Current *HST* Slitless Spectroscopy Simulation Software

STScI WFC3 Grism Group: G. Brammer, N. Pirzkal, R. E. Ryan

Abstract

We describe the properties of three independently-developed software packages used to simulate slitless spectroscopic data from the Hubble Space Telescope. The first, aXe/aXeSIM, was developed by the ST-ECF for ACS and later adapted for use with NICMOS and WFC3 data, and is officially supported by STScI for the HST user community. The other two packages were developed for personal use by N. Pirzkal and by G. Brammer. In this document we describe and compare the capabilities, key assumptions, outputs and runtime of the three separate software packages, which define the current state-of-the art for the analysis of space-based slitless spectroscopic data.

1. Grism data

Grism observations are obtained by inserting a dispersive element in the light path. In normal imaging mode, the detector provides a pixelized sampling of some scene:

$$I(x, y) = \int f(x, y, \lambda) \, \mathcal{S}(\lambda) \, d\lambda \tag{1}$$

where $I(x, y)$ refers to the direct image, $f(x, y, \lambda)$ is intrinsic spectrum at each pixel in the scene, and $\mathcal{S}(\lambda)$ is the sensitivity curve (which in principle depends on detector position as
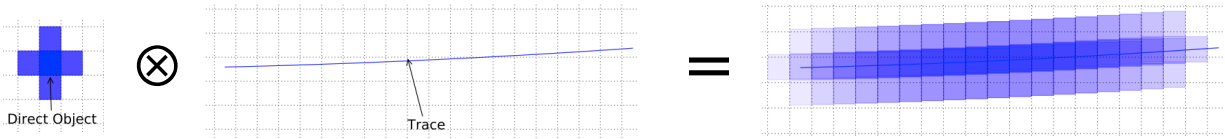
Fig. 1.— A simplified view of the relation between the information provided by the direct image (left), the transfer function of the dispersive element, the trace (middle), and the convolution of the two. If one ignores flat-fielding and wavelength dependence of the shape of the object, this can be considered (and simulated using) a 2D convolution.

well). The dispersive element transforms $f(x, y, \lambda)$ by redistributing the λ dimension. As described below, this transformation is field-dependent and can be encoded in a series of spatially-varying polynomials that have been calibrated for both WFC3 and ACS.

In the slitless case, the disperser defines a spectral trace, essentially mapping a given wavelength λ onto a unique x', y' position on the detector. Such a trace is shown in Figure 1. Assuming no change in sensitivity as a function of position, and ignoring pixel flat-fielding, a dispersed image can be thought of a the convolution of the direct image with the spectra trace:

$$G(x, y) = f(x, y, \lambda) \otimes tr(\lambda). \tag{2}$$

Slitless grism observations can therefore be simulated starting from a set of direct images, which provide information about the field and the objects within it, and sufficient understanding of the properties of the dispersing element. All of the existing analysis software we describe below currently rely on a set of direct observations of the field, or at least a catalog of objects, complete with their size and brightness.

2. Contamination

Contamination is a drawback of slitless spectroscopy. First, the lack of slits results in the subtle effect that the effective resolution of the disperser is lowered by the size of the footprint of the objects along the dispersion direction. Hence, a larger object will produce a spectrum that is smoothed out. This is visible in Figure 1 where we show that the leftmost and rightmost pixels of the source, shown on the left of the figure, are projected onto the same sets of pixels on the right-hand side of the Figure. This self contamination depends on the direction of the dispersion and orientation of the source. A non-symmetrical source will therefore produce spectra of different effective resolution at different position angles on the sky. Second, spectra of neighboring objects can contaminate one another and this is

again sensitive to the orientation at which the data are obtained. Figure 2 illustrates this by showing three distinct sources, shown in blue, red and green, dispersed at three position angles. At some orientations, some spectrum remains uncontaminated (i.e. green object in panels 1 and 2) while it is strongly contaminated at some other orientation (e.g., the green object in panel 3). As shown in Figure 2, keeping track of the contamination levels require a careful book keeping of the objects' location, their shape/footprint as well as their spectral characteristics.

3. Grism Analysis Software

There are currently three different simulation codes being used within the WFC3 Grism Group: `aXe/aXeSIM`, `threedhst`, and `NPSpec`. All three use the same WFC3 calibration products that describe the dispersion and sensitivity of the WFC3 grisms. `aXe/aXeSIM` can handle grism and prism (e.g., highly non-linear wavelength dispersion) dispersive elements. `aXe/aXeSIM` are both available to the users. `threedhst`, which is Python based, was developed independently by the 3D-HST team and is not supported at STScI. `NPSpec` is a Python implementation of a simulation kernel that allows one to perform quick simulations for the purpose of SED fitting. The latter is not directly available to the user community and was used to gain experience with direct 2D SED fitting of observed spectra.

The three codes described in this document share common configuration files (e.g. `aXe` Configuration Files) that define how a given pixel in the direct image is dispersed into the various grism orders (the "trace"), both geometrically (the location of the trace on the image) and spectro-photometrically (the intensity of the spectrum in electrons per second per pixel).

The geometric parameters that define the trace relative to a reference pixel in the direct image are

- `DYDX`: Pixel offset in y of the center of the trace as a function of the x distance from the reference pixel (see §A4).
- `DLDP`: Effective wavelength of the pixel defined *along* the trace (see §A6).

These parameters are stored in the configuration files as polynomials whose coefficients can vary in both x and y dimensions across the instrumental field of view. *The polynomials are defined in the nominal distorted frame of the HST science instruments* (i.e., FLT images produced by the ACS and WFC3 calibration pipelines). A more complete description of these polynomials is presented in Appendix A.

