02-first_xray_simulation

September 23, 2022

```
[1]: %matplotlib widget
```

Session 2

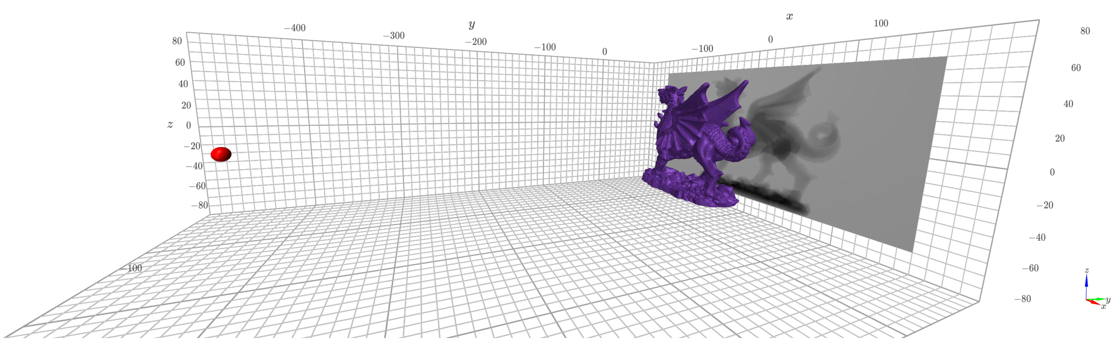
First X-ray radiograph simulations with

Author: Franck Vidal

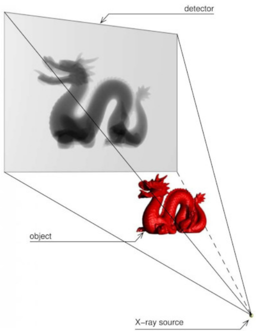
(version 1.0, 22 Sep 2022)

1 Aims of this session

- Create our first X-ray simulation, step-by-step;
- Save our X-ray image in a file format that preserves the original dynamic range;
- Visualise the results with 3 different look-up tables;
- Visualise the 3D environment.



2 Main steps



There are 6 main steps to simulate an X-ray image:

1. Create a renderer (OpenGL context);
2. Set the X-ray source;
3. Set the Spectrum;
4. Set the Detector;
5. Set the Sample; and
6. Compute the corresponding X-ray image.

3 Import packages

[2]:
```python
import os
import numpy as np # Who does not use Numpy?

import matplotlib # To plot images
import matplotlib.pyplot as plt # Plotting
from matplotlib.colors import LogNorm # Look up table
from matplotlib.colors import PowerNorm # Look up table

font = {'family' : 'serif',
         'size'   : 10
       }
matplotlib.rc('font', **font)

# Uncomment the line below to use LaTeX fonts
# matplotlib.rc('text', usetex=True)

from tifffile import imread, imwrite # Write TIFF files

import base64 # Save the visualisation

from gvxrPython3 import gvxr # Simulate X-ray images
from gvxrPython3.utils import saveProjections # Plot the X-ray image in linear,␣
 ↪log and power law scales
```

```python
from gvxrPython3.utils import compareWithGroundTruth # Plot the ground truth, the test image, and the relative error map in %
from gvxrPython3.utils import interactPlotPowerLaw # Plot the X-ray image using a Power law look-up table
from gvxrPython3.utils import visualise # Visualise the 3D environment if k3D is supported
```

```
Speckpy is not install, you won't be able to load a beam spectrum using Speckpy
SimpleGVXR 2.0.2 (2022-09-23T11:57:49) [Compiler: GNU g++] on Linux
gVirtualXRay core library (gvxr) 2.0.2 (2022-09-23T11:57:38) [Compiler: GNU g++]
on Linux
```

