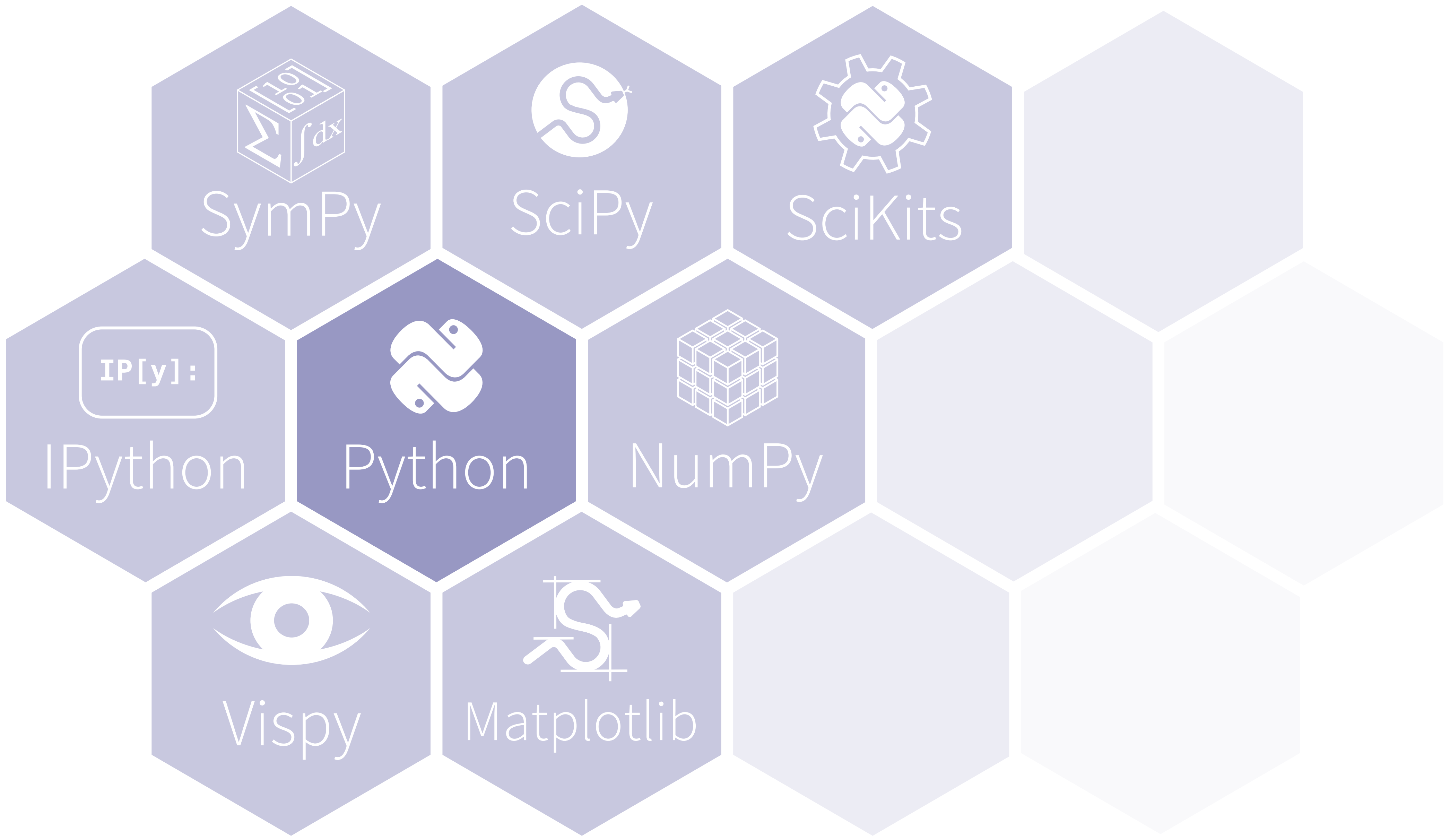


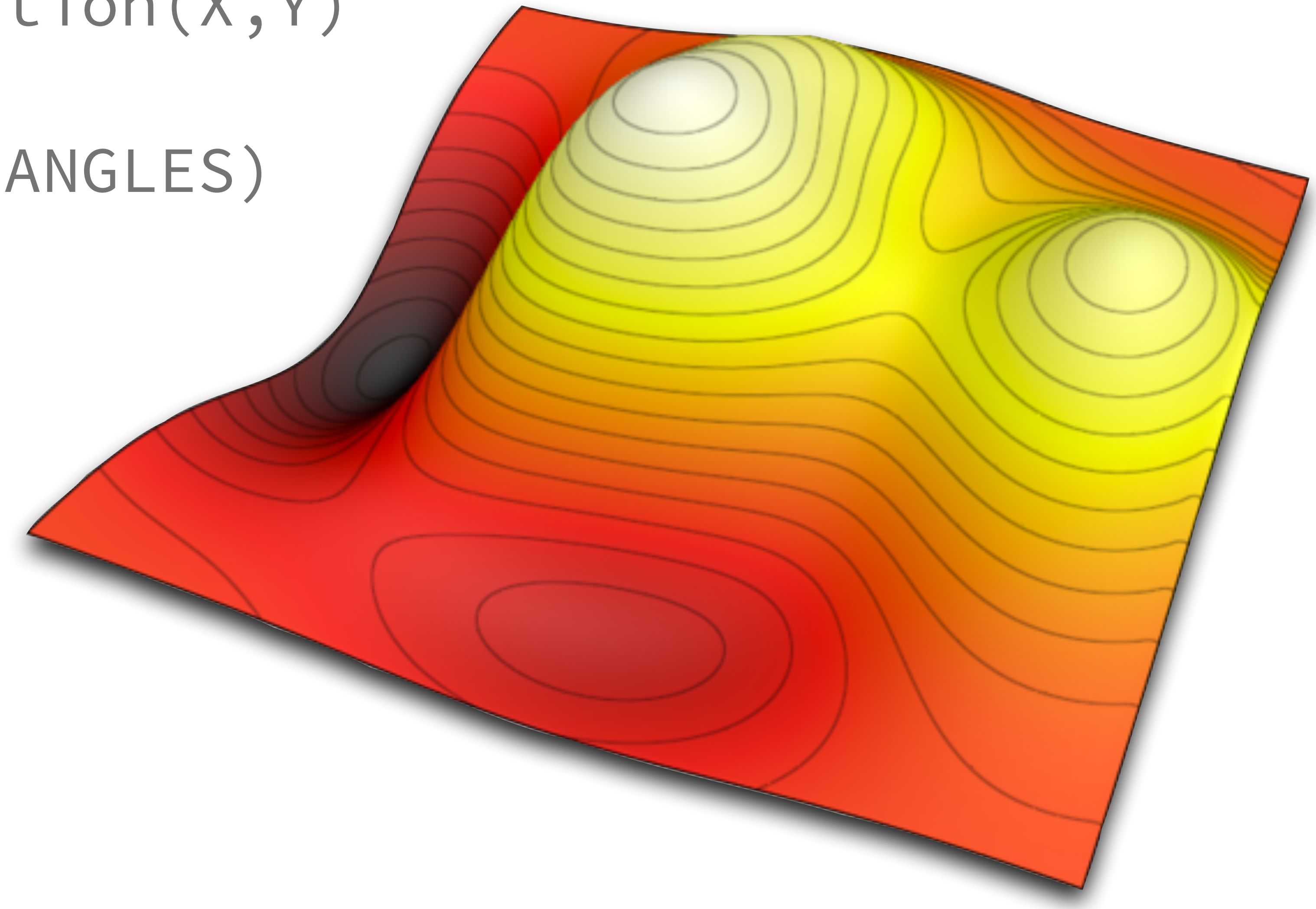
vispy.**gloo**

The missing OpenGL/Python glue

vispy.**gloo**

GLOO is an interface between OpenGL and python that provides a convenient and intuitive interface to shader programs and vertex buffers.

```
…
X,Y = np.meshgrid(x, y)
program['data'] = function(X,Y)
…
program.draw(gl.GL_TRIANGLES)
…
```



vispy.**gloo**

All gloo operations are deferred until the first call to the `draw` function of a program (no GL context needed).

vispy.gloo.**Program**
→ How to draw

vispy.gloo.**VertexBuffer**
→ What to draw (data, **first form**)

vispy.gloo.**IndexBuffer**
→ Vertex data drawing order

vispy.gloo.**Texture**
→ What to draw (data, **second form**)

vispy.gloo.**Collection**
→What to draw (data, **third form**)

vispy.gloo.**FrameBuffer**
→ Where to draw (offscreen)

vispy.gloo.**Program**

A shader program is built from a vertex source code and a fragment source code:

```
program = Program(“vertex.txt”,”fragment.txt”)
program = Program(vertex, fragment)
```

Vertex and fragment sources are parsed such that a program knows its attributes, uniforms and varyings (without compilation).

```
print (program.all_uniforms)
print (program.all_attributes)
```

Once the program has been built in GPU, it is also possible to know what are the active uniforms and attributes (the ones actually used).

vispy.gloo.**VertexBuffer**

A vertex buffer is built from a structured numpy array or from a structured dtype and size:

```
V = VertexBuffer(data=data)
V = VertexBuffer(size=size,dtype=dtype)
```

Vertex buffer dtype is parsed and checked against dtype allowed by ES 2.0.

