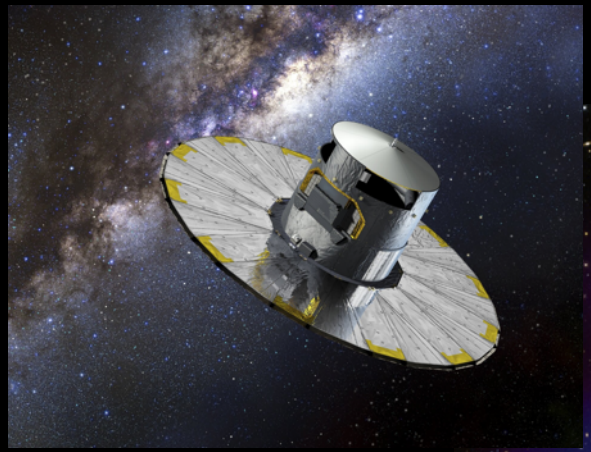
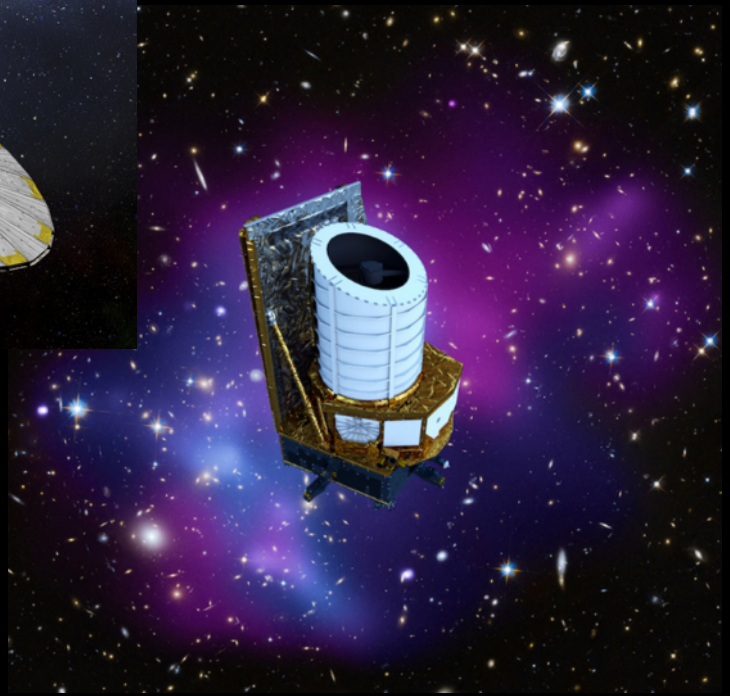
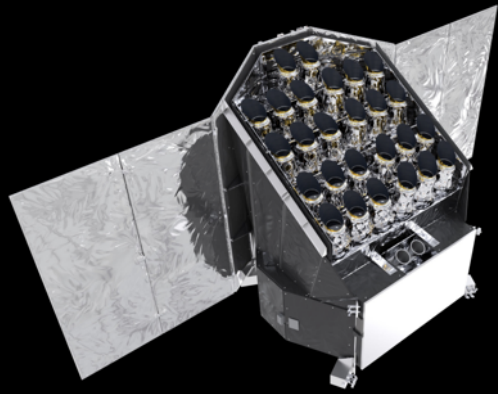
Pyxel: the collaborative detection simulation framework

Thibaut Prod'homme[a], Matej Arko[a], Frédéric Lemmel[a], Benoit Serra[b], Elizabeth George[b], Enrico Biancalani[c], Hans Smit[a], David Lucsanyi[d], Bradley Kelman[e]
[a]ESA, [b]ESO, [c]Leiden Observatory, [d]CERN, [e]OU/CEI

What is the common point between these ESA space missions?