4 Create an OpenGL context

- Create an interactive window (available on Linux, MacOS, and Windows)
- Create a context without a window using EGL (available on Linux and MacOS, but not Windows)
  - use on desktop computers,
  - supercomputers, or
  - the cloud.

```cpp
//--------------------------------------------------------------------------------
/// Create an OpenGL context (the window won't be shown).
/**
 *  @param aWindowID:   the numerical ID of the context to create
 *                      (default value: -1, means that the ID will be
 *                      automatically generated)
 *  @param aRendererMajorVersion: Select the major version of the renderer.
 *                      (default value: 3)
 *  @param aRendererMajorVersion: Select the minor version of the renderer.
 *                      (default value: 2)
 */
//--------------------------------------------------------------------------------
void createOpenGLContext(int aWindowID = -1,
        int aRendererMajorVersion = 3,
        int aRendererMinorVersion = 2);

//--------------------------------------------------------------------------------
/// Create an OpenGL context and display it in a window
/**
 *  @param aWindowID:   the numerical ID of the context to create
 *                      (default value: -1, means that the ID will be
 *                      automatically generated)
 *  @param aRenderer: Select the renderer to use, e.g. OpenGL or Vulkan.
 *                      (default value: OPENGL)
 *  @param aRendererMajorVersion: Select the major version of the renderer.
 *                      (default value: 3)
 *  @param aRendererMajorVersion: Select the minor version of the renderer.
```

```
 *                     (default value: 2)
 *  @param aVisibilityFlag: flag controling if the window should be visible (1)
 *                           or hidden (0). (default value: 0)
 */
//--------------------------------------------------------------------------------
#ifndef __APPLE__
void createWindow(int aWindowID = -1,
        int aVisibilityFlag = 0,
        const std::string& aRenderer = "OPENGL",
        int aRendererMajorVersion = 3,
        int aRendererMinorVersion = 2);
#else // __APPLE__
void createWindow(int aWindowID = -1,
        int aVisibilityFlag = 0,
        const std::string& aRenderer = "OPENGL",
        int aRendererMajorVersion = 2,
        int aRendererMinorVersion = 1);
#endif // __APPLE__
```

[3]:
```python
print("Create an OpenGL context")

window_id = 0
opengl_major_version = 4
opengl_minor_version = 5

gvxr.createOpenGLContext(window_id, opengl_major_version, opengl_minor_version);

backend = "OPENGL"
visible = True

# gvxr.createWindow(window_id, visible, backend, opengl_major_version,␣
 ↪opengl_minor_version);

visible = False
# gvxr.createWindow(window_id, visible, backend, opengl_major_version,␣
 ↪opengl_minor_version);

backend = "EGL"
# visible has no effect with EGL
# gvxr.createWindow(window_id, visible, backend, opengl_major_version,␣
 ↪opengl_minor_version);
```

```
Create an OpenGL context

Fri Sep 23 13:21:08 2022 ---- Create window (ID: 0)
Fri Sep 23 13:21:08 2022 ---- Initialise GLFW
Fri Sep 23 13:21:08 2022 ---- Create an OpenGL window with a 4.5 context.
Fri Sep 23 13:21:08 2022 ---- Make the window's context current
```

```
Fri Sep 23 13:21:08 2022 ---- Initialise GLEW
Fri Sep 23 13:21:08 2022 ---- OpenGL vendor: NVIDIA Corporation
Fri Sep 23 13:21:08 2022 ---- OpenGL renderer: NVIDIA GeForce RTX 2080
Ti/PCIe/SSE2
Fri Sep 23 13:21:08 2022 ---- OpenGL version: 4.5.0 NVIDIA 515.48.07
Fri Sep 23 13:21:08 2022 ---- Use OpenGL 4.5.
Fri Sep 23 13:21:08 2022 ---- Initialise the X-ray renderer if needed and if
possible
```

5 Set the X-ray source

- Position
  - x: 0.0 cm,
  - y: -40.0 cm,
  - z: 0.0 cm.

```
gvxr.setSourcePosition(
    x, # float
    x, # float
    z, # float
    unit of length # string, e.g. "cm"
)
```

