Contents

1 Goals 3
2 Features 5
3 Contribute 7
4 License 9
5 Indices and tables 11
Bibliography 67
Python Module Index 69
XDesign is an open-source Python package for creating x-ray imaging phantoms, simulating data acquisition, and benchmarking x-ray tomographic image reconstruction.
CHAPTER 1

Goals

• Assist faster development of new generation tomographic reconstruction methods
• Allow quantitative comparison of different reconstruction methods
• Create a framework for designing x-ray imaging experiments
CHAPTER 2

Features

• Configurable analytic 2D phantoms.
• Various visualization tools for statistics.
• Analytic projection operators.
CHAPTER 3

Contribute

- **Issue Tracker**: https://github.com/tomography/xdesign/issues
- **Documentation**: https://github.com/tomography/xdesign/tree/master/doc
- **Source Code**: https://github.com/tomography/xdesign/tree/master/xdesign
- **Tests**: https://github.com/tomography/xdesign/tree/master/test
The project is licensed under the BSD-3 license.
API

XDesign Modules:

\texttt{xdesign.acquisition}

Defines objects and methods for simulated data acquisition.

The acquisition module contains the objects and procedures necessary to simulate the operation of equipment used to collect tomographic data. This not only includes physical things like Probes, detectors, turntables, and lenses, but also non-physical things such as scanning patterns and programs.

### Classes:

- \texttt{Beam(p1, p2[, size])} 
  A thick line in 2-D cartesian space.
- \texttt{Probe(p1, p2[, size])} 
  An object for probing Phantoms.

### Functions:

- \texttt{sinogram(sx, sy, phantom[, noise])} 
  Return a sinogram of phantom and the probe.
- \texttt{angleogram(sx, sy, phantom[, noise])} 
  Return a angleogram of phantom and the probe.

Continued on next page
class xdesign.acquisition.Beam(p1, p2, size=0)
   Bases: xdesign.geometry.Line
   A thick line in 2-D cartesian space.
   A Beam is defined by two distinct points and a size (thickness). It is a subclass of a Probe.

   p1
       Point
   p2
       Point

   size
       scalar, optional – Size of the beam. i.e. the diameter

   distance(other)
       Return the closest distance between entities.

   half_space = <MagicMock name='mock()' id='139656766250624'>

   rotate(theta=theta, point=None, axis=None)
       Rotate entity around an axis which passes through an point by theta radians.

   skip

   translate(vector)
       Translate entity along vector.

class xdesign.acquisition.Probe(p1, p2, size=0)
   Bases: xdesign.acquisition.Beam
   An object for probing Phantoms.
   A Probe provides an interface for measuring the interaction of a Phantom and a beam. It contains information
   for interacting with Materials such as energy and brightness.

   measure(phantom, sigma=0)
       Return the probe measurement with optional Gaussian noise.

   Parameters
       sigma (float >= 0) – The standard deviation of the normally distributed noise.

   record()

   translate(dx)
       Translate beam along its normal direction.

xdesign.acquisition.sinogram(sx, sy, phantom, noise=False)
   Return a sinogram of phantom and the probe.

   Parameters
       • sx (int) – Number of rotation angles.
       • sy (int) – Number of detection pixels (or sample translations).
       • phantom (Phantom)

   Returns
       • sino (ndarray) – Sinogram.
• **probe** *(Probe)* – Probe with history.

```python
xdesign.acquisition.angleogram(sx, sy, phantom, noise=False)
```

Return a angleogram of phantom and the probe.

**Parameters**

- **sx** *(int)* – Number of rotation angles.
- **sy** *(int)* – Number of detection pixels (or sample translations).
- **phantom** *(Phantom)*

**Returns**

- **angl** *(ndarray)* – Angleogram.
- **probe** *(Probe)* – Probe with history.

```python
xdesign.acquisition.raster_scan(sx, sy)
```

Provides a beam list for raster-scanning.

The same Probe is returned each time to prevent recomputation of cached properties.

**Parameters**

- **sx** *(int)* – Number of rotation angles.
- **sy** *(int)* – Number of detection pixels (or sample translations).

**Yields** *Probe*

```python
xdesign.acquisition.angle_scan(sx, sy)
```

Provides a beam list for angle-scanning.

The same Probe is returned each time to prevent recomputation of cached properties.

**Parameters**

- **sx** *(int)* – Number of rotation angles.
- **sy** *(int)* – Number of detection pixels (or sample translations).

**Yields** *Probe*

### xdesign.algorithms

Defines methods for reconstructing data from the *acquisition* module.

The algorithm module contains methods for reconstructing tomographic data including gridrec, SIRT, ART, and MLEM. These methods can be used as benchmarks for custom reconstruction methods or as an easy way to access reconstruction algorithms for developing other methods such as noise correction.

**Note:** Using *tomopy* [https://github.com/tomopy/tomopy] is recommended instead of these functions for heavy computation.

**Functions:**

```python
art(probe, data, init[, niter])
```

Reconstruct data using ART algorithm.
Table 5.3 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sirt(probe, data, init[, niter])</code></td>
<td>Reconstruct data using SIRT algorithm.</td>
</tr>
<tr>
<td><code>mlem(probe, data, init[, niter])</code></td>
<td>Reconstruct data using MLEM algorithm.</td>
</tr>
<tr>
<td><code>stream(probe, data, init)</code></td>
<td>Reconstruct data.</td>
</tr>
<tr>
<td><code>update_progress(progress)</code></td>
<td>Draw a process bar in the terminal.</td>
</tr>
</tbody>
</table>

`xdesign.algorithms.art(probe, data, init, niter=10)`
Reconstruct data using ART algorithm.

`xdesign.algorithms.sirt(probe, data, init, niter=10)`
Reconstruct data using SIRT algorithm.

`xdesign.algorithms.mlem(probe, data, init, niter=10)`
Reconstruct data using MLEM algorithm.

`xdesign.algorithms.stream(probe, data, init)`
Reconstruct data.

`xdesign.algorithms.update_progress(progress)`
Draw a process bar in the terminal.

**Parameters**
- `process (float)` – The percentage completed e.g. 0.10 for 10%

## `xdesign.geometry`

Defines geometric objects to support Phantom definition and perform computational geometry for acquisition.

### Classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Entity()</code></td>
<td>Base class for all geometric entities.</td>
</tr>
<tr>
<td><code>Point(x)</code></td>
<td>A point in ND cartesian space.</td>
</tr>
<tr>
<td><code>Circle(center, radius)</code></td>
<td>Circle in 2D cartesian space.</td>
</tr>
<tr>
<td><code>Line(p1, p2)</code></td>
<td>Line in 2D cartesian space.</td>
</tr>
<tr>
<td><code>Triangle(p1, p2, p3)</code></td>
<td>Triangle in 2D cartesian space.</td>
</tr>
<tr>
<td><code>Rectangle(p1, p2, p3, p4)</code></td>
<td>Rectangle in 2D cartesian space.</td>
</tr>
<tr>
<td><code>Square(center, side_length)</code></td>
<td>Square in 2D cartesian space.</td>
</tr>
<tr>
<td><code>Polygon(vertces)</code></td>
<td>A convex polygon in 2D cartesian space.</td>
</tr>
<tr>
<td><code>Mesh([obj, faces])</code></td>
<td>A mesh object.</td>
</tr>
</tbody>
</table>

### `class xdesign.geometry.Entity`  
**Bases:** object

Base class for all geometric entities. All geometric entities should have these attributes and methods.

### Example

Examples can be given using either the Example or Examples sections. Sections support any reStructured-Text formatting, including literal blocks:

```bash
$ python example_numpy.py
```
Section breaks are created with two blank lines. Section breaks are also implicitly created anytime a new section starts. Section bodies may be indented:

**Parameters** `x (ndarray, list)` – ND coordinates of the point.

**Notes**

This is an example of an indented section. It’s like any other section, but the body is indented to help it stand out from surrounding text.

If a section is indented, then a section break is created by resuming unindented text.

---

**Note:** There are many other directives such as versionadded, versionchanged, rubric, centered, ... See the Sphinx documentation for more details.

---

**collision** *(other)*

Returns True if this entity collides with another entity.

**Note:** This method is inherited from `Entity` which means it is not implemented and will throw an error.

**contains** *(other)*

Return whether this Entity strictly contains the other entity.

Points on edges are contained by the Entity.

Returns a boolean for all `Entity`. Returns an array of boolean for MxN size arrays where M is the number of points and N is the dimensionality.

**Note:** This method is inherited from `Entity` which means it is not implemented and will throw an error.

**dim**

The dimensionality of the points which describe the entity.

**distance** *(other)*

Returns the closest distance between entities.

**Note:** This method is inherited from `Entity` which means it is not implemented and will throw an error.

**midpoint** *(other)*

Returns the midpoint between entities.

**rotate** *(theta, point=None, axis=None)*

Rotates the entity theta radians around an axis defined by a point and a vector

**Note:** This method is inherited from `Entity` which means it is not implemented and will throw an error.

**scale** *(vector)*

Scales the entity in each dimension according to vector. Scaling is centered on the origin.

---

5.1. API
Note: This method is inherited from Entity which means it is not implemented and will throw an error.

```python
translate(vector)
```
Translates the entity in the direction of a vector.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

class xdesign.geometry.Point(x)
Bases: xdesign.geometry.Entity
A point in ND cartesian space.

Parameters:
x (ndarray, list) – ND coordinates of the point.