The flux calibration of the spectral orders is defined in separate sensitivity files with
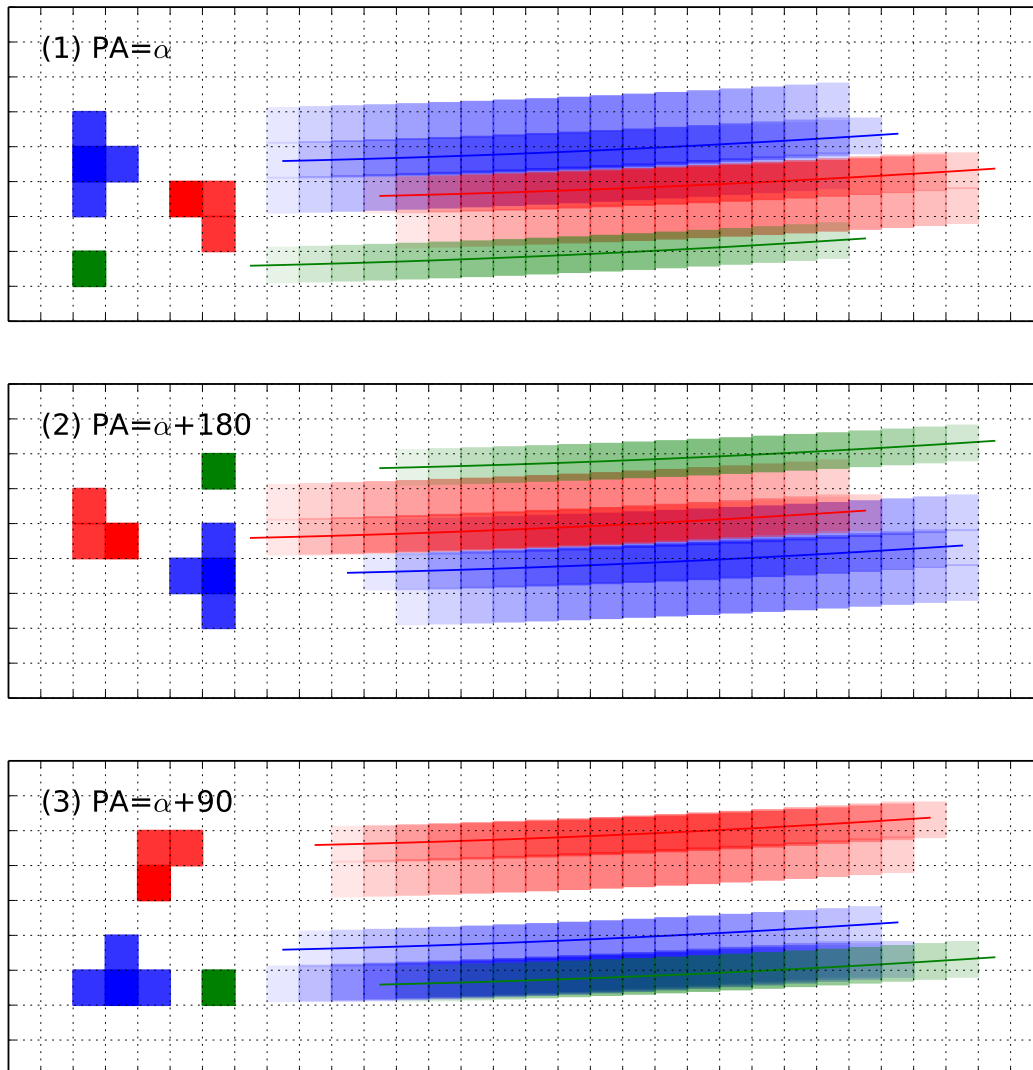
Fig. 2.— A simplified view of the complex relation between object morphologies, spatial distribution and spectral contamination when slitless observations are made at different position angles. In each panel, three objects with different morphologies are shown at left (blue, green and red) and their respective spectra are shown dispersed toward the right. The middle panel shows the dispersion when the field is rotated 180 degrees with respect to the top panel while the bottom panel shows the result of a 90 degree rotation.

the sensitivity and its corresponding uncertainty in units of f_λ (erg/s/cm^2/Å) provided as a function of wavelength. Though the grism dispersion can vary across the instrumental field of view, the sensitivity curve is currently assumed to be constant across the ACS fields. In WFC3, the variation of the sensitivity across the field of view is corrected by the grism-
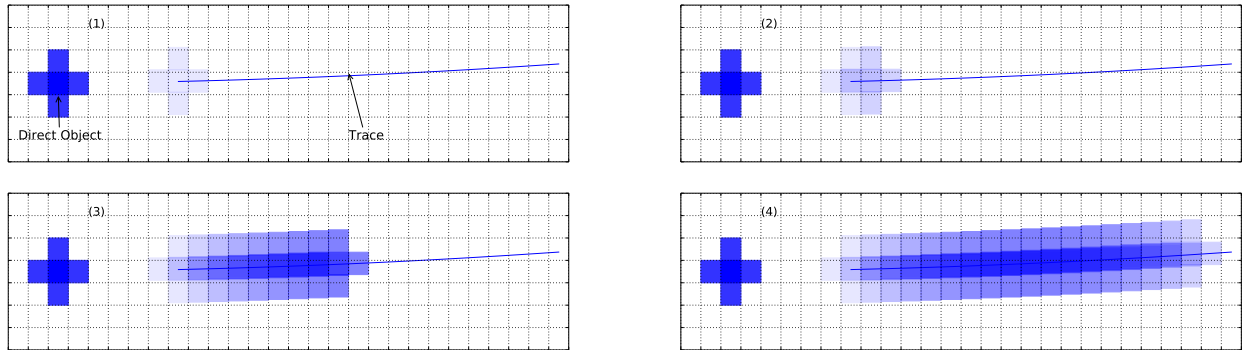
Fig. 3.— This example shows the process (Panels 1 through 4) of Object Based simulation, where the footprint of the object, shown on the left, is convolved with the spectral trace.

specific flat-field calibration files.

3.1. Two approaches of the existing software

The simulation software currently available approach the problem in two fundamentally different ways: disperse the entire object at once (Object Based), or disperse each input pixel separately (Pixel Based).