[4]:
```
gvxr.setSourcePosition(0, -40, 0, "cm")
```

- Shape:
  - Cone beam: `gvxr.usePointSource()`, or
  - Parallel (e.g. synchrotron): `gvxr.useParallelBeam();`

[5]:
```
gvxr.usePointSource()
```

6 Set the spectrum

- monochromatic (0.08 MeV, i.e. 80 keV),
- 1000 photons per ray.

```
gvxr.setMonoChromatic(
    energy, # float
    unit_of_energy, # string, e.g. "MeV"
    number of photons # float
)
```

[6]:
```
gvxr.setMonoChromatic(80, "keV", 1000)
```

7 Set the detector:

- Position
  - x: 0.0 cm,

- y: 10.0 cm,
- z: 0.0 cm.

```
gvxr.setDetectorPosition(
    x, # float
    x, # float
    z, # float
    unit of length # string, e.g. "cm"
)
```

[7]:
```
gvxr.setDetectorPosition(0, 10, 0, "cm")
```

- Orientation: (unit vector)
  - x: 0.0,
  - y: 0.0,
  - z: -1.0.

```
gvxr.setDetectorUpVector(
    x, # float
    x, # float
    z # float
)
```

[8]:
```
gvxr.setDetectorUpVector(0, 0, -1)
```

- Resolution:
  - width: 640 pixels,
  - height: 320 pixels.

```
gvxr.setDetectorNumberOfPixels(
    width, # unsigned int
    height # unsigned int
)
```

[9]:
```
gvxr.setDetectorNumberOfPixels(640, 320)
```

- Pixel spacing:
  - width: 0.5 mm,
  - height: 0.5 mm.

```
gvxr.setDetectorPixelSize(
    width, # float
    height # float
    unit of length # string, e.g. "mm"
)
```

[10]:
```
gvxr.setDetectorPixelSize(0.5, 0.5, "mm")
```

```
Fri Sep 23 13:21:09 2022 ---- Initialise the renderer
```

8 Set the sample

- Welsh dragon in a STL file:
  - ID: "Dragon",
  - fname: "input_data/welsh-dragon-small.stl",
  - Unit: mm.

```
gvxr.loadMeshFile(
    ID, # string
    fname, # string
    unit of length # string, e.g. "mm"
)
```

[11]:
```
gvxr.loadMeshFile("Dragon",  "../input_data/welsh-dragon-small.stl", "mm")
```

```
Fri Sep 23 13:21:10 2022 ---- file_name:        ../input_data/welsh-dragon-
small.stl    nb_faces:       457345  nb_vertices:    1372035 bounding_box (in
cm):   (-4.47065, -74.9368, 23.5909)   (2.37482, -59.4256, 36.0343)
```

- Rotate the sample by 90 degrees around the axis (0, 0, 1)
  - ID: "Dragon",
  - angle: 90 degrees
  - x: 0,
  - y: 0,
  - z: 1.

```
gvxr.rotateNode(
    ID, # string
    90, # float
    0, # float
    0, # float
    1 # float
)
```

[12]:
```
gvxr.rotateNode("Dragon",
    90,
    0,
    0,
    1);
```

9 Move the sample to the centre of the world

No ID is provided in case the sample is made of several components.

```
gvxr.moveToCentre()
```

or

```
gvxr.moveToCenter()
```

or if you prefer the American spelling

```
[13]: gvxr.moveToCentre()
```

- Material of the sample (ID = "Dragon"):
  - For a chemical element such as iron, you can use the Z number or symbol:
    - `gvxr.setElement("Dragon", 26)`, or
    - `gvxr.setElement("Dragon", "Fe")`
  - For a compound such as water, do not forget to specify the density:
    - `gvxr.setCompound("Dragon", "H2O")`
    - `gvxr.setDensity("Dragon", 1.0, "g/cm3")`
    - `gvxr.setDensity("Dragon", 1.0, "g.cm-3")`
  - For a mixture such as Titanium-Aluminum-Vanadium alloy, do not forget to specify the density:
    - `gvxr.setMixture("Dragon", "Ti90Al6V4")`
    - `gvxr.setMixture("Dragon", [22, 13, 23], [0.9, 0.06, 0.04])`
    - `gvxr.setMixture("Dragon", ["Ti", "Al", "V"], [0.9, 0.06, 0.04]) # Not yet implemented`
    - `gvxr.setDensity("Dragon", 4.43, "g/cm3")`
    - `gvxr.setDensity("Dragon", 4.43, "g.cm-3")`