Raises:
TypeError – If x is not a list or ndarray.

collision(other)
Returns True if this Point collides with another entity.

contains(other)
Return whether the other is within the bounds of the Point. Points can only contain other Points.

dim
The dimensionality of the points which describe the entity.

distance(other)
Returns the closest distance between entities.

midpoint(other)
Returns the midpoint between entities.

norm
Calculates the euclidian (L2) norm of the vector to the point.

rotate(theta, point=None, axis=None)
Rotates the point theta radians around the axis defined by the given point and axis.

scale(vector)
Scales the coordinates of the point in each dimension according to the given vector. Scaling is centered on the origin.

```python
translate(vector)
```
Translates the point along the given vector.

x
Dimension 0 of the point.

y
Dimension 1 of the point.

z
Dimension 2 of the point.

class xdesign.geometry.Circle(center, radius)
Bases: xdesign.geometry.Curve
Circle in 2D cartesian space.

center
Point – The center point of the circle.
radius
  scalar – The radius of the circle.

area
  Return the area.

circumference
  Returns the circumference.

collision (other)
  Returns True if this entity collides with another entity.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

contains (other)
  Return whether the Circle contains the other.
  Return one boolean for all geometric entities. Return an array of boolean for array input.

diameter
  Returns the diameter.

dim
  The dimensionality of the points which describe the entity.

distance (other)
  Returns the closest distance between entities.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

list
  Return list representation for saving to files.

midpoint (other)
  Returns the midpoint between entities.

patch
  Returns a matplotlib patch.

rotate (theta, point=None, axis=None)
  Rotates the Curve by theta radians around an axis which passes through a point radians.

scale (vector)
  Scales the entity in each dimension according to vector. Scaling is centered on the origin.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

translate (vector)
  Translates the Curve along a vector.

class xdesign.geometry.Line (p1, p2)
  Bases: xdesign.geometry.LinearEntity
  Line in 2D cartesian space.
  The constructor takes two unique Point.
p1
Point
p2
Point
collision(other)
Returns True if this entity collides with another entity.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

contains(other)
Return whether this Entity strictly contains the other entity.
Points on edges are contained by the Entity.
Returns a boolean for all Entity. Returns an array of boolean for MxN size arrays where M is the number of points and N is the dimensionality.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

dim
The dimensionality of the points which describe the entity.
distance(other)
Returns the closest distance between entities.
horizontal
True if line is horizontal.
intercept(n)
Calculates the intercept for the nth dimension.
length
Returns the length of the segment between p1 and p2.
list
Returns an list of coordinates where p1 is the first D coordinates and p2 is the next D coordinates.
midpoint(other)
Returns the midpoint between entities.
normal
Return the unit normal vector.
numpy
Returns an array of coordinates where the first row is p1 and the second row is p2.
points
Returns the two points used to define this linear entity as a 2-tuple.
rotate(theta, point=None, axis=None)
Rotates the LinearEntity by theta radians around an axis defined by an axis and a point.
scale(vector)
Scales the entity in each dimension according to vector. Scaling is centered on the origin.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.
slope
    Returns the slope of the line.

standard
    Returns coefficients for the first N-1 standard equation coefficients. The Nth is returned separately.

tangent
    Returns the unit tangent vector.

translate (vector)
    Translates the LinearEntity by the given vector.

vertical
    True if line is vertical.

xintercept
    Return the x-intercept.

yintercept
    Return the y-intercept.

class xdesign.geometry.Polygon (vertices)
    Bases: xdesign.geometry.Entity

    A convex polygon in 2D cartesian space.
    It is defined by a number of distinct vertices of class Point. Superclasses include Square, Triangle, etc.

    vertices
        List of Points

    area
        Returns the area of the Polygon.

    bounds
        Returns a 4-tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the Polygon.

    center
        The center of the bounding circle.

    collision (other)
        Returns True if this entity collides with another entity.

        Note: This method is inherited from Entity which means it is not implemented and will throw an error.

    contains (other)
        Return whether this Polygon contains the other.

    dim
        The dimensionality of the points which describe the entity.

    distance (other)
        Returns the closest distance between entities.

        Note: This method is inherited from Entity which means it is not implemented and will throw an error.

    edges
        Return a list of lines connecting the points of the Polygon.
list
Return list representation.

**midpoint** *(other)*
Returns the midpoint between entities.

**numpy**
Return Numpy representation.

**patch**
Returns a matplotlib patch.

**perimeter**
Return the perimeter of the Polygon.

**radius**
The radius of the bounding circle.

**rotate** *(theta, point=None, axis=None)*
Rotates the Polygon around an axis which passes through a point by theta radians.

**scale** *(vector)*
Scales the entity in each dimension according to vector. Scaling is centered on the origin.

*Note: This method is inherited from *Entity* which means it is not implemented and will throw an error.*

**translate** *(vector)*
 Translates the polygon by a vector.

**class** xdesign.geometry.Triangle *(p1, p2, p3)*
Bases: xdesign.geometry.Polygon

Triangle in 2D cartesian space.
It is defined by three distinct points.

**bounds**
Returns a 4-tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the Polygon.

**collision** *(other)*
Returns True if this entity collides with another entity.

*Note: This method is inherited from *Entity* which means it is not implemented and will throw an error.*

**contains** *(other)*
Return whether this Polygon contains the other.

**dim**
The dimensionality of the points which describe the entity.

**distance** *(other)*
Returns the closest distance between entities.

*Note: This method is inherited from *Entity* which means it is not implemented and will throw an error.*

**edges**
Return a list of lines connecting the points of the Polygon.
list
Return list representation.

midpoint (other)
Returns the midpoint between entities.

numpy
Return Numpy representation.

patch
Returns a matplotlib patch.

perimeter
Return the perimeter of the Polygon.

radius
The radius of the bounding circle.

rotate (theta, point=None, axis=None)
Rotates the Polygon around an axis which passes through a point by theta radians.

scale (vector)
Scales the entity in each dimension according to vector. Scaling is centered on the origin.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

translate (vector)
Translates the polygon by a vector.

class xdesign.geometry.Rectangle (p1, p2, p3, p4)
Bases: xdesign.geometry.Polygon
Rectangle in 2D cartesian space.
It is defined by four distinct points.

bounds
Returns a 4-tuple (xmin, ymin, xmax, ymax) representing the bounding rectangle for the Polygon.

collision (other)
Returns True if this entity collides with another entity.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

contains (other)
Return whether this Polygon contains the other.

dim
The dimensionality of the points which describe the entity.

distance (other)
Returns the closest distance between entities.

Note: This method is inherited from Entity which means it is not implemented and will throw an error.

edges
Return a list of lines connecting the points of the Polygon.
list
Return list representation.

midpoint \( \text{(other)} \)
Returns the midpoint between entities.

numpy
Return Numpy representation.

patch
Returns a matplotlib patch.

perimeter
Return the perimeter of the Polygon.

radius
The radius of the bounding circle.

rotate \( (\theta, \text{point=None, axis=None}) \)
Rotates the Polygon around an axis which passes through a point by \( \theta \) radians.

scale \( \text{(vector)} \)
Scales the entity in each dimension according to vector. Scaling is centered on the origin.

Note: This method is inherited from \text{Entity} which means it is not implemented and will throw an error.

translate \( \text{(vector)} \)
Translates the polygon by a vector.

class \text{xdesign.geometry.Square}(\text{center, side_length})
Bases: \text{xdesign.geometry.Rectangle}
Square in 2D cartesian space.
It is defined by a center and a side length.

bounds
Returns a 4-tuple \((\text{xmin, ymin, xmax, ymax})\) representing the bounding rectangle for the Polygon.

collision \( \text{(other)} \)
Returns True if this entity collides with another entity.

Note: This method is inherited from \text{Entity} which means it is not implemented and will throw an error.

contains \( \text{(other)} \)
Return whether this Polygon contains the other.

dim
The dimensionality of the points which describe the entity.

distance \( \text{(other)} \)
Returns the closest distance between entities.

Note: This method is inherited from \text{Entity} which means it is not implemented and will throw an error.

edges
Return a list of lines connecting the points of the Polygon.
list
   Return list representation.

midpoint (other)
   Returns the midpoint between entities.

numpy
   Return Numpy representation.

patch
   Returns a matplotlib patch.

perimeter
   Return the perimeter of the Polygon.

radius
   The radius of the bounding circle.

rotate (theta, point=None, axis=None)
   Rotates the Polygon around an axis which passes through a point by theta radians.

scale (vector)
   Scales the entity in each dimension according to vector. Scaling is centered on the origin.

   Note: This method is inherited from Entity which means it is not implemented and will throw an error.

translate (vector)
   Translates the polygon by a vector.

class xdesign.geometry.Mesh (obj=None, faces=[])
   Bases: xdesign.geometry.Entity
   A mesh object. It is a collection of polygons

append (t)
   Add a triangle to the mesh.

collision (other)
   Returns True if this entity collides with another entity.

   Note: This method is inherited from Entity which means it is not implemented and will throw an error.

dim
   The dimensionality of the points which describe the entity.

distance (other)
   Returns the closest distance between entities.

   Note: This method is inherited from Entity which means it is not implemented and will throw an error.

import_triangle (obj)
   Loads mesh data from a Python Triangle dict.

midpoint (other)
   Returns the midpoint between entities.

pop (i=-1)
   Pop i-th triangle from the mesh.
\textbf{rotate} (theta, point=None, axis=None)
Rotate entity around an axis which passes through a point by theta radians.

\textbf{scale} (vector)
Scale entity.

\textbf{translate} (vector)
Translate entity.

\textbf{xdesign.constants}

Constants in cgs units.