3.1.1. Object based

The Object-Based method generates a simulated dispersed spectrum of a source by convolving the trace and the footprint of the object. In this approach, the one dimensional spectrum of the source is first generated in units of e^-/s. This spectrum is computed by multiplying the assumed intrinsic spectrum of the object by the sensitivity function. This spectrum is dropped onto an array using the DYDX relation and then simply convolved with the footprint of the object. This footprint can take several forms. It can be a simple Gaussian description of the objects (based, for example, on the SExtractor A_IMAGE, B_IMAGE, THETA_IMAGE parameters). It can also be a normalized stamp image of the object (generated by combing a SExtractor segmenation map and a direct image of the object). The convolution can then be either direct, which allows for changing the convolution kernel as a function of wavelength, or it can be FFT based. Figure 3 illustrates this method. Currently, `aXe`/`aXeSIM` and `NPspec` use this method.
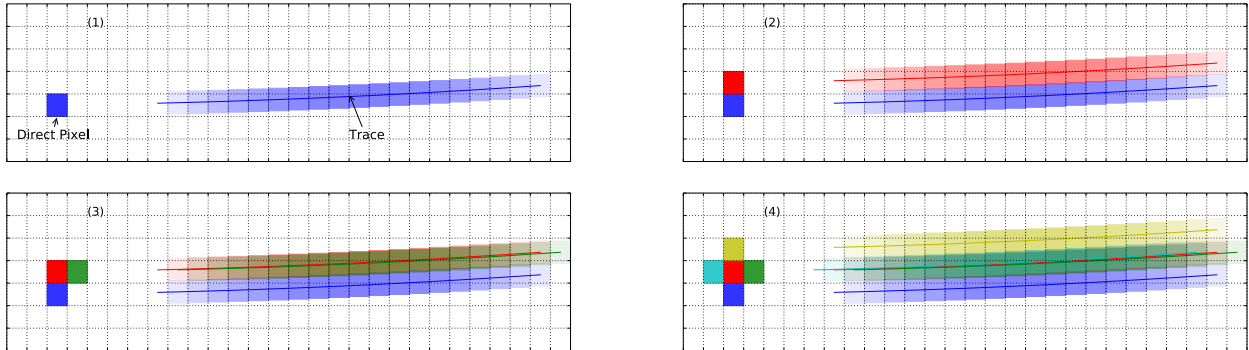
Fig. 4.— This example shows the process (Panels 1 through 4) of Pixel Based simulation, where individual pixels, shown on the left, are convolved with the spectral trace.

3.1.2. Pixel based

The other approach, Pixel Based, treats each pixel as a separate entity and the trace and wavelength relations are obtained and mapped onto the simulated image. This allows for individual pixels to have different colors/spectra as a different spectrum can be assigned to each pixel in the direct image.

4. Use of Simulations

Currently, we are aware of simulations of grism observations to have been used for the following purpose:

1. **To obtain accurate count rate estimates for faint sources and hence serve as a realistic ETC**
   The WFC3 Exposure Time Calculator supports the G102 and G141 grism modes but does so in an approximate manner. Users have found in the past that a full 2D simulation is preferred to account for the signal dilution that occurs with different (extended) object morphologies.

2. **To estimate the amount of contamination in extracted spectra**
   Contamination of overlapping spectra from neighboring objects is an important problem that must be addressed with slitless spectroscopy. The lack of slits results in every source in the field producing spectra (more than one since multiple orders are present). Software packages such as `aXe` use simulations to estimate this contamination. The quality of this estimate is ultimately limited by the amount of information that is available for every single source in the field (e.g. their location, morphology and spectral

characteristics). Simulations have hence been an intrinsic part of the extraction and calibration process from the beginning, although in large part the process has been hidden from users.

3. **To examine contamination for a given position angle on the sky**
   The amount of contamination affecting the spectrum of a source is determined by the brightness, location and spectral characteristics of neighboring sources *and* by the orientation of the field since spectra are dispersed in a specific direction (in the case of WFC3, roughly aligned with the x-axis; see Figure 2).

4. **To directly fit models to 2D spectra to estimate redshift and other galaxy stellar population models**
   As computers have become faster it has become more attractive to fit models to dispersed spectra directly. This removes the need to extract and calibrate spectra (which is akin to a de-convolution process) when estimating quantities such as spectro-photometric redshifts (using both broadband and spectral information).

4.1. "aXe/aXeSIM/NPSpec"

The `aXe` software package contains all the code required to generate accurate simulations of a field. `aXe`, by design, is an end-to-end solution to spectral extraction and calibration. `aXe` was developed by N. Pirzkal at ST-ECF (Space Telescope European Coordinating Facility) in 2001 specifically to extract and calibrate ACS sitless spectra (WFC, HRC, grism and prims). It was further improved at ST-ECF by M. Kümmel until it was formally handed to STScI/OED when ST-ECF was shut down. `aXe` was designed to be a generalized, "pipeline-able" package written in C that followed on the footsteps of `NICMOSlook` and `CALNIC-C`, developed by W. Freudling. Many of the features found in `aXe` were based on lessons learned from `NICMOSlook`. As such, its inputs are high-level products, such as AstroDrizzled mosaics of a field and `SExtractor` catalogs. The end result it produces are 2D and 1D extracted, rectified and calibrated spectra. The `aXe` package is run as a black-box and the user has little control on the low level operations `aXe` performs, and there are only a limited number of ways the `aXe` operations can be customized with or without the addition of custom code. The `aXe` process is expected to be essentially hands-off and to can take a significant amount of time to execute (ten of minutes to hours). `aXeSIM`, written around `aXe`'s internal algorithms, is based on the same principles. `aXeSIM`, using its Objects Based approach, offers a comprehensive approach to generate dispersed simulations of an entire field, perhaps with thousands of objects. `aXeSIM` was not designed to generate single object simulations quickly.

The Objects Based approach was also implemented in pure Python code, dubbed

`NPSpec`. This implementation was created with the specific task of 2D in-situ fitting of model spectra (e.g., Bruzual and Charlot 2003 spectral synthesis models) to an observed spectrum in an *HST* pipeline calibrated FLT file. The main goal was to create a small kernel of Python code that could create reasonable 2D simulations of a spectrum that could then be compared to the observation and used as part as a larger Markov Chain Monte Carlo approach. The code followed the same approach as `aXe` and produced similar results, using either stamp images and direct convolution or gaussian kernels and FFT convolution. This prototype showed that a pure Python approach could easily generate a simulated spectrum in approximately 30 ms, when using FFT and gaussian kernels.

The following is an illustrative list of the input products needed by `aXe` and `aXeSIM` and the output products they produce. Although `aXe` and `aXeSIM` are meant to be high level packages, a significant amount of work is required to assemble some of the needed data products. Broadband mosaics of the field need to be created, SExtractor catalogs need to be generated and merged into a master catalog, "stamp" images need to be created if anything more complex than simple gaussian morphologies are required:

Input/output products for `aXeSIM`

Inputs

- SExtractor catalog with multiple broad-band magnitudes
- Optional:"Stamp" image (broad band images and segmentation footprint) for each object to simulate
- `aXe` configuration file of the G102 or G141 WFC3 grism.