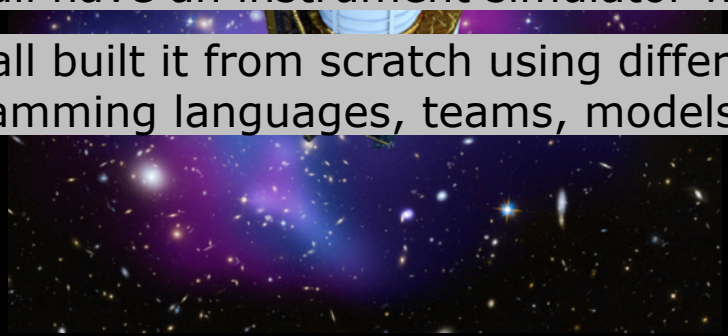


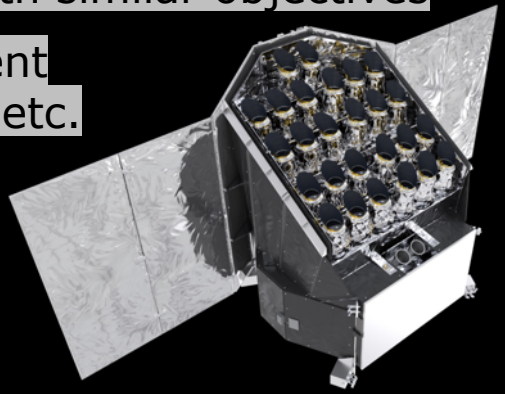
What is the common point between these ESA space missions?

Same instrument detector techno: CCDs

They all have an instrument simulator with similar objectives

They all built it from scratch using different programming languages, teams, models etc.



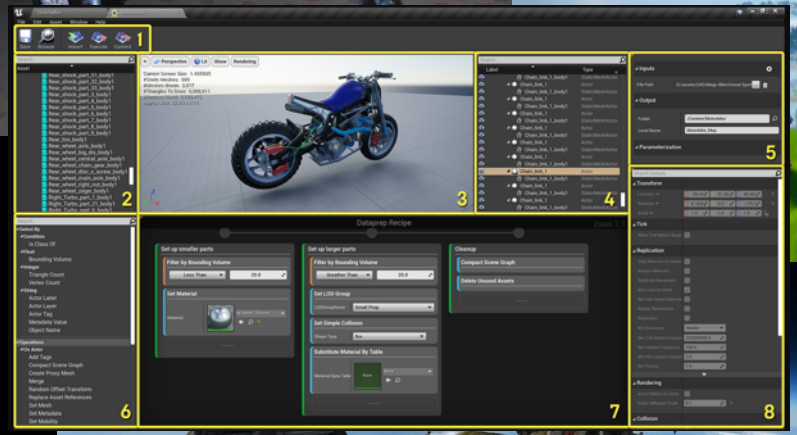


What is the common point between these video games?

What is the common point between these video games?

They are from different epochs, editors, styles but..

were built using the same framework: the UNREAL engine!



Unreal Engine

Unreal Engine is a top game engine developed by Epic Games since 1998. The latest engine is Unreal Engine 4 (UE4). Unreal Engine 5 is expected in 2021.

Unreal Engine is well-documented and easy-to-use. You can use it to develop any type of game — from console to mobile (including Android and iOS). It's even used in industries beyond game development, including automotive.

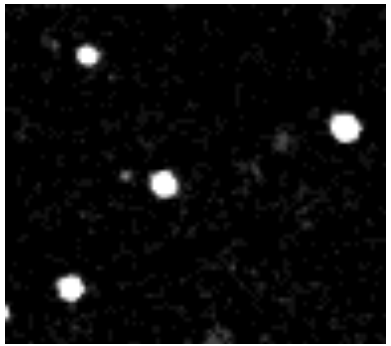
https://www.perforce.com/blog/vcs/most-popular-game-engines

Pyxel: the unreal engine behind 2020s instrument simulators?

- Pyxel is a detector simulation framework
- It simulates detector effects on images from optics down to readout electronics
- It can host and pipeline any detector effect models (CIS, CCD, MCT, MKID)
- Main objectives:
    1. **avoid duplication of work within and between institutes,**
    2. **provide a reliable, validated set of detector models,**
    3. **foster knowledge transfer in the instrumentation community**
- Opensource python package – soon to be released under MIT license
- Joint development effort between ESA and ESO
- First introduced to the community at SPIE 2018
- Beta released in 2019, now version 0.10, 70+ users -> growing community
- New features: Jupyter notebook interface, parallel computing on cluster/cloud, xarray output
- New models: persistency, IPC, cross-talk, CTI (ARCTIC) etc.

What is a “detection (chain) simulation”?