AVOGADRO\_NUMBER  [float] Avagadro constant [1/mol]
BOLTZMANN\_CONSTANT  [float] Boltzmann constant [erg/k]
CLASSICAL\_ELECTRON\_RADIUS  [float] Classical electron radius [cm]
ELECTRONIC\_CHARGE  [float] Electronic charge [esu]
ELECTRON\_VOLT  [float] Electron volt (keV) [erg]
ELECTRON\_MASS  [float] Electron mass [g]
FINE\_STRUCTURE\_CONSTANT  [float] Fine structure constant
PLANCK\_CONSTANT  [float] Reduced planck’s constant [keV*s]
PROTON\_MASS  [float] Proton mass [g]
SPEED\_OF\_LIGHT  [float] Speed of light in vacuum [cm/s]
THOMPSON\_CROSS\_SECTION  [float] Thomson cross section [cm^2]
PI  [float] Ratio of a circle’s circumference to its diameter

\textbf{xdesign.formats}

Functions:

\textbf{xdesign.material}

Defines objects which auto-generate a parameterized Phantom.

Classes:

\begin{tabular}{|l|l|}
\hline
\textit{Material}(formula, density) & Placeholder for class which uses NIST data to automatically calculate material properties based on beam energy. \\
\hline
\textit{XDesignDefault}() & Generates a Phantom for internal testing of XDesign. \\
\hline
\textit{HyperbolicConcentric}([min_width, exponent]) & Generates a series of cocentric alternating black and white circles whose radii are changing at a parabolic rate. \\
\hline
\textit{DynamicRange}([steps, jitter, geometry]) & Generates a phantom of randomly placed circles for determining dynamic range. \\
\hline
\end{tabular}

Continued on next page
Table 5.6 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DogaCircles</strong></td>
<td>(in_sizes, size_ratio, n_shuffles) Rows of increasingly smaller circles.</td>
</tr>
<tr>
<td><strong>SlantedSquares</strong></td>
<td>([count, angle, gap]) Generates a collection of slanted squares.</td>
</tr>
<tr>
<td><strong>UnitCircle</strong></td>
<td>([radius, mass_atten]) Generates a phantom with a single circle in its center.</td>
</tr>
<tr>
<td><strong>Soil</strong></td>
<td>([porosity]) Generates a phantom with structure similar to soil.</td>
</tr>
<tr>
<td><strong>WetCircles</strong></td>
<td>()</td>
</tr>
<tr>
<td><strong>SiemensStar</strong></td>
<td>([in_sectors, center, radius]) Generates a Siemens star.</td>
</tr>
<tr>
<td><strong>Foam</strong></td>
<td>([size_range, gap, porosity]) Generates a phantom with structure similar to foam.</td>
</tr>
<tr>
<td><strong>Metal</strong></td>
<td>([shape])</td>
</tr>
<tr>
<td><strong>SoftBiomaterial</strong></td>
<td>([shape])</td>
</tr>
<tr>
<td><strong>Electronics</strong></td>
<td>([shape])</td>
</tr>
<tr>
<td><strong>FiberComposite</strong></td>
<td>([shape])</td>
</tr>
</tbody>
</table>

**class** `xdesign.material.Material (formula, density)`

**Bases:** `object`

Placeholder for class which uses NIST data to automatically calculate material properties based on beam energy.

- **atom_concentration**
  Number of atoms per unit volume [1/cm^3].

- **atomic_form_factor**
  Measure of the scattering amplitude of a wave by an isolated atom. Read from NIST database [Unitless].

- **compton_cross_section**
  Compton cross-section of the electron [cm^2].

- **compton_scattering**
  X-ray attenuation due to the Compton scattering [1/cm].

- **electron_density**
  Electron density [e/cm^3].

- **linear_attenuation**
  Total x-ray attenuation [1/cm].

- **mass_ratio()**

- **number_of_elements()**

- **photoelectric_absorption**
  X-ray attenuation due to the photoelectric effect [1/cm].

- **photoelectric_cross_section**

- **reduced_energy_ratio**
  Energy ratio of the incident x-ray and the electron energy [Unitless].

- ** refractive_index**

**class** `xdesign.material.XDesignDefault`

**Bases:** `xdesign.phantom.Phantom`

Generates a Phantom for internal testing of XDesign.

The default phantom is: **nested**, it contains phantoms within phantoms; **geometrically simple**, the sinogram can be verified visually; and **representative**, it contains the three main geometric elements: circle, polygon, and mesh.

5.1. API
class xdesign.material.HyperbolicConcentric (min_width=0.1, exponent=0.5)

    Bases: xdesign.phantom.Phantom

    Generates a series of concentric alternating black and white circles whose radii are changing at a parabolic rate. These line spacings cover a range of scales and can be used to estimate the Modulation Transfer Function. The radii change according to this function: \( r(n) = r_0(n+1)^k \).

    radii
       list – The list of radii of the circles

    widths
       list – The list of the widths of the bands

class xdesign.material.DynamicRange (steps=10, jitter=True, geometry=Rectangle(Point([]), Point([]), Point([]), Point([])))

    Bases: xdesign.phantom.Phantom

    Generates a phantom of randomly placed circles for determining dynamic range.

    Parameters
       • steps (scalar, optional) – The orders of magnitude (base 2) that the colors of the circles cover.
       • jitter (bool, optional) – True : circles are placed in a jittered grid False : circles are randomly placed
       • shape (string, optional)

class xdesign.material.DogaCircles (n_sizes=5, size_ratio=0.5, n_shuffles=5)

    Bases: xdesign.phantom.Phantom

    Rows of increasingly smaller circles. Initially arranged in an ordered Latin square, the initial arrangement can be randomly shuffled.

    radii
       ndarray – radii of circles

    x
       ndarray – x position of circles

    y
       ndarray – y position of circles

class xdesign.material.SlantedSquares (count=10, angle=0.08726646259972222, gap=0)

    Bases: xdesign.phantom.Phantom

    Generates a collection of slanted squares. Squares are arranged in concentric circles such that the space between squares is at least gap. The size of the squares is adaptive such that they all remain within the unit circle.

    angle
       scalar – the angle of slant in radians

    count
       scalar – the total number of squares

    gap
       scalar – the minimum space between squares

    side_length
       scalar – the size of the squares

    squares_per_level
       list – the number of squares at each level
radius_per_level
   list – the radius at each level

n_levels
   scalar – the number of levels

class xdesign.material.UnitCircle(radius=0.5, mass_atten=1)
   Bases: xdesign.phantom.Phantom

Generates a phantom with a single circle in its center.

class xdesign.material.Soil(porosity=0.412)
   Bases: xdesign.material.UnitCircle

Generates a phantom with structure similar to soil.

References


class xdesign.material.WetCircles
   Bases: xdesign.material.UnitCircle

class xdesign.material.SiemensStar(n_sectors=4, center=Point([]), radius=0.5)
   Bases: xdesign.phantom.Phantom

Generates a Siemens star.

ratio
   scalar – The spatial frequency times the proportional radius. e.g to get the frequency, f, divide this ratio by some fraction of the maximum radius: f = ratio/radius_fraction

class xdesign.material.Foam(size_range=[0.05, 0.01], gap=0, porosity=1)
   Bases: xdesign.phantom.Phantom

Generates a phantom with structure similar to foam.

class xdesign.material.Metal(shape='square')
   Bases: xdesign.phantom.Phantom

class xdesign.material.SoftBiomaterial(shape='square')
   Bases: xdesign.phantom.Phantom

class xdesign.material.Electronics(shape='square')
   Bases: xdesign.phantom.Phantom

class xdesign.material.FiberComposite(shape='square')
   Bases: xdesign.phantom.Phantom

xdesign.metrics

Objects and methods for computing the quality of reconstructions.

Classes:

   ImageQuality(original, reconstruction[, method]) Store information about image quality.
**Functions:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compute_PCC(A, B[, masks])</code></td>
<td>Computes the Pearson product-moment correlation coefficients (PCC) for the two images.</td>
</tr>
<tr>
<td><code>compute_likeness(A, B, masks)</code></td>
<td>Predict the likelihood that each pixel in B belongs to a phase based on the histogram of A.</td>
</tr>
<tr>
<td><code>compute_background_ttest(image, masks)</code></td>
<td>Determine whether the background has significantly different luminance than the other phases.</td>
</tr>
<tr>
<td><code>compute_mtf(phantom, image)</code></td>
<td>Approximate the modulation transfer function using the Hyperbolic Cocentric phantom.</td>
</tr>
<tr>
<td><code>compute_mtf_ffst(phantom, image[, Ntheta])</code></td>
<td>Calculate the MTF using the method described in [2].</td>
</tr>
<tr>
<td><code>compute_mtf_lwkj(phantom, image)</code></td>
<td>Calculate the MTF using the modulated Siemens Star method in [4].</td>
</tr>
<tr>
<td><code>compute_nps_ffst(phantom, A[, B, plot_type])</code></td>
<td>Calculate the noise power spectrum from a unit circle image using the method from [2].</td>
</tr>
<tr>
<td><code>compute_neq_d(phantom, A, B)</code></td>
<td>Calculate the NEQ according to recommendations by [1].</td>
</tr>
<tr>
<td><code>compute_quality(reference, reconstructions)</code></td>
<td>Computes full-reference image quality metrics for each of the reconstructions.</td>
</tr>
</tbody>
</table>

```python
xdesign.metrics.compute_PCC(A, B, masks=None)

Computes the Pearson product-moment correlation coefficients (PCC) for the two images.

**Parameters**

- A, B (ndarray) – The two images to be compared
- masks (list of ndarrays, optional) – If supplied, the data under each mask is computed separately.

**Returns**

- covariances (array, list of arrays)

xdesign.metrics.compute_likeness(A, B, masks)

Predict the likelihood that each pixel in B belongs to a phase based on the histogram of A.

**Parameters**

- A (ndarray)
- B (ndarray)
- masks (list of ndarrays)

**Returns**

- likelihoods (list of ndarrays)

xdesign.metrics.compute_background_ttest(image, masks)

Determine whether the background has significantly different luminance than the other phases.

**Parameters**

- image (ndarray)
- masks (list of ndarrays) – Masks for the background and any other phases. Does not auto-generate the non-background mask because maybe you want to compare only two phases.

**Returns**

- tstat (scalar)
- pvalue (scalar)
xdesign.metrics.compute_mtf (phantom, image)
Approximate the modulation transfer function using the HyperbolicCocentric phantom. Calculate the MTF from
the modulation depth at each edge on the line from (0.5,0.5) to (0.5,1). MTF = (hi-lo)/(hi+lo)

Parameters

- **phantom** (*HyperbolicConcentric*) – Predefined phantom of cocentric rings whose widths
decay parabolically.
- **image** (*ndarray*) – The reconstruction of the above phantom.

Returns

- **wavelength** (*list*) – wavelength in the scale of the original phantom
- **MTF** (*list*) – MTF values
- .. deprecated:: 0.3 – This method rapidly becomes inaccurate at small wavelengths because
  the measurement gets out of phase with the waves due to rounding error. Use another one
  of the MTF functions instead.
- ..seealso:: :ref:`compute_mtf_ffst <compute_mtf_ffst>` :ref:`compute_mtf_lwkj <compute_mtf_lwkj>`

xdesign.metrics.compute_mtf_ffst (phantom, image, Ntheta=4)
Calculate the MTF using the method described in [2].

Parameters

- **phantom** (*UnitCircle*) – Predefined phantom with single circle whose radius is less than
  0.5.
- **image** (*ndarray*) – The reconstruction of the above phantom.
- **Ntheta** (*scalar*) – The number of directions at which to calculate the MTF.

Returns

- **wavenumber** (*ndarray*) – wavelength in the scale of the original phantom
- **MTF** (*ndarray*) – MTF values
- **bin_centers** (*ndarray*) – the center of the bins if Ntheta >= 1
- ..seealso:: :ref:`compute_mtf_lwkj <compute_mtf_lwkj>`

xdesign.metrics.compute_mtf_lwkj (phantom, image)
Calculate the MTF using the modulated Siemens Star method in [4].

Parameters

- **phantom** (*SiemensStar*)
- **image** (*ndarray*) – The reconstruction of the SiemensStar

Returns

- **frequency** (*array*) – The spatial frequency in cycles per unit length
- **M** (*array*) – The MTF values for each frequency
- ..seealso:: :ref:`compute_mtf_ffst <compute_mtf_ffst>`

xdesign.metrics.compute_nps_ffst (phantom, A, B=None, plot_type='frequency')
Calculate the noise power spectrum from a unit circle image using the method from [2].

Parameters

- **phantom** (*UnitCircle*) – The unit circle phantom.
• A (ndarray) – The reconstruction of the above phantom.
• B (ndarray) – The reconstruction of the above phantom with different noise. This second reconstruction enables allows use of trend subtraction instead of zero mean normalization.
• plot_type (string) – ‘histogram’ returns a plot binned by radial coordinate wavenumber ‘frequency’ returns a wavenumber vs wavenumber plot

Returns
• bins – Bins for the radially binned NPS
• counts – NPS values for the radially binned NPS
• X, Y – Frequencies for the 2D frequency plot NPS
• NPS (2Darray) – the NPS for the 2D frequency plot

xdesign.metrics.compute_neq_d(phantom, A, B)
Calculate the NEQ according to recommendations by [1].

Parameters
• phantom (UnitCircle) – The unit circle class with radius less than 0.5
• A (ndarray) – The reconstruction of the above phantom.
• B (ndarray) – The reconstruction of the above phantom with different noise. This second reconstruction enables allows use of trend subtraction instead of zero mean normalization.

Returns
• mu_b – The spatial frequencies
• NEQ – the Noise Equivalent Quanta

class xdesign.metrics.ImageQuality (original, reconstruction, method='')
Bases: object
Store information about image quality.

orig
    numpy.ndarray
recon
    numpy.ndarray
qualities
    list of scalars
maps
    list of numpy.ndarray
scales
    list of scalars

add_quality (quality, scale, maps=None)

Parameters
• quality (scalar, list) – The average quality for the image
• map (array, list of arrays, optional) – the local quality rating across the image
• scale (scalar, list) – the size scale at which the quality was calculated

sort ()
Sorts the qualities by scale
xdesign.metrics.compute_quality(reference, reconstructions, method='MSSSIM', L=1)

Computes full-reference image quality metrics for each of the reconstructions.

Available methods include SSIM [6], MSSSIM [7], VIFp [5], and FSIM [8].

Parameters

- **reference** (array) – the discrete reference image. In a future release, we will determine the best way to compare a continuous domain to a discrete reconstruction.

- **reconstructions** (list of arrays) – A list of discrete reconstructions

- **method** (string, optional) – The quality metric desired for this comparison. Options include: SSIM, MSSSIM, VIFp, FSIM

- **L** (scalar) – The dynamic range of the data. This value is 1 for float representations and $2^{\text{bitdepth}}$ for integer representations.

Returns **metrics** (list of ImageQuality)

xdesign.phantom

Defines an object for simulating X-ray phantoms.

Classes:

<table>
<thead>
<tr>
<th>Phantom(geometry, children, mass_atten)</th>
<th>An object for the purpose of evaluating X-ray imaging methods.</th>
</tr>
</thead>
</table>

class xdesign.phantom.Phantom(geometry=None, children=[], mass_atten=0.0)

An object for the purpose of evaluating X-ray imaging methods.

Phantoms may be hierarchical structures with children that are contained within and/or a parent which contains them. They have two parts: a geometry and properties. The geometry defines the spatial extent over which the properties are valid. Properties are parameters which a Probe uses to measure the Phantom.

All Phantoms must fit within the geometry of their ancestors. Phantoms whose geometry is None act as containers.

- **geometry**

  *Entity* – The spatial boundary of the Phantom; may be None.

- **children**

  A list of Phantoms contained in this Phantom.

- **parent**

  The Phantom containing this Phantom.

- **mass_atten**

  The mass_attenuation of the phantom.

- **population**

  The number of descendents of this phantom.

- **append**(child)

  Add a child to the Phantom.
Only add the child if it is contained within the geometry of its ancestors.

center
Return the centroid of the Phantom.

density
Return the geometric density of the Phantom.

gallery
Return the geometry of the Phantom.

is_leaf
Return whether the Phantom is a leaf node.

pop
Pop the i-th child from the Phantom.

radius
Return the radius of the smallest boundary sphere.

rotate
Rotate around an axis that passes through the given point.

sprinkle
Sprinkle a number of Circle shaped Phantoms around the Phantom. Uses various termination criteria to determine when to stop trying to add circles.

Parameters
- counts \( \text{(int)} \) – The number of circles to be added.
- radius \( \text{(scalar or list)} \) – The radius of the circles to be added.
- gap \( \text{(float, optional)} \) – The minimum distance between circle boundaries. A negative value allows overlapping edges.
- region \( \text{(Entity, optional)} \) – The new circles are confined to this shape. None if the circles are allowed anywhere.
- max_density \( \text{(scalar, optional)} \) – Stops adding circles when the geometric density of the phantom reaches this ratio.
- mass_atten \( \text{(scalar, optional)} \) – A mass attenuation parameter passed to the circles.

Returns counts \( \text{(scalar)} \) – The number of circles successfully added.

translate
Translate the Phantom.

volume
Return the volume of the Phantom

xdesign.phantom.save_phantom(phantom, filename)
Save phantom to file.

xdesign.phantom.load_phantom(filename)
Load phantom from file.

xdesign.plot
Contains functions for visualizing Phantom and ImageQuality metrics.
Functions:

```
sidebyside(p[, size, labels, prop]) Displays the geometry and the discrete property function of the given Phantom side by side.
discrete_phantom(phantom, size[, ratio, ...]) Returns discrete representation of the property function, prop, in the Phantom.
plot_phantom(phantom[, axis, labels, ...]) Plots a Phantom to the given axis.
plot_mesh(mesh[, axis, labels, ...]) Plots a Mesh to the given axis.
plot_polygon(polygon[, axis, alpha, c]) Plots a Polygon to the given axis.
plot_curve(curve[, axis, alpha, c]) Plots a Curve to the given axis.
plot_metrics(imqual) Plots full reference metrics of ImageQuality data.
plot_mtf(faxis, MTF[, labels]) Plots the MTF.
plot_nps(X, Y , NPS) Plots the 2D frequency plot for the NPS.
plot_neq(freq, NEQ) Plots the NEQ.
```

xdesign.plot.plot_phantom (phantom, axis=None, labels=None, c_props=[], c_map=None, i=0) Plots a Phantom to the given axis.

**Parameters**

- **phantom** (*Phantom*) – A phantom to be plotted.
- **axis** (*matplotlib.axis.Axis*) – The axis where the phantom should be plotted. *None* creates a new axis.
- **labels** (*bool, optional*) – *True* : Each Phantom given a unique number.
- **c_props** (*list of str, optional*) – List of Phantom properties to use for colormapping the geometries.
- **c_map** (*function, optional*) – A function which takes the list of prop(s) for a Phantom as input and returns a matplotlib color specifier. [3]

xdesign.plot.plot_mesh (mesh, axis=None, alpha=None, c=None) Plots a Mesh to the given axis.

**Parameters**

- **mesh** (*Mesh*) – A Mesh to plot on the given axis.
- **axis** (*matplotlib.axis.Axis, optional*) – The axis where the Mesh should be plotted. *None* creates a new axis.
- **alpha** (*float, optional*) – The plot opaqueness. 0 is transparent. 1 is opaque.
- **c** (*matplotlib.color, optional*) – The color of the plotted Mesh.

xdesign.plot.plot_polygon (polygon, axis=None, alpha=None, c=None) Plots a Polygon to the given axis.

**Parameters**

- **polygon** (*Polygon*) – A Polygon to plot on the given axis.
- **axis** (*matplotlib.axis.Axis, optional*) – The axis where the Polygon should be plotted. *None* creates a new axis.
- **alpha** (*float, optional*) – The plot opaqueness. 0 is transparent. 1 is opaque.
- **c** (*matplotlib.color, optional*) – The color of the plotted Polygon.
xdesign.plot.plot_curve(curve, axis=None, alpha=None, c=None)
Plots a Curve to the given axis.

Parameters

- curve (Curve) – A Curve to plot on the given axis.
- axis (matplotlib.axis.Axis, optional) – The axis where the Curve should be plotted. None creates a new axis.
- alpha (float, optional) – The plot opaqueness. 0 is transparent. 1 is opaque.
- c (matplotlib.color, optional) – The color of the plotted curve.

xdesign.plot.discrete_phantom(phantom, size, ratio=8, uniform=True, prop='mass_atten')
Returns discrete representation of the property function, prop, in the Phantom. The values of overlapping Phantoms are additive.

Parameters

- phantom (Phantom)
- size (scalar) – The side length in pixels of the resulting square image.
- ratio (scalar, optional) – The antialiasing works by supersampling. This parameter controls how many pixels in the larger representation are averaged for the final representation. e.g. if ratio = 8, then the final pixel values are the average of 64 pixels.
- uniform (boolean, optional) – When set to False, changes the way pixels are averaged from a uniform weights to gaussian weights.
- prop (str, optional) – The name of the property function to discretize

Returns image (numpy.ndarray) – The discrete representation of the Phantom that is size x size.

xdesign.plot.sidebyside(p, size=100, labels=None, prop='mass_atten')
Displays the geometry and the discrete property function of the given Phantom side by side.

xdesign.plot.multiroll(x, shift, axis=None)
Roll an array along each axis.

Parameters

- x (array_like) – Array to be rolled.
- shift (sequence of int) – Number of indices by which to shift each axis.
- axis (sequence of int, optional) – The axes to be rolled. If not given, all axes is assumed, and len(shift) must equal the number of dimensions of x.

Returns y (numpy array, with the same type and size as x) – The rolled array.

Notes

The length of x along each axis must be positive. The function does not handle arrays that have axes with length 0.

See also:

numpy.roll()
Example

Here’s a two-dimensional array:

```python
>>> x = np.arange(20).reshape(4,5)
>>> x
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

Roll the first axis one step and the second axis three steps:

```python
>>> multiroll(x, [1, 3])
array([[17, 18, 19, 15, 16],
       [ 2,  3,  4,  0,  1],
       [ 7,  8,  9,  5,  6],
       [12, 13, 14, 10, 11]])
```

That’s equivalent to:

```python
>>> np.roll(np.roll(x, 1, axis=0), 3, axis=1)
array([[17, 18, 19, 15, 16],
       [ 2,  3,  4,  0,  1],
       [ 7,  8,  9,  5,  6],
       [12, 13, 14, 10, 11]])
```

Not all the axes must be rolled. The following uses the `axis` argument to roll just the second axis:

```python
>>> multiroll(x, [2], axis=[1])
array([[ 3,  4,  0,  1,  2],
       [ 8,  9,  5,  6,  7],
       [13, 14, 10, 11, 12],
       [18, 19, 15, 16, 17]])
```

which is equivalent to:

```python
>>> np.roll(x, 2, axis=1)
array([[ 3,  4,  0,  1,  2],
       [ 8,  9,  5,  6,  7],
       [13, 14, 10, 11, 12],
       [18, 19, 15, 16, 17]])
```

References


`xdesign.plot.plot_metrics(imqual)`

Plots full reference metrics of ImageQuality data.

Parameters

- `imqual` (ImageQuality) – The data to plot.

References

Colors taken from this gist <https://gist.github.com/thriveth/8560036>
XDesign Documentation, Release 0.3

xdesign.plot.plot_mtf (faxis, MTF, labels=None)
   Plots the MTF. Returns the figure reference.

xdesign.plot.plot_nps (X, Y, NPS)
   Plots the 2D frequency plot for the NPS. Returns the figure reference.

xdesign.plot.plot_neq (freq, NEQ)
   Plots the NEQ. Returns the figure reference.

xdesign.plot.plot_histograms (images, masks=None, thresh=0.025)
   Plots the normalized histograms for the pixel intensity under each mask.

   Parameters
   
   - **images** (list of ndarrays, ndarray) – image(s) for comparing histograms.
   - **masks** (list of ndarrays, float, optional) – If supplied, the data under each mask is plotted separately.
   - **strict** (boolean) – If true, the mask takes values >= only. If false, the mask takes all values > 0.

Examples

This section contains Jupyter Notebooks examples for various XDesign functions.

To run these examples in a notebook install Jupyter and run the notebooks from their source.

Simple How-to Explaining Phantoms

Demonstrate simple basic custom phantom and sinogram generation with XDesign.

In [1]:
   import matplotlib.pyplot as plt
   import numpy as np
   from xdesign import *

   'polytope' failed to import `cvxopt.glpk`.
   Will use `scipy.optimize.linprog`.

Phantom creation

Create various Phantoms and assign attenuation values to each.

In [2]:
   head = Phantom(geometry=Circle(Point([0.5, 0.5]), radius=0.5))
   head.mass_atten = 1.0
   print ('Head looks like this: {}
         '.format(repr(head)))

   eyeL = Phantom(geometry=Circle(Point([0.3, 0.5]), radius=0.1))
   eyeL.mass_atten = 1.0
   print ('Left eye looks like this: {}
         '.format(repr(eyeL)))

   eyeR = Phantom(geometry=Circle(Point([0.7, 0.5]), radius=0.1))
   eyeR.mass_atten = 1.0
   print ('Right eye looks like this: {}
         '.format(repr(eyeR)))

   mouth = Phantom(geometry=Triangle(Point([0.2, 0.7]), Point([0.5, 0.8]), Point([0.8, 0.7])))
   mouth.mass_atten = -1.0
   print ('Mouth looks like this: {}
         '.format(repr(mouth)))
Head looks like this: Phantom(geometry=Circle(center=Point([0.5, 0.5]), radius=0.5), children=[], mass_atten=1.0)
Left eye looks like this: Phantom(geometry=Circle(center=Point([0.29999999999999999, 0.5]), radius=0.1), children=[], mass_atten=1.0)
Right eye looks like this: Phantom(geometry=Circle(center=Point([0.69999999999999996, 0.5]), radius=0.1), children=[], mass_atten=1.0)
Mouth looks like this: Phantom(geometry=Triangle(Point([0.20000000000000001, 0.69999999999999996]), Point([0.5, 0.80000000000000004]), Point([0.80000000000000004, 0.69999999999999996])), children=[], mass_atten=-1.0)

Collect the phantoms together by making the eyes and mouth children of the head Phantom.

In [3]: head.append(eyeL)
   head.append(eyeR)
   head.append(mouth)
   print('Head looks like this: {}

'.format(repr(head)))

Head looks like this: Phantom(geometry=Circle(center=Point([0.5, 0.5]), radius=0.5), children=Phantom(children=[Phantom(geometry=Circle(center=Point([0.69999999999999996, 0.5]), radius=0.1), children=[])], mass_atten=-1.0), mass_atten=1.0)

Viewing phantom geometry and properties

Plot the Phantom geometry and properties with a colorbar.

In [4]: fig = plt.figure(figsize=(7, 3), dpi=600)
   
   # plot geometry
   axis = fig.add_subplot(121, aspect='equal')
   plt.grid('on')
   plt.gca().invert_yaxis()
   plot_phantom(head, axis=axis, labels=False)

   # plot property
   plt.subplot(1, 2, 2)
   im = plt.imshow(discrete_phantom(head, 100, prop='mass_atten'), interpolation='none', cmap=plt.cm.inferno)

   # plot colorbar
   fig.subplots_adjust(right=0.8)
   cbar_ax = fig.add_axes([0.85, 0.16, 0.05, 0.7])
   fig.colorbar(im, cax=cbar_ax)

   # save the figure
   plt.savefig('Shepp_sidebyside.png', dpi=600,
               orientation='landscape', papertype=None, format=None,
               transparent=True, bbox_inches='tight', pad_inches=0.0,
               frameon=False)

   plt.show()
Simulate data acquisition

Simulate data acquisition for parallel beam around 180 degrees.

In [5]: sx, sy = 100, 100
   sino, prb = sinogram(sx, sy, head)

Plot the sinogram.

