions over models, materials and cameras listboxes (for example, set an active material in green, or display the association model-material in models listbox). It is also used to determine which are the actives material/model/camera.

See classes documentations for further information.

IV. Upkeeping: Application

The file «main.py » contains the application class and the main part of the GUI. This is entry point of the program.

The **Application** class is in charge of treating the user actions on buttons and others widgets through commands and bindings methods.

It is also in charge of initializing all the GUI widgets (buttons, listboxes, logs text, halt value entry, etc.).

The method **_command_render()** controls if all models are assigned with materials, and previously if scene/object/materials directory/cameras directory/Packshot destination/Indigo Renderer Console executable/halt value/ were properly given and set.

Finally, the method **_render_image()** is in charge of calling a subprocess in order to achieve the render of the scene through Indigo Renderer Console executable.

See class documentation for further information.

IV. Upkeeping: Possible improvements

- The vertical scrolling of the logs lines column and the logs region is not in common.
- It might be interesting to be able to assign complex materials (blending, PBR, etc.).
- The assignment of materials to models is kind of « boring » (need to click on Active buttons each time).
- The user has to be informed when the render has ended in the logs, not only by the terminal.
- It might be interesting to be able to catch messages displayed in the terminal in the logs.
- Finally, the user might want to not give the Indigo Renderer Console executable himself, or he might want to render multiples Packshots in once.

IV. Upkeeping: Sources

- Python Tkinter guide (don’t worry about WaybackMachine :P):

  https://web.archive.org/web/20190524140835/https://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

- Python ElementTree reference:

  https://docs.python.org/3/library/xml.etree.elementtree.html

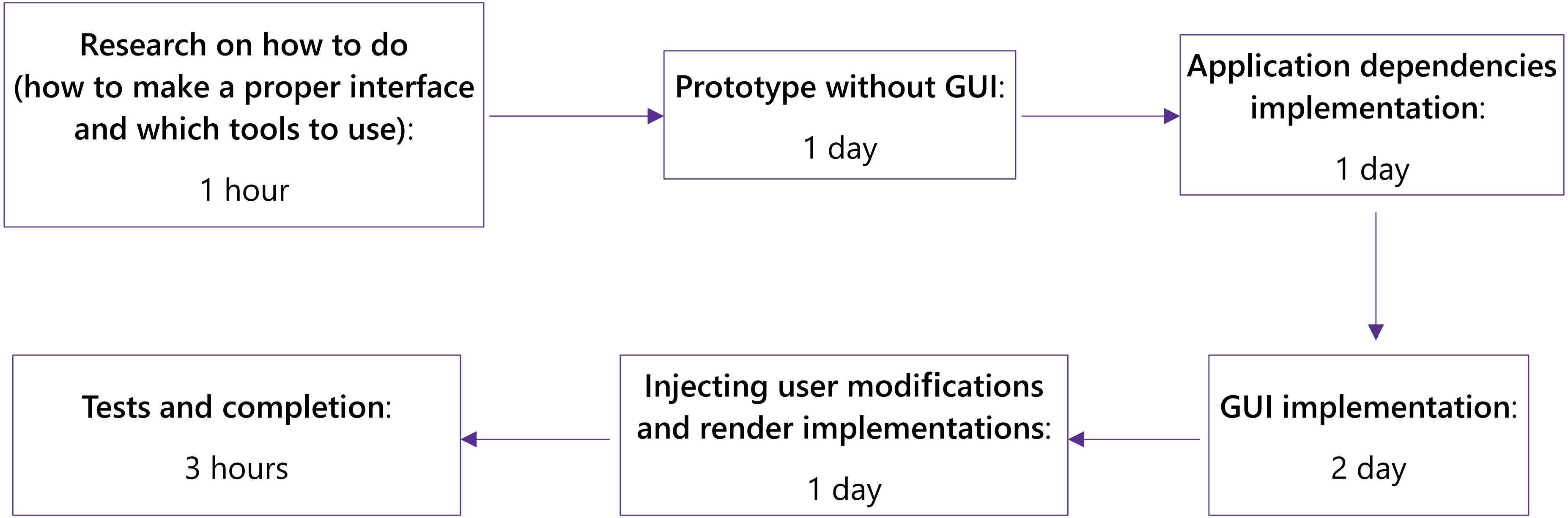
- Indigo Renderer files format technical reference:

  http://indigorenderer.github.io/

- Indigo Renderer command line reference and parameters:

  https://www.indigorenderer.com/indigo-technical-reference/command-line-parameter-reference

V. Implementation Planning



Research on how to do (how to make a proper interface and which tools to use):

1 hour

Prototype without GUI:

1 day

Application dependencies implementation:

1 day

GUI implementation:

2 day

Injecting user modifications and render implementations:

1 day

Tests and completion:

3 hours

** Note that time spend on the implementation is not sequential, a more or less rest period has been taken between each step, and more particulary between Prototype without GUI and Application dependencies implementation (at least a rest period of a week).*

Need to contact for further information?

Olivier Dupont
[o.dupont186@gmail.com]