IN

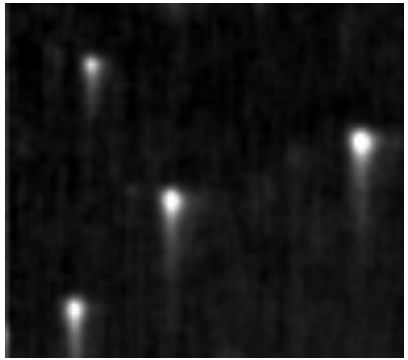


```
simulation:
  mode: parametric
  parametric:
    ...: mode: sequential
    ...:
      - key: pipeline.photon_generation.illumination.arguments.level
        values: numpy.unique(numpy.logspace(0, 6, 100, dtype=int))

outputs:
  output_folder: 'examples/PTC/outputs'
  parametric_plot:
    x: 'detector.image.mean'
    y: 'detector.image.std_deviation'
    plot_args:
      title: 'CCD Photon Transfer Curve'
      xscale: 'log'
      yscale: 'log'
      xlim: [1., 2.e+6]
      ylim: [1., 3.e+4]
      grid: true

detector: <5 keys>

pipeline: <8 keys>
```



OUT

Why simulating detection chains?

Instrument and detector simulations are needed each stage of a project

Technology and concept trade-off	Definition
Performance estimate	Design
Instrument parameter optimization	Design Consolidation
Performance verification	Manufacture & Testing
Data processing pipeline validation	Launch/First light
Instrument optimization and calibration	Commissioning
Performance verification	First science data
Calibrating out instrumental effect	Operation
Supporting data analysis	Scientific exploitation

Pyxel: motivations & principles

Every project develops from scratch its own instrument simulator..

Thousands of detector models are described in the literature >> Source code not always available/runnable >> cannot be pipelined

Sometimes several implementations >> Not always validated

Reusability

Do not reinvent the wheel

Spend time on what matters

Flexible

Multi-purpose

Python

Based on popular, open-source Python packages

Knowledge Transfer

Collaborative

User-oriented

Simple

Cross-platform

Python

Gitlab

Documentation

Reliability

Transparency

Readability

Open source

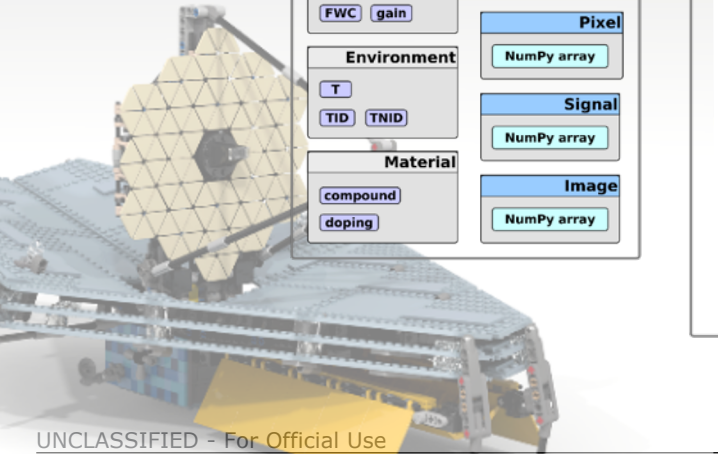
Unit test

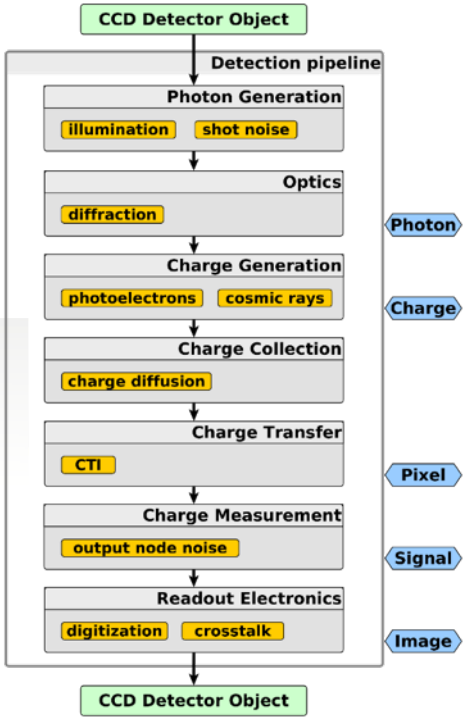
Integrated testing

Documentation

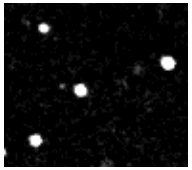
YAML

Pyxel architecture





Input photon distribution

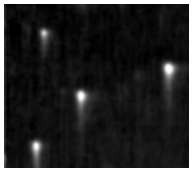


Pipeline: The **core algorithm**, hosting and running the **models**, which **are grouped** into different levels **imitating the working principles** of the detector/instrument