Output

- 2D FITS file containing the simulated image of the entire field, in units of electrons/s
- Optional: Extracted 2D/1D spectra for objects down to a magnitude limit in the direct images.

Running time

- Approximately 0.05 second per object and spectral order. A field containing 3168 objects and a total of 8662 spectral order takes approximately 8 minutes to simulate (not including the time required to generate stamp images)
- An aXe extraction doubles this time per object for a simple extraction.

Input/output products for `NPSpec`

Inputs

- Broad-band magnitudes for the source to simulate

- "Stamp" image of the object (broad band images and segmentation footprint) to simulate [optional]

- `aXe` configuration file of the G102 or G141 WFC3 grism.

Outputs

- 2D numeric Python array containing the expected count rate per pixel for the simulated spectrum.

Running Time

- A single simulated spectrum requires approximately 0.03 seconds.

4.2. "threedhst"

G. Brammer developed a pipeline independent of `aXe` (but that relies on the `aXe` configuration files) to process the WFC3/IR G141 spectra obtained by the 3D-HST treasury program (GO 12177 & 12328; PI: van Dokkum; see Brammer et al. 2012). At the heart of this pipeline, dubbed `threedhst`, is the "Pixel-Based" approach demonstrated in Fig. 4 with the spatial template taken from either the direct imaging itself or, optionally, an aligned reference image (for example, a deeper wide-field mosaic). In practice, the pipeline presently assumes pixels grouped within a segmentation region of a particular object share a common spectrum, so in some sense it is a hybrid of the two methods described above. However, it would be straightforward (i.e., additional bookkeeping) to extend the algorithm for an arbitrary separation of independent spatial components (e.g., a bulge and a disk of a distant galaxy).

The `threedhst` pipeline has been designed for speed and flexibility in generating model two-dimensional grism spectra from arbitrary input spectra, for example, a galaxy continuum plus emission line template. The model spectra can then be compared directly to the observed 2D grism spectra, and fits can be evaluated based on the individual pixel uncertainties as defined by the instrument noise model and unaffected by the correlated noise that results from drizzling. The computation of a model 2D WFC3/G141 spectrum from a higher-resolution input template takes $\sim$3.5 ms for a (NY, NX) = (16, 143) spectral cutout

($2''$ along the y spatial axis). The `threedhst` pipeline includes an automated redshift fitting algorithm that can account for additional redshift constraints from broad-band photometry of a given object. The algorithm, which evaluates galaxy continuum plus emission-line template fits computed on a high-resolution redshift grid takes $\sim$10 s per object, with the runtime depending on the number of spectral templates and the size of the thumbnail used as the spatial kernel. We have recently explored implementing the Object-Based model-generation method in the `threedhst` pipeline, finding substantial speed improvement in generating the model spectra: models for the (16,143) spectral cutout can be generated in just 0.2 ms. These models are essentially indistinguishable from those computed with the Pixel-Based approach, given that both methods currently assume that all pixels within the spatial kernel share the same spectrum.

Input/output products for `threedhst`

Inputs

- FLT images from the direct and grism exposures. The FLT images are WCS-aligned and background-subtracted before they are used for modelling and extracting the grism spectra.

- SExtractor catalog and segmentagion image [Optional. If not provided, will be generated directly from the FLT images.]

- `aXe` configuration file of the G102 or G141 WFC3 grism.

- Optional: aligned deep image that can be used as the spatial reference in place of the usually short direct image exposures

- Optional: photometric catalogs with constraints on galaxy redshifts

Outputs

- 2D FITS file containing the simulated image of the entire field, in units of electrons/s. The simulated image is used as the contamination model for individual object spectra that are extracted from the field.

- Optional: Extracted 2D/1D spectra for objects down to a magnitude limit in the direct image.

- Optional: Fits to redshifts and emission line strengths of individual objects.

Running time

The following run times were computed for a test case running a full 2D model of a single G141 pointing and extracting and fitting a single object in that pointing. Note that these

calculations assume fitting on the native WFC3 pixel grid. Each visit of the 3D-HST survey (and other similar programs) obtains 4 dithered exposures in a 2×2 box pattern that samples the WFC3 pixel at half-pixel intervals. Fits for the 3D-HST survey involve interlacing the direct images to a 2×2 grid so the run times can be $\sim$4 times longer than the numbers reported here (analogous to fitting the four exposures simultaneously).

- Generate a full 2D model for a test pointing with 500 objects at $H_{140} < 26.5$ ... 106 s
- Extract 2D+1D spectra of a single object .... 0.4 s
- Fit the object redshift on a redshift grid .... 9.2 s
- Monte-Carlo fit for the emission line strengths .... 6.4 s

5.   Example comparison

We have run the three grism analysis pipelines on a common dataset to facilitate comparison between them. The comparison dataset, IBHM52[1], is taken from the 3D-HST program (GO 13238). This visit contains four dithered direct image exposures in the F140W filter ($t_{\mathrm{exp}} = 203$ s) each followed by a single G141 exposure ($t_{\mathrm{exp}} = 1103$ s) taken at the same position without moving the telescope. The pointing lies within the larger COSMOS survey field imaged in a variety of ACS and WFC3 bands. Using the larger image mosaics of the COSMOS field, all three codes can be used to model the grism spectra of objects near the edges of the detector field of view whose direct images fall off of the F140W imaging of the visit itself.

Examples of two-dimensional spectra of stars and galaxies as modeled with the different codes are shown in Figs.5 and 6 below. The top two panels show the observed direct and grism images. Note that the direct image has been shifted for the display; the position of the direct image in the FLT frame is $\sim$40 pixels to the left of the start of the G141 +1st order spectrum. The bottom panels show residuals computed with the `aXeSIM` and `threedhst` pipelines. With `aXeSIM`, we compute models for both Gaussian spatial profiles ("gauss") and for spatial profiles as extracted from the direct image itself within a segmentation region around an object ("seg"). The spectral shape is also modeled in two ways with `aXeSIM`, the first assuming a flat f_λ spectrum with the normalization determined from the F140W direct image, and the second ("All") with a continuum shape determined from the additional ancillary imaging available in this field (WFC3/IR F125W, F140W, and F160W). The `threedhst` model uses a segmentation-based approach like `aXe` but can additionally refine the spectral

---

[1]http://archive.stsci.edu/hst/search.php?action=Search&sci_data_set_name=IBHM52*

shape based on the observed spectrum itself. *The segmentation-based approach is clearly preferable to assuming Gaussian object profiles*, and even more so for point sources.