In this example, we'll use Ti90Al6V4.

```
[14]: gvxr.setMixture("Dragon", "Ti90Al6V4")
      gvxr.setDensity("Dragon", 4.43, "g/cm3")
```

10 Compute the corresponding X-ray image

Get a 2D array

`xray_image = gvxr.computeXRayImage()`

or make sure it's a Numpy array in float32 with

`xray_image = np.array(gvxr.computeXRayImage()).astype(np.single)`

```
[15]: xray_image = np.array(gvxr.computeXRayImage()).astype(np.single)
```

11 Update the visualisation window

```
[16]: gvxr.displayScene()
```

12 Create the output directory if needed

```
[17]: if not os.path.exists("output_data"):
          os.mkdir("output_data")
```

13 Save the X-ray image in a TIFF file and store the data using single-precision floating-point numbers.

- using gVXR directly
  - fname = output_data/02-gvxr-save.tif

```
gvxr.saveLastXRayImage(
    fname # string
)
```

[18]:
```
gvxr.saveLastXRayImage("output_data/02-gvxr-save.tif")
```

- using the tifffile package
  - fname = output_data/02-tifffile-save.tif

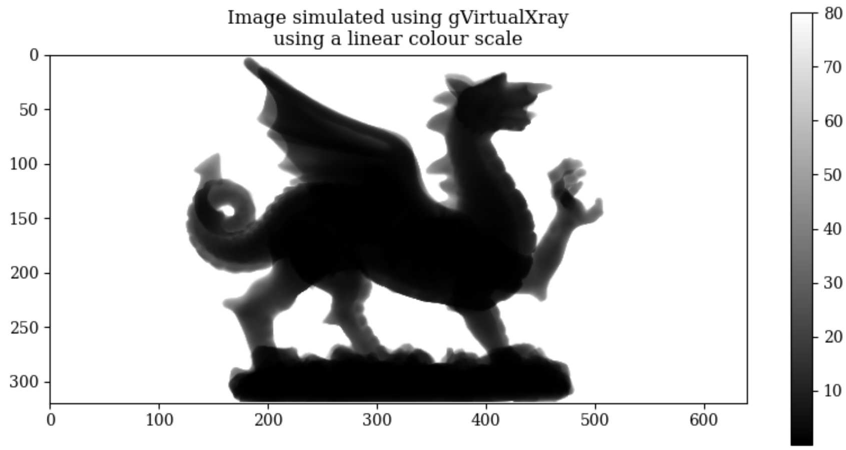
```
from tifffile import imwrite
imwrite(
    fname, # string
    xray_image # 2D array
)
```