Detector object: A **bucket** containing all **properties and data** of the simulated instrument, which may be used or modified by the models

Configuration file: Pyxel's **user unique entry point**, based on YAML, structured, easy-to-read and understand

Output image



Configuration file

YAML> structured, easy to read and understand

Structured around 3 sections:

1. running mode (which also holds information about the outputs)
2. detector: environment, geometry, characteristics and material
3. detection pipeline: model groups and functions

```yaml
parametric:

  mode: sequential
  parameters:
    - key: pipeline.photon_generation.illumination.arguments.level
      values: numpy.unique(numpy.logspace(0, 6, 100, dtype=int))

  outputs:
    output_folder:  'outputs'

ccd_detector:

  geometry:

    row:    100                         # pixel
    col:    100                         # pixel
    total_thickness: 10.                # um
    pixel_vert_size: 10.                # um
    pixel_horz_size: 10.                # um

  material:
    material: 'silicon'

  environment:

  characteristics:
    qe: 0.5                 # -
    eta: 1.                 # e/photon
    sv: 1.e-6               # V/e
    amp: 0.8                # V/V
    a1: 100.                # V/V
    a2: 65536               # DN/V
    fwc: 100000             # e
    fwc_serial: 200000      # e
```

```yaml
pipeline:

  # -> photon
  photon_generation:
    - name: illumination
      func: pyxel.models.photon_generation.illumination
      enabled: true
      arguments:
          level: 0
    - name: shot_noise
      func: pyxel.models.photon_generation.shot_noise
      enabled: true

  # photon -> photon
  optics:

  # photon -> charge
  charge_generation:
    - name: photoelectrons
      func: pyxel.models.charge_generation.simple_conversion
      enabled: true

  # charge -> pixel
  charge_collection:
    - name: simple_collection
      func: pyxel.models.charge_collection.simple_collection
      enabled: true
    - name: fixed_pattern_noise
      func: pyxel.models.charge_collection.fix_pattern_noise
      enabled: true
      arguments:
        pixel_non_uniformity: data/pixel_non_uniformity.npy
    - name: full_well
      func: pyxel.models.charge_collection.simple_full_well
      enabled: true

  # pixel -> pixel
  charge_transfer:

  # pixel -> signal
  charge_measurement:
    - name: simple_measurement
      func: pyxel.models.charge_measurement.simple_measurement
      enabled: true
    - name: output_node_noise
      func: pyxel.models.charge_measurement.output_node_noise
      enabled: true
      arguments:
        std_deviation: 1.e-6   # Volt

  # signal -> image
  readout_electronics:
    - name: simple_amplifier
      func: pyxel.models.readout_electronics.simple_amplifier
      enabled: true
    - name: simple_processing
      func: pyxel.models.readout_electronics.simple_processing
      enabled: true
```

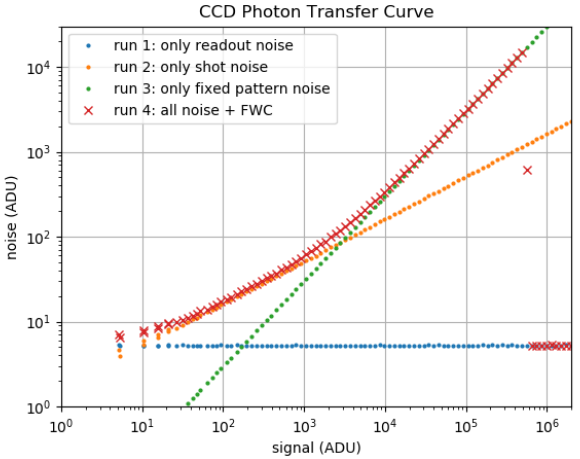
Pyxel 4x running modes

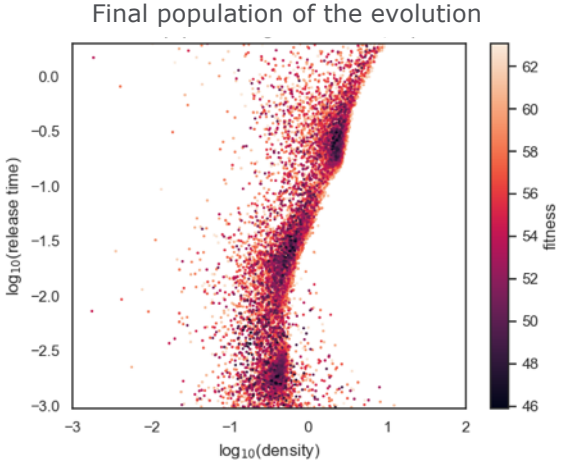
- **Single run**: one image in, one image out, single pipeline run > quick look/ health check