In [6]: plt.figure(figsize=(8, 8))
   plt.imshow(np.reshape(sino, (sx, sy)), cmap='inferno', interpolation='nearest')
   plt.savefig('Shepp_sinogram.png', dpi=600,
   orientation='landscape', papertype=None, format=None,
   transparent=True, bbox_inches='tight', pad_inches=0.0,
   frameon=False)
   plt.show()
Standard Test Patterns

Generates sidebyside plots of all the standard test patterns in xdesign.

```python
In [1]: from xdesign import *
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

`polytope` failed to import `cvxopt.glpk`.
Will use `scipy.optimize.linprog`.

In [2]: p = SlantedSquares(count=16, angle=5/360*2*np.pi, gap=0.01)
sidebyside(p)
```
plt.savefig('SlantedSquares_sidebyside.png', dpi='figure',
    orientation='landscape', papertype=None, format=None,
    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)
plt.show()

In [3]: h = HyperbolicConcentric()
sidebyside(h)
plt.savefig('HyperbolicConcentric_sidebyside.png', dpi='figure',
    orientation='landscape', papertype=None, format=None,
    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)
plt.show()

In [4]: u = UnitCircle(radius=0.4, mass_atten=1)
sidebyside(u)
plt.savefig('UnitCircle_sidebyside.png', dpi='figure',
    orientation='landscape', papertype=None, format=None,
    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)
plt.show()
In [5]: d = DynamicRange(steps=16, jitter=True)
sidebyside(d)
plt.savefig('DynamicRange_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()

In [6]: l = DogaCircles(n_sizes=8, size_ratio=0.5, n_shuffles=0)
l.rotate(np.pi/2, Point([0.5, 0.5]))
sidebyside(l)
plt.savefig('DogaCircles_sidebyside.png', dpi='figure',
            orientation='landscape', papertype=None, format=None,
            transparent=True, bbox_inches='tight', pad_inches=0.0,
            frameon=False)
plt.show()
In [7]: s = SiemensStar(32)
sidebyside(s)
plt.savefig('SiemensStar_sidebyside.png', dpi='figure',
orientation='landscape', papertype=None, format=None,
transparent=True, bbox_inches='tight', pad_inches=0.0,
frameon=False)
plt.show()

In [8]: fig = plt.figure(figsize=(8, 6), dpi=600)
   
gs1 = gridspec.GridSpec(3, 4)
gs1.update(wspace=0.4, hspace=0.4)  # set the spacing between axes.
phantoms = [l, d, u, h, p, s]
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
for i in range(0, len(phantoms)):
   axis = plt.subplot(gs1[2*i], aspect=1)
   plt.grid('on')
   plt.gca().invert_yaxis()
   plot_phantom(phantoms[i], axis=axis)
   plt.title('('+ letters[i] +')')
   plt.subplot(gs1[2*i+1], aspect=1)
   plt.imshow(discrete_phantom(phantoms[i], 200), cmap='inferno')

plt.savefig('standard_patterns.png', dpi='figure',
orientation='landscape', papertype=None, format=None,
# Figure for TomoBank Poster

```python
fig = plt.figure(figsize=(5, 4), dpi=600)
gs1 = gridspec.GridSpec(3, 4)
gs1.update(wspace=0.1, hspace=0.1)  # set the spacing between axes.
phantoms = ['l', 'd', 'u', 'h', 'p', 's']
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

for i in range(0, len(phantoms)):
    axis = plt.subplot(gs1[2*i], aspect=1)
    plt.grid('on')
    plt.gca().invert_yaxis()
    plt.gca().axes.set_xticklabels([])
    plt.gca().axes.set_yticklabels([])
    plot_phantom(phantoms[i], axis=axis)
    plt.subplot(gs1[2*i+1], aspect=1)
    plt.imshow(discrete_phantom(phantoms[i], 200), cmap='inferno')
    plt.gca().axes.set_xticklabels([])
    plt.gca().axes.set_yticklabels([])

plt.savefig('TomoBank_standard_patterns.png', dpi='figure', orientation='landscape', papertype=None, format=None,

```

5.2. Examples
Parameterized Phantom Generation

Demonstrates how a parameterized function can generate 4 different phantoms from the same parameterized class.

```python
In [1]: from xdesign import *
    import matplotlib.pyplot as plt
    import numpy as np
    import matplotlib.gridspec as gridspec

SIZE = 600

'polytope' failed to import 'cvxopt.glpk'.
Will use 'scipy.optimize.linprog'.

In [3]: np.random.seed(0) # random seed for repeatability
    pl = Foam(size_range=[0.05, 0.01], gap=0, porosity=1)
    dl = discrete_phantom(pl, SIZE)
    plt.imshow(d1, cmap='viridis')
    plt.show()
```

/home/beams/B242827/miniconda3/lib/python3.5/site-packages/xdesign-0.1.0.dev0+c571565207c0c2e6731d5f5
In [4]: np.random.seed(0)  # random seed for repeatability
    p2 = Foam(size_range=[0.07, 0.01], gap=0, porosity=0.75)
    d2 = discrete_phantom(p2, SIZE)
    plt.imshow(d2, cmap='viridis')
    plt.show()

/home/beams/B242827/miniconda3/lib/python3.5/site-packages/xdesign-0.1.0.dev0+b8b68c0fbc3d9d4306f0af588c0d7308858eb7f1-py3.5.egg/xdesign/phantom.py:226: RuntimeWarning: Reached termination criteria of 200 attempts before adding all of the circles.

In [5]: np.random.seed(0)  # random seed for repeatability
    p3 = Foam(size_range=[0.1, 0.01], gap=0, porosity=0.5)
    d3 = discrete_phantom(p3, SIZE)
    plt.imshow(d3, cmap='viridis')
Create a composite figure of all four discrete phantoms.
In [38]: fig = plt.figure(dpi=600)
gs1 = gridspec.GridSpec(2, 2)
gs1.update(wspace=0.1, hspace=0.1)  # set the spacing between axes.
plt.subplot(gs1[0])
plt.title('(a)')
plt.axis('off')
plt.imshow(d1, interpolation='none', cmap=plt.cm.gray)
plt.subplot(gs1[1])
plt.title('(b)')
plt.axis('off')
plt.imshow(d2, interpolation='none', cmap=plt.cm.gray)
plt.subplot(gs1[2])
plt.title('(c)')
plt.axis('off')
plt.imshow(d3, interpolation='none', cmap=plt.cm.gray)
plt.subplot(gs1[3])
plt.title('(d)')
plt.axis('off')
plt.imshow(d4, interpolation='none', cmap=plt.cm.gray)
fig.set_size_inches(6, 6)
plt.savefig('Foam_parameterized.png', dpi='figure',
orientation='landscape', papertype=None, format=None,
transparent=True, bbox_inches='tight', pad_inches=0.0,
frameon=False)
plt.show()
Demonstrate the use of the no-reference metrics: noise power spectrum (NPS), modulation transfer function (MTF), and noise equivalent quanta (NEQ).

In [1]:
```
from xdesign import *
import tomopy
import numpy as np
import matplotlib.pylab as plt
```

`polytope` failed to import `cvxopt.glpk`.
Will use `scipy.optimize.linprog`.
WARNING:dxchange.reader:astropy module not found

Generate a UnitCircle test phantom. For the MTF, the radius must be less than 0.5, otherwise the circle touches the edges of the field of view.

In [2]:
```
p = UnitCircle(mass_atten=4, radius=0.35)
sidebyside(p, 100)
plt.show()
```
Generate two sinograms and reconstruct. Noise power spectrum and Noise Equivalent Quanta are meaningless without noise so add some poisson noise to the reconstruction process with the `noise` argument. Collecting two sinograms allows us to isolate the noise by subtracting out the circle.

```
In [3]: np.random.seed(0)
    sinoA, _ = sinogram(100, 100, p, noise=0.1)
sinoB, _ = sinogram(100, 100, p, noise=0.1)
    theta = np.arange(0, np.pi, np.pi / 100.)
    recA = tomopy.recon(np.expand_dims(sinoA, 1), theta, algorithm='gridrec', center=(sinoA.shape[-1]-1)/2.)
    recB = tomopy.recon(np.expand_dims(sinoB, 1), theta, algorithm='gridrec', center=(sinoB.shape[-1]-1)/2.)
```

Take a look at the two noisy reconstructions.

```
In [4]: plt.imshow(recA[0], cmap='inferno', interpolation="none")
    plt.colorbar()
    plt.savefig('UnitCircle_noise0.png', dpi=600,
                orientation='landscape', papertype=None, format=None,
                transparent=True, bbox_inches='tight', pad_inches=0.0,
                frameon=False)
```

5.2. Examples
Calculate MTF

Use Friedman et al’s method for computing the MTF. You can separate the MTF into multiple directions or average them all together using the $N_{\text{theta}}$ argument.

In [6]: mtf_freq, mtf_value, labels = compute_mtf_ffst(p, recA[0], Ntheta=4)
The MTF is really a symmetric function around zero frequency, so usually people just show the positive portion. Sometimes, there is a peak at the higher spatial frequencies instead of the MTF approaching zero. This is probably because of aliasing noise content with frequencies higher than the Nyquist frequency.

In [7]: plot_mtf(mtf_freq, mtf_value, labels)
   plt.gca().set_xlim([0, 50])  # hide negative portion of MTF
   plt.show()

You can also use a Siemens Star to calculate the MTF using a fitted sinusoidal method instead of the slanted edges that the above method uses.

In [8]: s = SiemensStar(n_sectors=32, center=Point([0.5, 0.5]), radius=0.5)
   d = sidebyside(s, 100)
   plt.show()

Here we are using the discreet verison of the phantom (without noise), so we are only limited by the resolution of the
image.

In [9]: mtf_freq, mtf_value = compute_mtf_lwkj(s, d)
In [10]: plot_mtf(mtf_freq, mtf_value, labels=None)
   plt.gca().set_xlim([0,50])  # hide portion of MTF beyond Nyquist frequency
   plt.show()

Calculate NPS

Calculate the radial or 2D frequency plot of the NPS.

In [11]: X, Y, NPS = compute_nps_ffst(p, recA[0], plot_type='frequency', B=recB[0])
In [12]: plot_nps(X, Y, NPS)
   plt.show()
In [13]: bins, counts = compute_nps_ffst(p, recA[0], plot_type='histogram', B=recB[0])

In [14]: plt.figure()
   plt.bar(bins, counts)
   plt.xlabel('spatial frequency [cycles/length]')
   plt.title('Noise Power Spectrum')
   plt.show()
Calculate NEQ

In [15]: freq, NEQ = compute_neq_d(p, recA[0], recB[0])

In [16]: plt.figure()
   plt.plot(freq.flatten(), NEQ.flatten())
   plt.xlabel('spatial frequency [cycles/length]')
   plt.title('Noise Equivalent Quanta')
   plt.show()
Demonstrate the use of full reference metrics by comparing the reconstruction of a simulated phantom using SIRT, ART, and MLEM.

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage.exposure import adjust_gamma, rescale_intensity
from xdesign import *

def rescale(reconstruction, hi):
    I = rescale_intensity(reconstruction, out_range=(0., 1.))
    return adjust_gamma(I, 1, hi)

`polytope` failed to import `cvxopt.glpk`.
Will use `scipy.optimize.linprog`.