6.   A Comment on Next Generation Grism Analysis Software

A key limitation to the methods presented above is dealing with deblending (or contamination) of overlapping spectra and optimally combining information from multiple orientations or positions. While `aXe` uses the same engine as `aXeSIM` to quantitatively estimates the amount of spectral contamination, this information is not used to deblend, or clean, calibrated spectra. This information is instead provided solely as a (fixed) quantitative estimate of the contamination. At present, the WFC3 Grism Group is exploring two different (but related) methods based on their computational feasibility, fundamental assumptions, and limiting systematics. The first method, *forward modeling*, essentially seeks to predict a given dispersed image based on the properties of the sources (such as number/position of sources and their spectra). These properties are optimized by comparing with the observed dispersed image, but typically the positions are held fixed (to those seen in the direct image). The second method, *linear reconstruction*, attempts to reconstruct the optimal spectra by inversion using least-squares techniques. Each approach has unique pros and cons, and neither seems to be inherently superior at this stage. We have just begun developing prototypes of these tools which are not necessarily built using previously discussed pipelines or configurations, indeed we leave open the possibility that new calibrations, reference files, or descriptions of the grism images will be needed for these tools.
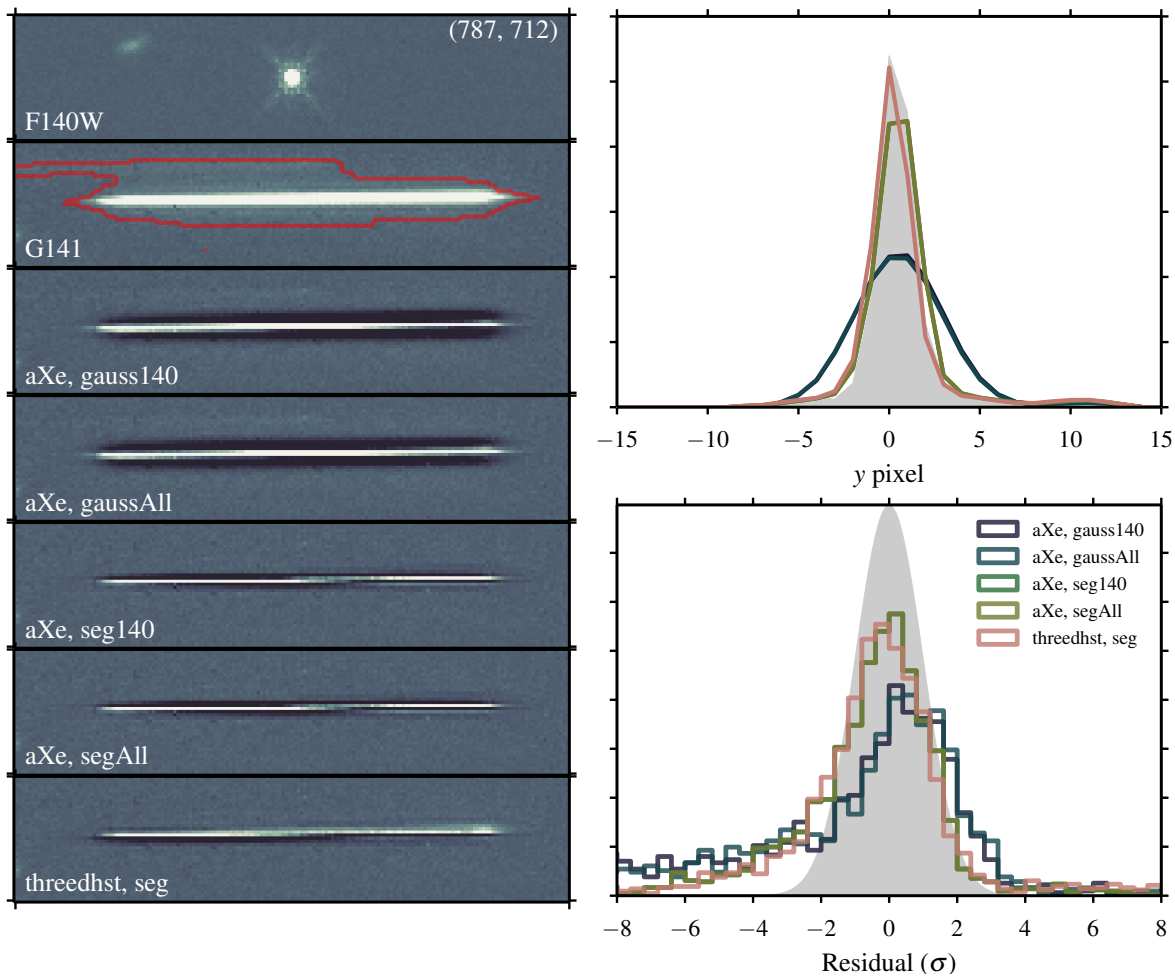
Fig. 5.— 2D spectrum of a star. The top panel shows the F140W image offset to put the object at the center of the panel. The second panel shows the observed G141 spectrum. The red outline shows the subset of pixels used in the distribution of residuals shown in the lower right panel. The bottom five 2D spectrum panels show the *observed − model* residuals for the modeling codes and input assumptions, as labeled. The top right panel shows the spatial profile of the 2D spectrum collapsed along the x (wavelength) axis for the observed (filled gray) and model (colored lines) spectra.
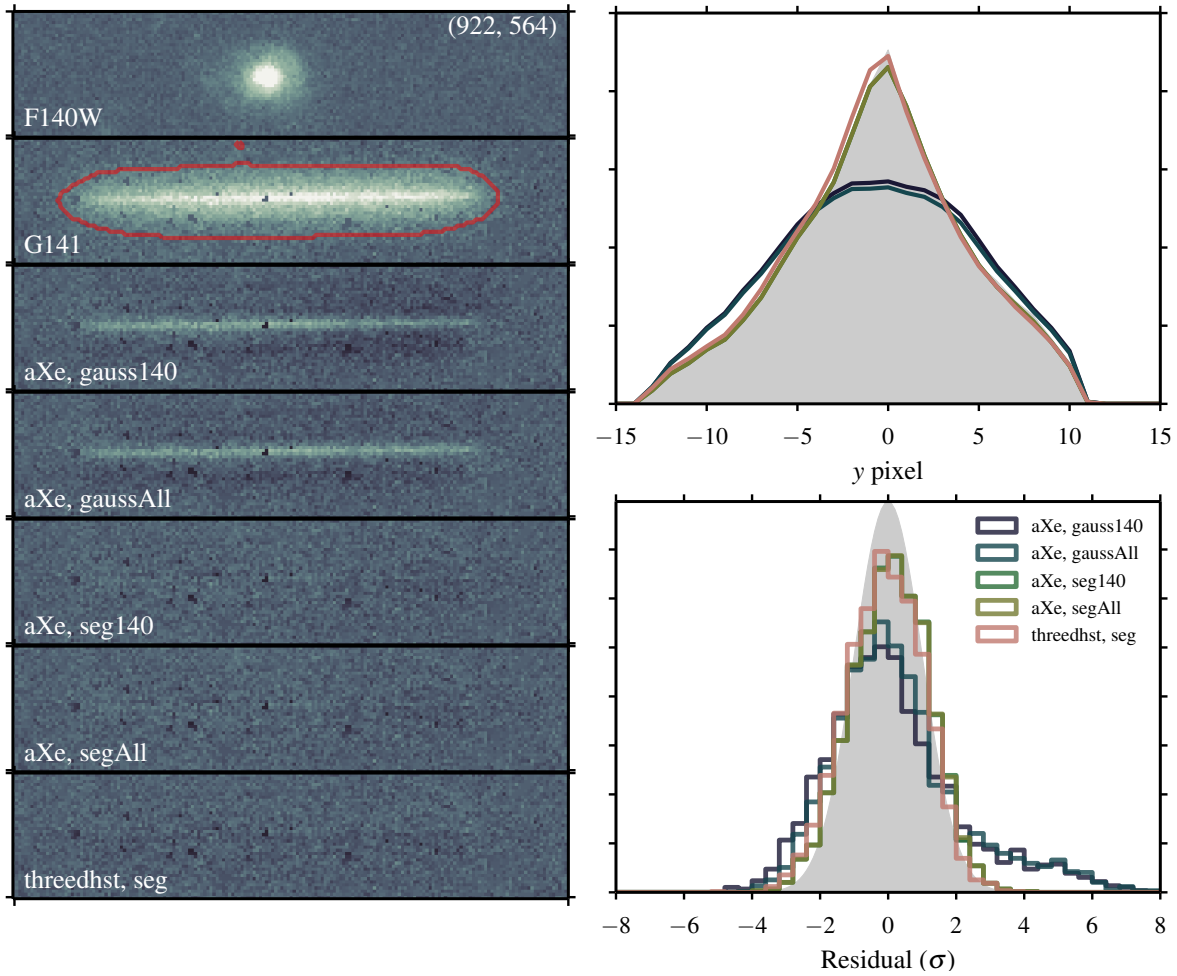
Fig. 6.— 2D spectrum of an extended galaxy. The panels are the same as Fig. 6. The fit residuals are somewhat better than for the stellar model because PSF and discrete pixel grid effects are less important for the extended galaxy.