- **Parametric**: pipeline is run multiple times looping over a range of model or detector parameters > sensitivity analysis

- **Calibration**: optimize model or detector parameters to fit target data sets using a user-defined fitness function/figure of merit > model fitting, instrument optimization

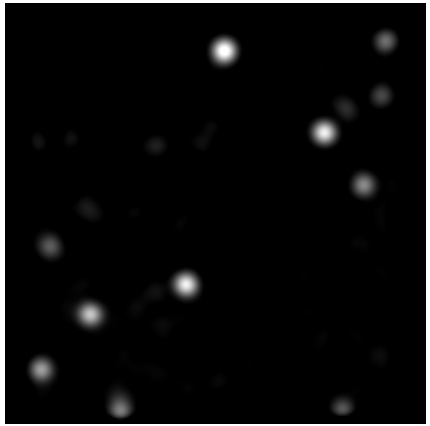
- **Dynamic**: the pipeline is run N times incrementing time a saving detector attributes > simulation of non-destructive readout mode, and time-dependent effects (e.g. persistence)





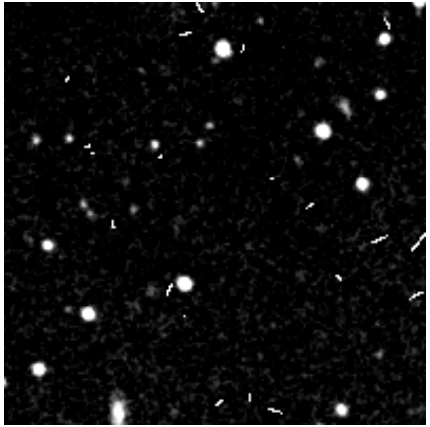
Models: examples of output images

POPPY[1]
(STScI, JWST)



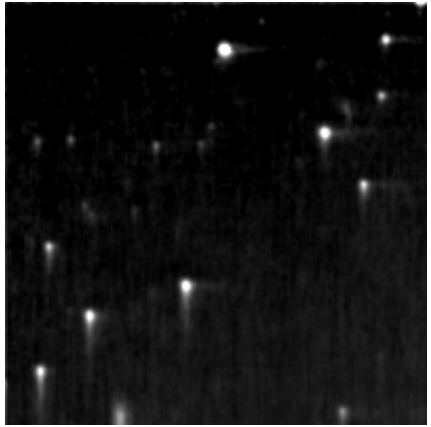
Optical PSF due to diffraction
(circular aperture)

CosmiX[2]
(ESA)



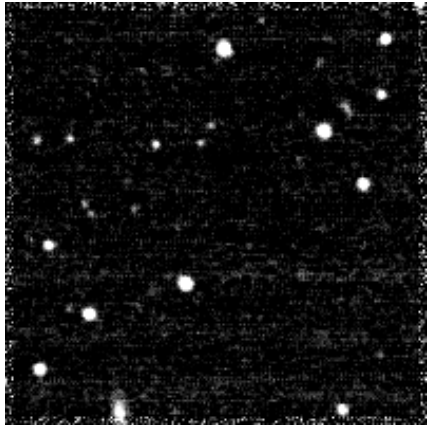
Cosmic ray tracks
(GCR protons at L2)

CDM[3]
(ESA, Gaia)



CCD Charge Transfer Inefficiency
(displacement damage in irradiated CCD)

ngHxRG[4]
(NASA, JWST)



CMOS pixel readout noises
(corr. & uncorr. pink, white, ACN, PCA0)

References:

[1] M. D. Perrin et al.: "Simulating point spread functions for the James Webb Space Telescope with WebbPSF", Space Telescopes and Instrumentation 2012, SPIE Proc., Vol. 8442, pp. 11. (2012).

[2] Lucsanyi, D. and Prod'homme, T., "Simulating Charge Deposition by Cosmic Rays Inside AstronomicalImaging Detectors,"IEEE Transactions on Nuclear Science67, 1623–1628 (July 2020)

[3] A. Short et al.: "An analytical model of radiation-induced Charge Transfer Inefficiency for CCD detectors", Monthly Notices of the Royal Astronomical Society 430(4), 3078{3085 (2013).