Generate a phantom

Use one of XDesign’s various pre-made and procedurally generated phantoms.

```python
np.random.seed(0)
soil_like_phantom = Soil()

/home/beams0/B242827/Documents/xdesign/xdesign/phantom.py:317: RuntimeWarning: Reached termination criteria of 200 attempts before adding all of the circles.
kTERM_CRIT), RuntimeWarning)

Generate a figure showing the phantom and save the discretized ground truth map for later.
In [3]: discrete = sidebyside(soil_like_phantom, 100)

    plt.savefig('Soil_sidebyside.png', dpi='figure',
    orientation='landscape', papertype=None, format=None,
    transparent=True, bbox_inches='tight', pad_inches=0.0,
    frameon=False)

    plt.show()

Simulate data acquisition

Use the built in sinogram function to simulate data acquisition for parallel beam around 180 degrees.

In [4]: sx, sy = 100, 100
       sino, prb = sinogram(sx, sy, soil_like_phantom)

In [5]: plt.imshow(sino, cmap='viridis', interpolation='nearest')
       plt.show()
Reconstruct

Reconstruct the phantom using 3 different techniques: ART, SIRT, and MLEM.

In [6]: hi = 1  # highest expected value in reconstruction (for rescaling)
    niter = 10  # number of iterations

    init = 1e-12 * np.ones((sx, sy))
    rec_art = art(prb, sino, init, niter)
    rec_art = rescale(np.rot90(rec_art)[::-1], hi)

    init = 1e-12 * np.ones((sx, sy))
    rec_sirt = sirt(prb, sino, init, niter)
    rec_sirt = rescale(np.rot90(rec_sirt)[::-1], hi)

    init = 1e-12 * np.ones((sx, sy))
    rec_mlem = mlem(prb, sino, init, niter)
    rec_mlem = rescale(np.rot90(rec_mlem)[::-1], hi)

[##########] 100.00%
[##########] 100.00%
[##########] 100.00%

In [7]: plt.figure(figsize=(12,4))
    plt.subplot(131)
    plt.imshow(rec_art, cmap='gray', interpolation='none')
    plt.title('ART')
    plt.subplot(132)
    plt.imshow(rec_sirt, cmap='gray', interpolation='none')
    plt.title('SIRT')
    plt.subplot(133)
    plt.imshow(rec_mlem, cmap='gray', interpolation='none')
    plt.title('MLEM')
    plt.show()
Quality Metrics

Compute local quality for each reconstruction using MS-SSIM, a convolution based quality metric.

```python
In [8]: metrics = compute_quality(discrete, [rec_art, rec_sirt, rec_mlem], method="MSSSIM")
```

Plot the average quality at each level of detail for each reconstruction in a line plot. Then display the local quality map for each reconstruction to see why certain reconstructions are ranked higher than others.

In this case, it’s clear that ART is ranking higher than MLEM and SIRT at smaller scales because the small dark particles are visible; whereas for SIRT and MLEM they are unresolved. We can also see that the large yellow circles have a more accurately rendered luminance for SIRT and MLEM which is what causes these methods to be ranked higher at larger scales.