A. The aXe Configuration File

aXe, aXeSIM, NPspec and `threedsht` rely on an aXe configuration file to describe how to simulate and extract *HST* slitless spectra. The aXe configuration file is based on the assumption that every aspect of the instrument calibration can be parametrized as n^{th} order polynomial. The field dependence (how the dispersion varies across the detector) is also assumed to be smooth varying and to be approximated as a 2D polynomial function of the mth order. While multiple orders are treated independently in a single aXe configuration file, a separate aXe configuration file is required for each detector of each instrument. The two WFC detectors on the ACS instrument, for example, are treated independently. Overlaps bettween multiple detectors of an instrument is not usually a problem as the dispersion is in the x-direction while the detectors are stacked in the y-direction. aXe does not attempt to combine spectra obtained using both detectors, for example when large dithers in the y-direction are used.

For the purpose of simulations, the aXe configuration file is used to identify the various spectral orders to be simulated (which are referred as "BEAMs" in the aXe configuration file). For a given order (beam "A" is usually associated with the brightest $+1^{st}$ order), there are two parameters of interest: DYDX and DLDP. They describe the spectral trace and its wavelength dependence, respectively. DYDX produces the expected offset dy in the y-direction with respect to the y-position of the source (as measured through an imaging filter without the dispersing element) as a function of dx, the offset between source and a given column in the x-direction:

$$dy = a_0 + a_1 \cdot dx + a_2 \cdot dx^2 + ... + a_n \cdot dx^n, \tag{A1}$$

where dy is the offset (in pixels) between the centroid of the trace and the y position (pixel j) of the source, dx is the offset in the x-direction between a particular pixel on the trace and the x-position (pixel i) of the source (Figure 7). The order of the polynomial is n, with $n = 1$ corresponding to a simple linear dispersion. If the instrument had no field dependence and dispersed all spectra in the same way across the field of view, the coefficients a_n would be simple constants. In the case of WFC3, however, there is a significant amount of field dependence of the trace and this is parametrized by allowing each value of a_n to itself be a 2D polynomial that is a function of the source position (i,j) on the detector. In the case of a second order 2D field dependence, we have

$$a_n(i, j) = b_{n,0} + b_{n,1} \cdot i + b_{n,2} \cdot j + b_{n,3} \cdot i^2 + b_{n,4} \cdot i \cdot j + b_{n,5} \cdot j^2. \tag{A2}$$