[19]:
```
imwrite("output_data/02-tifffile-save.tif", xray_image)
```

14 Display the X-ray image

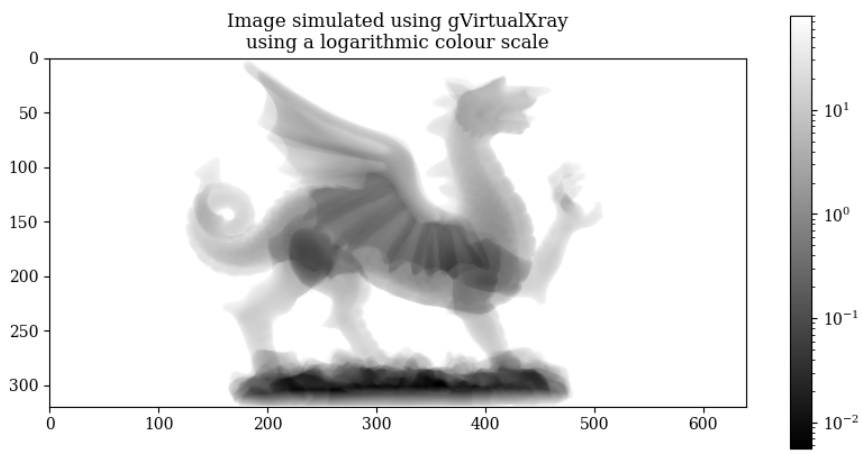
15 using a linear colour scale

[20]:
```
plt.figure(figsize=(10, 5))
plt.title("Image simulated using gVirtualXray\nusing a linear colour scale")
plt.imshow(xray_image, cmap="gray")
plt.colorbar(orientation='vertical');
plt.margins(0,0)
```



16 using a logarithmic colour scale

[21]:
```python
plt.figure(figsize=(10, 5))
plt.title("Image simulated using gVirtualXray\nusing a logarithmic colour scale")
plt.imshow(xray_image, cmap="gray", norm=LogNorm(vmin=xray_image.min(), vmax=xray_image.max()))
plt.colorbar(orientation='vertical');
plt.margins(0,0)
```



17 Using a Power-law colour scale

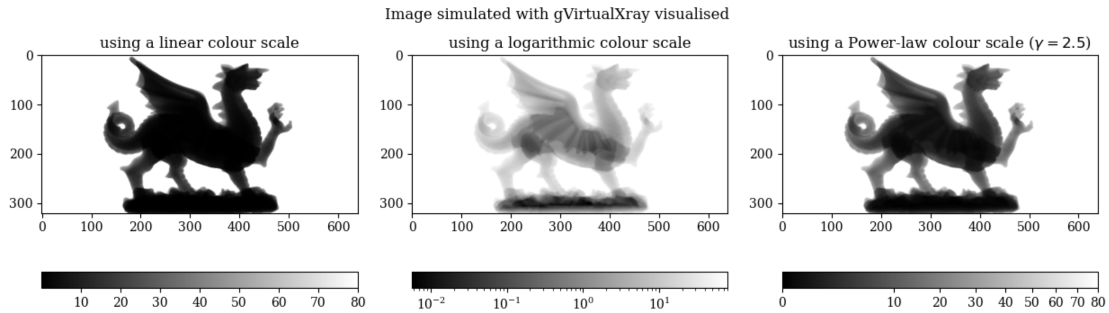
[22]:
```python
interactPlotPowerLaw(xray_image, gamma=1.5, figsize=(10, 5))
```

```
interactive(children=(FloatSlider(value=1.5, description='gamma', max=10.0, min=0.01, step=0.5), Output()), _d…
```

18 Display the X-ray image and compare three different lookup tables

Replace `???` below with the value of gamma that you selected above.

[23]:
```python
saveProjections(xray_image, "output_data/02-projections-dragon-TiAlV.pdf", gamma=2.5, figsize=(12.5, 5))
```



19 Get some image statistics using Numpy

and compare with the values I got. We should get the same, or at least something comparable.

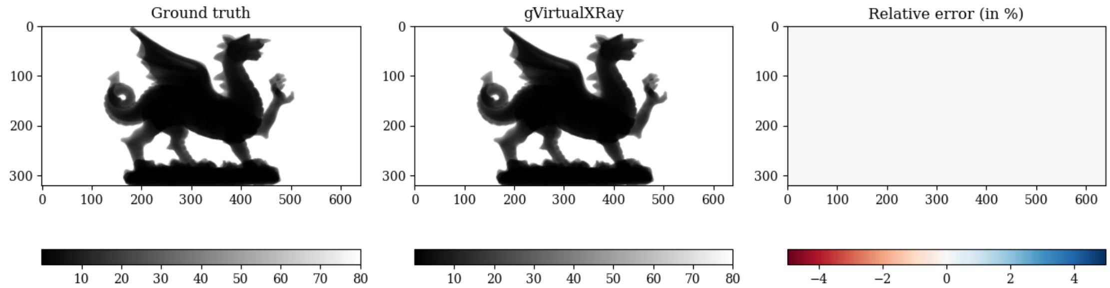
What?	Value (in keV)
Min pixel value:	0.0056294748
Mean pixel value:	56.367035
Median pixel value:	80.0
Stddev pixel value:	33.96409
Max pixel value:	80.0