```python
In [9]: plot_metrics(metrics)
plt.show()
```
Appendix

In [13]: # Figure for TomoBank Poster
    plt.figure(figsize=(9,9), dpi=600)
    plt.subplot(2,2,1)
    plt.imshow(discrete)
    plt.title('XDesign Discrete', fontsize=20)
    plt.subplot(2,2,2)
    plt.imshow(rec_art)
    plt.title('ART', fontsize=20)
    plt.subplot(2,2,3)
    plt.imshow(rec_mlem)
    plt.title('MLEM', fontsize=20)
    plt.subplot(2,2,4)
    plt.imshow(rec_sirt)
    plt.title('SIRT', fontsize=20)
    plt.viridis()
    plt.savefig('TomoBank_discrete.png', dpi='figure',
                orientation='landscape', papertype=None, format=None,
                transparent=True, bbox_inches='tight', pad_inches=0.0,
                frameon=False)
    plt.show()
In [1]:
import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt
from xdesign import *
from skimage.exposure import adjust_gamma, rescale_intensity

def rescale(reconstruction, hi):
    I = rescale_intensity(reconstruction, out_range=(0., 1.))
    return adjust_gamma(I, 1, hi)

`polytope` failed to import `cvxopt.glpk`. Will use `scipy.optimize.linprog`.

5.2. Examples
In [ ]: wet = WetCircles()
sidebyside(wet, size=200)
plt.savefig('Wet_sidebyside.png', dpi='figure', 
orientation='landscape', papertype=None, format=None, 
transparent=True, bbox_inches='tight', pad_inches=0.0, 
frameon=False)
plt.show(block=True)

In [ ]: sx, sy = 100, 100
sino, prb = sinogram(sx, sy, wet)
In [ ]: plt.figure(figsiz=(8, 8))
plt.imshow(np.reshape(sino, (sx, sy)), cmap='gray', interpolation='nearest')
plt.show(block=True)

In [10]: hi = 1
niter = 20
# Reconstruct object.
init = 1e-12 * np.ones((sx, sy))
rec_art = art(prb, sino, init, niter)
rec_art = rescale(np.rot90(rec_art)[:,::-1], hi)
plt.figure(figsiz=(8, 8))
plt.imshow(rec_art, cmap='gray', interpolation='none')
plt.title('ART')

init = 1e-12 * np.ones((sx, sy))
rec_sirt = sirt(prb, sino, init, niter)
rec_sirt = rescale(np.rot90(rec_sirt)[:,::-1], hi)
plt.figure(figsiz=(8, 8))
plt.imshow(rec_sirt, cmap='gray', interpolation='none')
plt.title('SIRT')

init = 1e-12 * np.ones((sx, sy))
rec_mlem = mlem(prb, sino, init, niter)
rec_mlem = rescale(np.rot90(rec_mlem)[:,::-1], hi)
plt.figure(figsiz=(8, 8))
plt.imshow(rec_mlem, cmap='gray', interpolation='none')
plt.title('MLEM')
plt.show()
References

Credits

We kindly request that you cite the following article(s) [A1] if you use XDesign.

[2] Saul N. Friedman, George S. K. Fung, Jeffrey H. Siewerdsen, and Benjamin M. W. Tsui. A simple approach to measure computed tomography (ct) modulation transfer function (mtf) and noise-power spectrum (nps) using the american college of radiology (acr) accreditation phantom. *Medical Physics*, 40(5):, 2013. URL: http://scitation.aip.org/content/aapm/journal/medphys/40/5/10.1118/1.4800795, doi:http://dx.doi.org/10.1118/1.4800795.


x
xdesign, 65
xdesign.acquisition, 11
xdesign.algorithms, 13
xdesign.constants, 24
xdesignformats, 24
xdesign.geometry, 14
xdesign.material, 24
xdesign.metrics, 27
xdesign.phantom, 31
xdesign.plot, 32
A
add_quality() (xdesign.metrics.ImageQuality method), 30
angle (xdesign.material.SlantedSquares attribute), 26
angle_scan() (in module xdesign.acquisition), 13
angleogram() (in module xdesign.acquisition), 13
append() (xdesign.geometry.Mesh method), 23
append() (xdesign.phantom.Phantom method), 31
area (xdesign.geometry.Circle attribute), 17
area (xdesign.geometry.Polygon attribute), 19
art() (in module xdesign.algorithms), 14
atom_concentration (xdesign.material.Material attribute), 25
atomic_form_factor (xdesign.material.Material attribute), 25
B
Beam (class in xdesign.acquisition), 12
bounds (xdesign.geometry.Polygon attribute), 19
bounds (xdesign.geometry.Rectangle attribute), 21
bounds (xdesign.geometry.Square attribute), 22
bounds (xdesign.geometry.Triangle attribute), 20
C
center (xdesign.geometry.Circle attribute), 16
center (xdesign.geometry.Polygon attribute), 19
center (xdesign.phantom.Phantom attribute), 32
children (xdesign.phantom.Phantom attribute), 31
Circle (class in xdesign.geometry), 16
circumference (xdesign.geometry.Circle attribute), 17
collision() (xdesign.geometry.Circle method), 17
collision() (xdesign.geometry.Entity method), 15
collision() (xdesign.geometry.Line method), 18
collision() (xdesign.geometry.Mesh method), 23
collision() (xdesign.geometry.Point method), 16
collision() (xdesign.geometry.Polygon method), 19
collision() (xdesign.geometry.Rectangle method), 21
collision() (xdesign.geometry.Square method), 22
collision() (xdesign.geometry.Triangle method), 20
distance() (xdesign.acquisition.Beam method), 12
distance() (xdesign.geometry.Circle method), 17
distance() (xdesign.geometry.Entity method), 15
distance() (xdesign.geometry.Point method), 16
distance() (xdesign.geometry.Polygon method), 19
distance() (xdesign.geometry.Rectangle method), 21
distance() (xdesign.geometry.Square method), 22
distance() (xdesign.geometry.Triangle method), 20
discrete_phantom() (in module xdesign.plot), 34
density (xdesign.phantom.Phantom attribute), 32
diameter (xdesign.geometry.Circle attribute), 17
dim (xdesign.geometry.Circle attribute), 17
dim (xdesign.geometry.Entity attribute), 15
dim (xdesign.geometry.Line attribute), 18
dim (xdesign.geometry.Mesh attribute), 23
dim (xdesign.geometry.Point attribute), 16
dim (xdesign.geometry.Polygon attribute), 19
dim (xdesign.geometry.Rectangle attribute), 21
dim (xdesign.geometry.Square attribute), 22
dim (xdesign.geometry.Triangle attribute), 20
distance() (xdesign.acquisition.Beam method), 12
distance() (xdesign.geometry.Line method), 18
distance() (xdesign.geometry.Mesh method), 23
distance() (xdesign.geometry.Point method), 16
distance() (xdesign.geometry.Polygon method), 19
distance() (xdesign.geometry.Rectangle method), 21
distance() (xdesign.geometry.Square method), 22
distance() (xdesign.geometry.Triangle method), 20
DogaCircles (class in xdesign.material), 26
DynamicRange (class in xdesign.material), 26
edges (xdesign.geometry.Polygon attribute), 19
edges (xdesign.geometry.Rectangle attribute), 21
edges (xdesign.geometry.Triangle attribute), 22
electron_density (xdesign.material.Material attribute), 25
Electronics (class in xdesign.material), 27
Entity (class in xdesign.geometry), 14
FiberComposite (class in xdesign.material), 27
Foil (class in xdesign.material), 27
Gap (class in xdesign.material), 26
geometry (xdesign.phantom.Phantom attribute), 31, 32
gap (xdesign.material.SlantedSquares attribute), 26
gap (xdesign.material.SlantedSquares attribute), 27
geometry (xdesign.phantom.Phantom attribute), 31, 32
half_space (xdesign.acquisition.Beam attribute), 12
horizontal (xdesign.geometry.Line attribute), 18
HyperbolicConcentric (class in xdesign.material), 25
ImageQuality (class in xdesign.metrics), 30
import_triangle() (xdesign.geometry.Mesh method), 23
intercept() (xdesign.geometry.Line method), 18
is_leaf (xdesign.phantom.Phantom attribute), 32
length (xdesign.geometry.Line attribute), 18
Line (class in xdesign.geometry), 17
mass_atten (xdesign.phantom.Phantom attribute), 31
mass_ratio() (xdesign.material.Material method), 25
Material (class in xdesign.material), 25
measure() (xdesign.acquisition.Probe method), 12
Mesh (class in xdesign.geometry), 23
Metal (class in xdesign.material), 27
midpoint() (xdesign.geometry.Circle method), 17
midpoint() (xdesign.geometry.Entity method), 15
midpoint() (xdesign.geometry.Line method), 18
midpoint() (xdesign.geometry.Mesh method), 23
midpoint() (xdesign.geometry.Point method), 16
midpoint() (xdesign.geometry.Polygon method), 20
midpoint() (xdesign.geometry.Rectangle method), 22
midpoint() (xdesign.geometry.Square method), 23
midpoint() (xdesign.geometry.Triangle method), 21
np() (in module xdesign.algorithms), 14
multiroll() (in module xdesign.plot), 34
n_levels (xdesign.material.SlantedSquares attribute), 27
norm (xdesign.geometry.Point attribute), 16
normal (xdesign.geometry.Line attribute), 18
number_of_elements() (xdesign.material.Material method), 25
numpy (xdesign.geometry.Line attribute), 18
numpy (xdesign.geometry.Polygon attribute), 20
numpy (xdesign.geometry.Rectangle attribute), 22
numpy (xdesign.geometry.Square attribute), 23
numpy (xdesign.geometry.Triangle attribute), 21
orig (xdesign.metrics.ImageQuality attribute), 30
p1 (xdesign.acquisition.Beam attribute), 12
p1 (xdesign.geometry.Line attribute), 17
p2 (xdesign.acquisition.Beam attribute), 12
p2 (xdesign.geometry.Line attribute), 18
parent (xdesign.phantom.Phantom attribute), 31
patch (xdesign.geometry.Circle attribute), 17
patch (xdesign.geometry.Polygon attribute), 20
patch (xdesign.geometry.Rectangle attribute), 22
patch (xdesign.geometry.Square attribute), 23
patch (xdesign.geometry.Triangle attribute), 21
perimeter (xdesign.geometry.Polygon attribute), 20
perimeter (xdesign.geometry.Rectangle attribute), 22
perimeter (xdesign.geometry.Square attribute), 23
perimeter (xdesign.geometry.Triangle attribute), 21
Phantom (class in xdesign.phantom), 31
photoelectric_absorption (xdesign.material.Material attribute), 25
photoelectric_cross_section (xdesign.material.Material attribute), 25
plot_curve() (in module xdesign.plot), 33
plot_histograms() (in module xdesign.plot), 36
plot_mesh() (in module xdesign.plot), 33
plot_metrics() (in module xdesign.plot), 35
plot_mtf() (in module xdesign.plot), 35
plot_neq() (in module xdesign.plot), 36
plot_nps() (in module xdesign.plot), 36
plot_phantom() (in module xdesign.plot), 33
plot_polygon() (in module xdesign.plot), 33
Point (class in xdesign.geometry), 16
points (xdesign.geometry.Line attribute), 18
Polygon (class in xdesign.geometry), 19
pop() (xdesign.geometry.Mesh method), 23
pop() (xdesign.phantom.Phantom method), 32
population (xdesign.phantom.Phantom attribute), 31
Probe (class in xdesign.acquisition), 12
qualities (xdesign.metrics.ImageQuality attribute), 30
radii (xdesign.material.DogaCircles attribute), 26
radii (xdesign.material.HyperbolicConcentric attribute),
   26
radius (xdesign.geometry.Circle attribute), 16
radius (xdesign.geometry.Polygon attribute), 20
radius (xdesign.geometry.Rectangle attribute), 22
radius (xdesign.geometry.Square attribute), 23
radius (xdesign.geometry.Triangle attribute), 21
radius (xdesign.phantom.Phantom attribute), 32
radius_per_level (xdesign.material.SlantedSquares attribute), 26
raster_scan() (in module xdesign.acquisition), 13
ratio (xdesign.material.SiemensStar attribute), 27
recon (xdesign.metrics.ImageQuality attribute), 30
record() (xdesign.acquisition.Probe method), 12
Rectangle (class in xdesign.geometry), 21
reduced_energy_ratio (xdesign.material.Material attribute), 25
refractive_index (xdesign.material.Material attribute), 25
rotate() (xdesign.acquisition.Beam method), 12
rotate() (xdesign.geometry.Circle method), 17
rotate() (xdesign.geometry.EntityManager), 18
rotate() (xdesign.geometry.Mesh method), 23
rotate() (xdesign.geometry.Point method), 16
rotate() (xdesign.geometry.Polygon method), 20
rotate() (xdesign.geometry.Rectangle method), 22
rotate() (xdesign.geometry.Square method), 23
rotate() (xdesign.geometry.Triangle method), 21
rotate() (xdesign.phantom.Phantom method), 32
scale() (xdesign.geometry.Circle method), 17
scale() (xdesign.geometry.Entity method), 15
scale() (xdesign.geometry.Line method), 18
scale() (xdesign.geometry.Mesh method), 24
scale() (xdesign.geometry.Point method), 16
scale() (xdesign.geometry.Polygon method), 20
scale() (xdesign.geometry.Rectangle method), 22
scale() (xdesign.geometry.Square method), 23
scale() (xdesign.geometry.Triangle method), 21
scale() (xdesign.metrics.ImageQuality attribute), 30
side_length (xdesign.material.SlantedSquares attribute), 26
sidebyside() (in module xdesign.plot), 34
SiemensStar (class in xdesign.material), 27
sinogram() (in module xdesign.acquisition), 12
sirt() (in module xdesign.algorithms), 14
size (xdesign.acquisition.Beam attribute), 12
skip (xdesign.acquisition.Beam attribute), 12
SlantedSquares (class in xdesign.material), 26
slope (xdesign.geometry.Line attribute), 18
SoftBiomaterial (class in xdesign.material), 27
sort() (xdesign.metrics.ImageQuality method), 30
sprinkle() (xdesign.phantom.Phantom method), 32
Square (class in xdesign.geometry), 22
squares_per_level (xdesign.material.SlantedSquares attribute), 26
standard (xdesign.geometry.Line attribute), 19
stream() (in module xdesign.algorithms), 14
tangent (xdesign.geometry.Line attribute), 19
translate() (xdesign.acquisition.Beam method), 12
translate() (xdesign.acquisition.Probe method), 12
translate() (xdesign.geometry.Circle method), 17
translate() (xdesign.geometry.EntityManager), 16
translate() (xdesign.geometry.Line method), 19
translate() (xdesign.geometry.Mesh method), 24
translate() (xdesign.geometry.Point method), 16
translate() (xdesign.geometry.Polygon method), 20
translate() (xdesign.geometry.Rectangle method), 22
translate() (xdesign.geometry.Square method), 23
translate() (xdesign.geometry.Triangle method), 21
translate() (xdesign.phantom.Phantom method), 32
Triangle (class in xdesign.geometry), 20
UnitCircle (class in xdesign.material), 27
update_progress() (in module xdesign.algorithms), 14
vertical (xdesign.geometry.Line attribute), 19
vertices (xdesign.geometry.Polygon attribute), 19
volume (xdesign.phantom.Phantom attribute), 32
W
WetCircles (class in xdesign.material), 27
widths (xdesign.material.HyperbolicConcentric attribute), 26

X
x (xdesign.geometry.Point attribute), 16
x (xdesign.material.DogaCircles attribute), 26
xdesign (module), 36, 65
xdesign.acquisition (module), 11
xdesign.algorithms (module), 13
xdesign.constants (module), 24
xdesign.formats (module), 24
xdesign.geometry (module), 14
xdesign.material (module), 24
xdesign.metrics (module), 27
xdesign.phantom (module), 31
xdesign.plot (module), 32
XDesignDefault (class in xdesign.material), 25
xintercept (xdesign.geometry.Line attribute), 19

Y
y (xdesign.geometry.Point attribute), 16
y (xdesign.material.DogaCircles attribute), 26
yintercept (xdesign.geometry.Line attribute), 19

Z
z (xdesign.geometry.Point attribute), 16