A second order 2D field dependent representation of a linear ($n = 1$) dispersion therefore
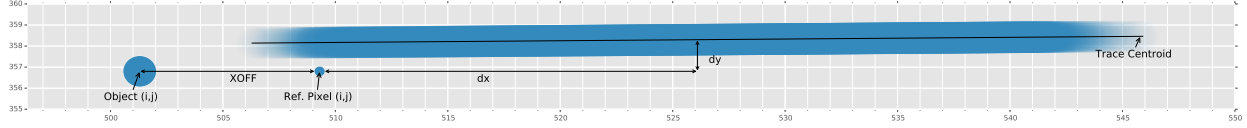
Fig. 7.— The slitless trace. The object, as it would be visible using a broad band filter, is shown at the bottom left and has pixel coordinates of (i,j). We also show on the same Figure the resulting spectrum as produced by the G141 grism. The measurements quantities dx and dy used in Equations A1, A2 and A3 are shown.
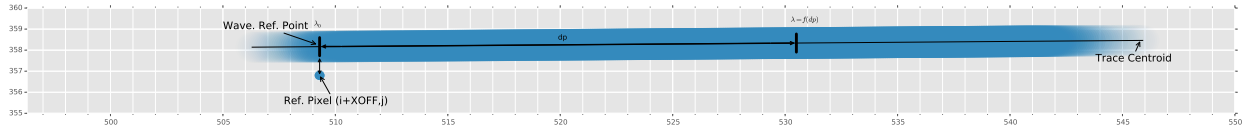

Fig. 8.— The wavelength calibration. The wavelength along the trace is computed as a function of the dp, the arc along the trace from the reference pixel (point on the trace with the same x-coordinate as the reference pixel) and a given position on the trace.

becomes

$$dy(i,j,dx) = b_{0,0} + b_{0,1} \cdot i + b_{0,2} \cdot j + b_{0,3} \cdot i^2 + b_{0,4} \cdot i \cdot j + b_{0,5} \cdot j^2 \tag{A3}$$
$$+ dx \ (b_{1,0} + b_{1,1} \cdot i + b_{1,2} \cdot j + b_{1,3} \cdot i^2 + b_{1,4} \cdot i \cdot j + b_{1,5} \cdot j^2),$$

where we have explicitly expressed dy as a function of the source position (i,j) and of the x-offset (dx) between the source and a particular point on the spectrum.

The description of the wavelength dependence *along* the trace DLDP is similar and parametrizes the difference in wavelength between the reference point on the trace and a given position on the trace.

$$\lambda = \alpha_0 + \alpha_1 \cdot dp + \alpha_2 \cdot dp^2 + ... + \alpha_n \cdot dp^n \tag{A4}$$

where again, a 2D field dependence implies

$$\alpha_n(i,j) = \beta_{n,0} + \beta_{n,1} \cdot i + \beta_{n,2} \cdot j + \beta_{n,3} \cdot i^2 + \beta_{n,4} \cdot i \cdot j + \beta_{n,5} \cdot j^2 \tag{A5}$$

and we have, in the case of a linear wavelength dispersion solution,

$$\lambda(i,j,dp) = \beta_{0,0} + \beta_{0,1} \cdot i + \beta_{0,2} \cdot j + \beta_{0,3} \cdot i^2 + \beta_{0,4} \cdot i \cdot j + \beta_{0,5} \cdot j^2 \tag{A6}$$
$$+ dp \ (\beta_{1,0} + \beta_{1,1} \cdot i + \beta_{1,2} \cdot j + \beta_{1,3} \cdot i^2 + \beta_{1,4} \cdot i \cdot j + \beta_{1,5} \cdot j^2)$$

The `aXe` configuration file encodes the $b_{n,m}$ parameters of Equation A3 (describing the trace) in DYDX and the $\beta_{n,m}$ parameters of Equation A6 (describing the wavelength dispersion) in DLDP. To be precise, dp in the preceding equations is the arc-length along the trace, given as:

$$dp(X) = \int_0^X \sqrt{1 + y'(x)^2}\, dx \tag{A7}$$

where $y'(x) = dy/dx$. For $n \leq 2$, this integral has analytic solutions, which we utilize in the example code below. For higher order dy/dx trace shapes, such as the curved orders of the WFC3/UVIS G280 grism, we evaluate this integral numerically (which can be very slow).

B.   Code to compute the spectral trace

We provide the Python code below (`axe_disperse.py`) to demonstrate reading the `aXe`-formatted configuration files and computing the spatially-dependent dispersion parameters DYDX and DLDP (§A). Updates to the code (and updates to this document itself) can be found at:

https://github.com/WFC3Grism/CodeDescription/.

```python
"""
Demonstrate aXe trace polynomials.

  v1.0 - October 14, 2014  (G. Brammer, N. Pirzkal, R. Ryan)

"""
import numpy as np

class aXeConf():
    def __init__(self, conf_file='WFC3.IR.G141.V2.5.conf'):

        if conf_file is not None:
            self.conf = self.read_conf_file(conf_file)
            self.conf_file = conf_file
            self.count_beam_orders()

            ## Global XOFF/YOFF offsets
            if 'XOFF' in self.conf.keys():
                self.xoff = np.float(conf['XOFF'])
```

```
20              else:
21                  self.xoff = 0.
22
23              if 'YOFF' in self.conf.keys():
24                  self.yoff = np.float(conf['YOFF'])
25              else:
26                  self.yoff = 0.
27
28      def read_conf_file(self, conf_file='WFC3.IR.G141.V2.5.conf'):
29          """
30          Read an aXe config file, convert floats and arrays
31          """
32          from collections import OrderedDict
33
34          conf = OrderedDict()
35          lines = open(conf_file).readlines()
36          for line in lines:
37              ## empty / commented lines
38              if (line.startswith('#')) | (line.strip() == '') | ('"' in
    line):
39                  continue
40
41              ## split the line, taking out ; and # comments
42              spl = line.split(';')[0].split('#')[0].split()
43              param = spl[0]
44              if len(spl) > 2:
45                  value = np.cast[float](spl[1:])
46              else:
47                  try:
48                      value = float(spl[1])
49                  except:
50                      value = spl[1]
51
52              conf[param] = value
53
54          return conf
55
56      def count_beam_orders(self):
57          """
58          Get the maximum polynomial order in DYDX or DLDP for each beam
59          """
60          self.orders = {}
61          for beam in ['A','B','C','D','E','F','G','H','I','J']:
62              order = 0
63              while 'DYDX_%s_%d' %(beam, order) in self.conf.keys():
64                  order += 1
```

```
65
66            while 'DLDP_%s_%d' %(beam, order) in self.conf.keys():
67                order += 1
68
69            self.orders[beam] = order-1
70
71    def field_dependent(self, xi, yi, coeffs):
72        """
73        aXe field-dependent coefficients
74        """
75        ## number of coefficients for a given polynomial order
76        ## 1:1, 2:3, 3:6, 4:10, order:order*(order+1)/2
77        if isinstance(coeffs, float):
78            order = 1
79        else:
80            order = int(-1+np.sqrt(1+8*len(coeffs)))/2
81
82        ## Build polynomial terms array
83        ## $a = a_0+a_1x_i+a_2y_i+a_3x_i^2+a_4x_iy_i+a_5yi^2+$ ...
84        xy = []
85        for p in range(order):
86            for px in range(p+1):
87                xy.append(xi**(p-px)*yi**(px))
88
89        ## Evaluate the polynomial, allowing for N-dimensional inputs
90        a = np.sum((np.array(xy).T*coeffs).T, axis=0)
91
92        return a
93
94    def get_beam_trace(self, x=507, y=507, dx=0., beam='A'):
95        """
96        Get an aXe beam trace for an input reference pixel and
97        list of output x pixels dx
98        """
99        NORDER = self.orders[beam]+1
100
101        xi, yi = x-self.xoff, y-self.yoff
102        xoff_beam = self.field_dependent(xi, yi, self.conf['XOFF_%s' %(
    beam)])
103        yoff_beam = self.field_dependent(xi, yi, self.conf['YOFF_%s' %(
    beam)])
104
105        ## y offset of trace (DYDX)
106        dydx = np.zeros(NORDER) #0 #+1.e-80
107        for i in range(NORDER):
108            if 'DYDX_%s_%d' %(beam, i) in self.conf.keys():
```

```
109                    coeffs = self.conf['DYDX_%s_%d' %(beam, i)]
110                    dydx[i] = self.field_dependent(xi, yi, coeffs)
111
112            # $dy = dydx_0+dydx_1 dx+dydx_2 dx^2+$ ...
113            dy = yoff_beam
114            for i in range(NORDER):
115                dy += dydx[i]*(dx-xoff_beam)**i
116
117            ## wavelength solution
118            dldp = np.zeros(NORDER)
119            for i in range(NORDER):
120                if 'DLDP_%s_%d' %(beam, i) in self.conf.keys():
121                    coeffs = self.conf['DLDP_%s_%d' %(beam, i)]
122                    dldp[i] = self.field_dependent(xi, yi, coeffs)
123
124            ## dp is the arc length along the trace
125            ## $\lambda = dldp_0 + dldp_1 dp + dldp_2 dp^2$ ...
126            if self.conf['DYDX_ORDER_%s' %(beam)] == 0:   ## dy=0
127                dp = dx
128            elif self.conf['DYDX_ORDER_%s' %(beam)] == 1: ## constant dy/dx
129                dp = np.sqrt(1+dydx[1]**2)*(dx-xoff_beam)
130            elif self.conf['DYDX_ORDER_%s' %(beam)] == 2: ## quadratic trace
131                u0 = dydx[1]+2*dydx[2]*(0)
132                dp0 = (u0*np.sqrt(1+u0**2)+np.arcsinh(u0))/(4*dydx[2])
133                u = dydx[1]+2*dydx[2]*(dx-xoff_beam)
134                dp = (u*np.sqrt(1+u**2)+np.arcsinh(u))/(4*dydx[2])-dp0
135            else:
136                ## high order shape, numerical integration along trace
137                ## (this can be slow)
138                xmin = np.minimum((dx-xoff_beam).min(), 0)
139                xmax = np.maximum((dx-xoff_beam).max(), 0)
140                xfull = np.arange(xmin, xmax)
141                dyfull = 0
142                for i in range(1, NORDER):
143                    dyfull += i*dydx[i]*(xfull-0.5)**(i-1)
144
145                ## Integrate from 0 to dx / -dx
146                dpfull = xfull*0.
147                lt0 = xfull <= 0
148                if lt0.sum() > 1:
149                    dpfull[lt0] = np.cumsum(np.sqrt(1+dyfull[lt0][::-1]**2))[::-1]
150                    dpfull[lt0] *= -1
151                #
152                gt0 = xfull >= 0
153                if gt0.sum() > 0:
```

```
154                    dpfull[gt0] = np.cumsum(np.sqrt(1+dyfull[gt0]**2))
155
156                dp = np.interp(dx-xoff_beam, xfull, dpfull)
157
158            ## Evaluate dldp
159            lam = dp*0.
160            for i in range(NORDER):
161                lam += dldp[i]*dp**i
162
163            return dy, lam
164
165        def show_beams(self, beams=['E','D','C','B','A']):
166            """
167            Make a demo plot of the beams of a given configuration file
168            """
169            import matplotlib.pyplot as plt
170
171            x0, x1 = 507, 507
172            dx = np.arange(-800,1200)
173
174            if 'WFC3.UV' in self.conf_file:
175                x0, x1 = 2073, 250
176                dx = np.arange(-1200,1200)
177            if 'G800L' in self.conf_file:
178                x0, x1 = 2124, 1024
179                dx = np.arange(-1200,1200)
180
181            s=200 # marker size
182            fig = plt.figure(figsize=[10,3])
183            plt.scatter(0,0,marker='s', s=s, color='black', edgecolor='0.8',
184                        label='Direct')
185
186            for beam in beams:
187                if 'XOFF_%s' %(beam) not in self.conf.keys():
188                    continue
189
190                xoff = self.field_dependent(x0, x1, self.conf['XOFF_%s' %(beam
    )])
191                dy, lam = self.get_beam_trace(x0, x1, dx=dx, beam=beam)
192                xlim = self.conf['BEAM%s' %(beam)]
193                ok = (dx >= xlim[0]) & (dx <= xlim[1])
194                plt.scatter(dx[ok]+xoff, dy[ok], c=lam[ok]/1.e4, marker='s', s
    =s,
195                            alpha=0.5, edgecolor='None')
196                plt.text(np.median(dx[ok]), np.median(dy[ok])+1, beam,
197                        ha='center', va='center', fontsize=14)
```

```
198            print 'Beam %s, lambda=(%.1f - %.1f)' %(beam, lam[ok].min(),
199                                               lam[ok].max())
200
201        plt.grid()
202        plt.xlabel(r'$\Delta x$')
203        plt.ylabel(r'$\Delta y$')
204
205        cb = plt.colorbar(pad=0.01, fraction=0.05)
206        cb.set_label(r'$\lambda\,(\mu\mathrm{m})$')
207        plt.title(self.conf_file)
208        plt.tight_layout()
209        plt.savefig('%s.pdf' %(self.conf_file))
```