[4] B. J. Rauscher: "Teledyne H1RG, H2RG, and H4RG Noise Generator", Publications of the Astronomical Society of the Pacific 127(957), 1144 (2015).

Models: examples of output images

POPPY[1]
(STScI, JWST)

CosmiX[2]
(ESA)

CDM[3]
(ESA, Gaia)

ngHxRG[4]
(NASA, JWST)

For a complete list of models go to: https://esa.gitlab.io/pyxel/page/models/

Optical PSF due to diffraction
(circular aperture)

Cosmic ray tracks
(GCR protons at L2)

CCD Charge Transfer Inefficiency
(displacement damage in irradiated CCD)

CMOS pixel readout noises
(corr. & uncorr. pink, white, ACN, PCA0)

References:

[1] M. D. Perrin et al.: "Simulating point spread functions for the James Webb Space Telescope with WebbPSF", Space Telescopes and Instrumentation 2012, SPIE Proc., Vol. 8442, pp. 11. (2012).

[2] Lucsanyi, D. and Prod'homme, T., "Simulating Charge Deposition by Cosmic Rays Inside AstronomicalImaging Detectors,"IEEE Transactions on Nuclear Science67, 1623–1628 (July 2020)

[3] A. Short et al.: "An analytical model of radiation-induced Charge Transfer Inefficiency for CCD detectors", Monthly Notices of the Royal Astronomical Society 430(4), 3078{3085 (2013).

[4] B. J. Rauscher: "Teledyne H1RG, H2RG, and H4RG Noise Generator", Publications of the Astronomical Society of the Pacific 127(957), 1144 (2015).

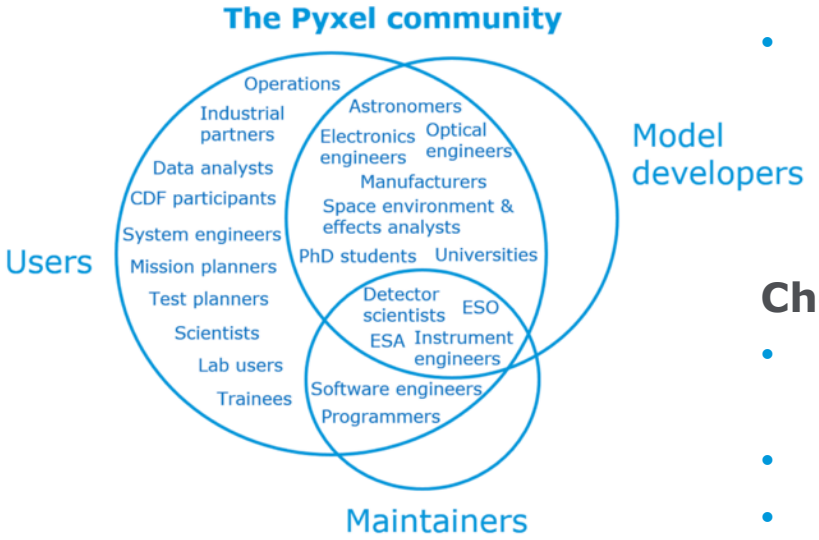
Want to try Pyxel?

- Remotely, without installing anything, by using binder

- Or locally, by following the install guide and downloading the examples from gitlab and run them using the command line or jupyter notebooks

- All links available @ esa.gitlab.io/pyxel/

Join the Pyxel collaboration!



Request membership: **pyxel@esa.int**

- Gitlab + mailing list + chat platform: where users can create/fix issues, reach out to the community for technical support and share experience, tips, hacks

Check website: esa.gitlab.io/pyxel

- blog to share the latest news, developments, and advertise job ads
- Pyxel's improved and comprehensive documentation
- detailed contribution guide to help more advanced users contributing to Pyxel via Gitlab e.g. improving documentation, adding new models, reporting bugs

Towards v1.0

Detector modelling is key to achieve the demanding objectives of the current and next generation of instruments onboard scientific space missions and ground-based experiments.

We developed Pyxel as a collaborative tool to provide a common framework to promote reusability, knowledge transfer, and reliability in the instrumentation community.

Pyxel has now reached v0.10: in beta but stable, contains many models and new features and it is becoming ever easier to use and contribute to.

Next milestone for Pyxel is version 1.0

Pyxel: the “unreal engine” behind 2020s instrument simulators and beyond!