Once the program is built (in GPU), it is also possible to know what are the active uniforms and attributes (the ones actually used).

gloo offers an interface to upload data “a la numpy” with the data contiguity restriction (because of OpenGL).

vispy.gloo.**IndexBuffer**

An index buffer is built from a numpy array or from a dtype and size:

```
V = IndexBuffer(data=data)
V = IndexBuffer(size=size,dtype=dtype)
```

Index buffer dtype is parsed and checked against dtype allowed by ES 2.0.

vispy.gloo.**Texture**

A Texture is used to represent a topological set of scalar values.

```
T = Texture1D(data=np.zeros(256))
T = Texture2D(data=np.zeros((256,256)))
```

vispy.gloo.**Collection**

A collection is a container for several objects having the same vertex structure (`vtype`) and same uniform type (`utype`). A collection allows to manipulate objects individually but they can be rendered at once using a single vertex buffer.

```
V = Collection(vtype, utype)
V.append(vdata, udata, partition)
…
V[0].vertices, V[0].indices, V[0].uniforms
…
V[“position”], V[“color”], …
```

Binding program and vertex buffers

A program knows a lot about its attributes and uniforms and thus allows us to offer a convenient and intuitive interface:

```
program = Program(vertex, fragment, 3)
program[“position”] = [[0,0,1],[0,1,0],[1,0,0]]
```

If program is given a vertex count, it can implicitly build the corresponding unique vertex buffer gathering all attributes.

It is also possible to assign a single vertex buffer to each attribute or any combination

```
program = Program(vertex, fragment)
program[“position”] = VertexBuffer(data)
```

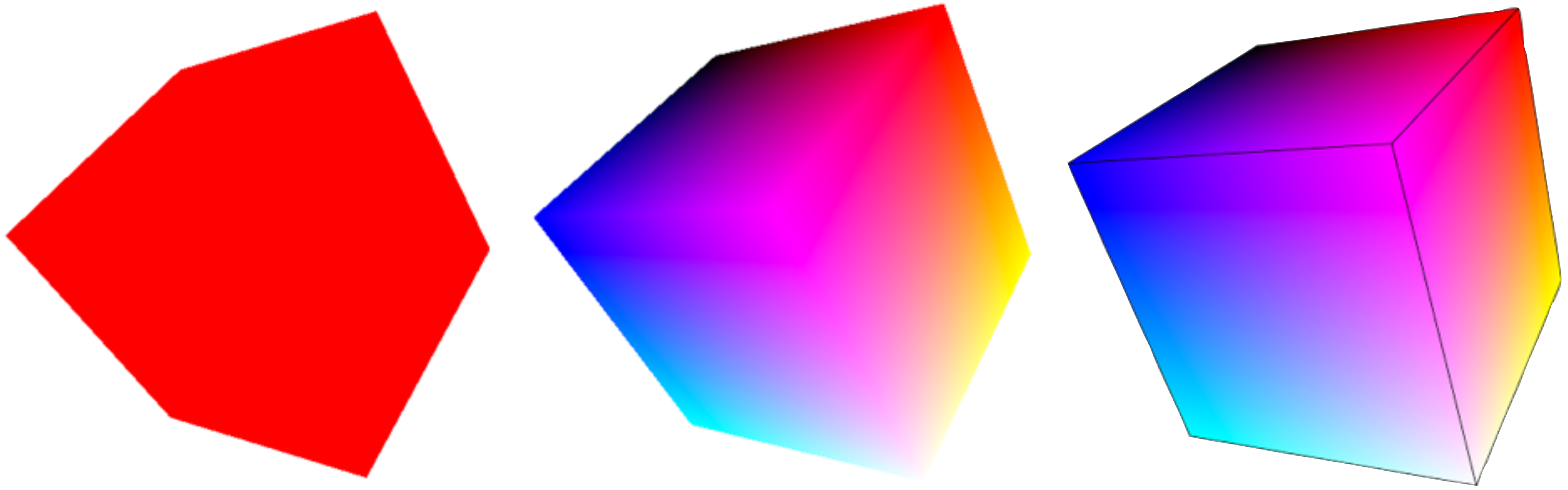
Using index buffer to specify drawing order

When drawing, the order of the vertices can be specified using an index buffer:

```
program = Program(vertex, fragment, 3)
…
program.draw(gl.GL_TRIANGLES, indices)
…
```

If no indices are given, the implicit order of the vertex buffer is used.

Enough talk, let’s try it out…



www.loria.fr/~rougier/teaching/opengl/