[24]:
```
print("Min pixel value:", np.min(xray_image))
print("Mean pixel value:", np.mean(xray_image))
print("Median pixel value:", np.median(xray_image))
print("Stddev pixel value:", np.std(xray_image))
print("Max pixel value:", np.max(xray_image))
```

```
Min pixel value: 0.0056294748
Mean pixel value: 56.367035
Median pixel value: 80.0
Stddev pixel value: 33.96409
Max pixel value: 80.0
```

20 Compare with the ground truth

[25]:
```
ground_truth = imread("../input_data/02-dragon-TiAlV-groundtruth.tif")
compareWithGroundTruth(ground_truth, xray_image, figsize=(12.5, 5))
```



21 Change the sample's colour

By default the object is white, which is not always pretty. Let's change it to purple.

[26]:
```
red = 102 / 255
green = 51 / 255
blue = 153 / 255
gvxr.setColour("Dragon", red, green, blue, 1.0)
```

22 3D visualisation using k3D if possible

[27]:
```
plot=visualise(use_log=True)
plot.display()
```

```
Output()
```

[28]:
```
if plot is not None:
    plot.fetch_screenshot()

    data = base64.b64decode(plot.screenshot)
    with open("output_data/02-visualisation.png",'wb') as fp:
        fp.write(data)
```

23 Change the background colour to white

This image can be used in a research paper to illustrate the simulation environment.

[29]:
```
gvxr.setWindowBackGroundColour(1.0, 1.0, 1.0)
```

24 Update the visualisation window

[30]:
```
gvxr.displayScene()
```

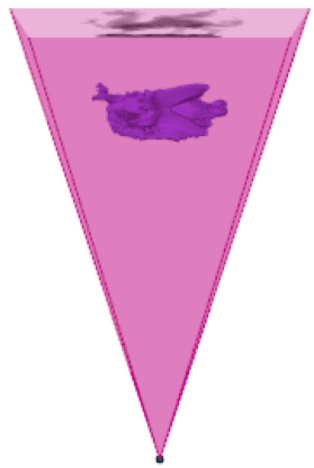
25 Take the screenshot and save it in a file

[31]:
```
screenshot = gvxr.takeScreenshot()
plt.imsave("output_data/02-screenshot.png", np.array(screenshot))
```

26 or display it using Matplotlib

[32]:
```
plt.figure(figsize=(10, 10))
plt.imshow(screenshot)
plt.title("Screenshot of the X-ray simulation environment")
plt.axis('off');
```

Screenshot of the X-ray simulation environment



27 Interactive visualisation

The user can rotate the 3D scene and zoom-in and -out in the visualisation window.

- Keys are:
  - Q/Escape: to quit the event loop (does not close the window)
  - B: display/hide the X-ray beam
  - W: display the polygon meshes in solid or wireframe
  - N: display the X-ray image in negative or positive
  - H: display/hide the X-ray detector
- Mouse interactions:
  - Zoom in/out: mouse wheel
  - Rotation: Right mouse button down + move cursor

Note: this function has no effect on supercomputers and on the cloud, but will work on a desktop or laptop computer.

[33]:
```
gvxr.renderLoop()
```

28 All done

[34]:
```
gvxr.terminate()
```

```
Fri Sep 23 13:21:53 2022 ---- Destroy all the windows
Fri Sep 23 13:21:53 2022 ---- Destroy window 0(0x563410b56a10)
```

[ ]:
```
```