Computational Graphics: Lecture 15

Alberto Paoluzzi

Mon, Apr 13, 2015

Outline: LAR primitive objects

1. Parametric curves and surfaces
2. LAR infrastructure
3. 2D primitives
4. 3D primitive surfaces
5. 3D primitive solids

Parametric curves and surfaces

Definitions

- Parametric curve is vector-valued function of one real parameter

in the following we discuss a simple implementation of several well-known curves and surfaces

Definitions

- Parametric curve is vector-valued function of one real parameter
- Parametric surface is a vector-valued function of two real parameter

in the following we discuss a simple implementation of several well-known curves and surfaces

Definitions

- Parametric curve is vector-valued function of one real parameter
- Parametric surface is a vector-valued function of two real parameter
- Parametric solid is a vector-valued function of three real parameter

in the following we discuss a simple implementation of several well-known curves and surfaces

LAR infrastructure

Affine transformation of LAR vertices (1/2)

Primitive maps of points to points via direct transformation of coordinates

Affine transformations of *d*-points

```
def translatePoints (points, tvect):                    # d-dimensional
    return [VECTSUM([p,tvect]) for p in points]

def rotatePoints (points, angle):                       # 2-dimensional
    a = angle
    return [[x*COS(a)-y*SIN(a), x*SIN(a)+y*COS(a)] for x,y in points]

def scalePoints (points, svect):                        # d-dimensional
    return [AA(PROD)(TRANS([p,svect])) for p in points]
```

Affine transformation of LAR vertices (1/2)

Primitive maps of points to points via direct transformation of coordinates

Affine transformations of *d*-points

```
def translatePoints (points, tvect):                    # d-dimensional
    return [VECTSUM([p,tvect]) for p in points]

def rotatePoints (points, angle):                       # 2-dimensional
    a = angle
    return [[x*COS(a)-y*SIN(a), x*SIN(a)+y*COS(a)] for x,y in points]

def scalePoints (points, svect):                        # d-dimensional
    return [AA(PROD)(TRANS([p,svect])) for p in points]
```

Assignment

make the rotation d-dimensional

Affine transformation of LAR vertices (2/2)

Primitive maps of points to points via direct transformation of coordinates

```
V,CV = larSimplexGrid([5,5])
V = rotatePoints (V, PI/4)
model = V,CV
VIEW(EXPLODE(1.2,1.2,1)(MKPOLS(model)))
```

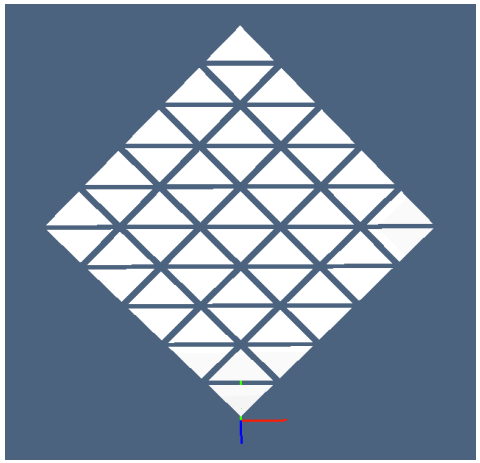


Figure: Positive 2D rotation of a simplicial grid

Standard and scaled partition of unit domain

Simplicial decomposition of the $[0,1]^d$ domain

Simplicial partition of standard d-cube

```
def larDomain(shape):
    V,CV = larSimplexGrid(shape)
    V = scalePoints(V, [1./n for n in shape])
    return V,CV
```

Simplicial partition of d-domain (scaled d-cube)

```
def larIntervals(shape):
    def larIntervals0(size):
        V,CV = larDomain(shape)
        V = scalePoints(V, [scaleFactor for scaleFactor in size])
        return V,CV
    return larIntervals0
```

larMap primitive: mapping domain vertices

Primitive generator of curved objects

```
def larMap(coordFuncs):
    def larMap0(domain):
        V,CV = domain
        V = TRANS(CONS(coordFuncs)(V))   # plasm CONStruction
        return V,CV
    return larMap0
```

Second-order function

Takes as input a list coordFuncs of coordinate **functions** and a domain of type: "model" $\equiv$ (V,CV)

```
larMap([x,y,z])(domain):
```

Identify close or coincident points (1/2)

Create a dictionary with key the point location

```
def checkModel(model):
    V,CV = model; n = len(V)
    vertDict = defaultdict(list)
    for k,v in enumerate(V): vertDict[vcode(v)].append(k)
    verts = (vertDict.values())
    invertedindex = [None]*n
    for k,value in enumerate(verts):
        for i in value:
            invertedindex[i] = value[0]
    CV = [[invertedindex[v] for v in cell] for cell in CV]
    # filter out degenerate cells
    CV = [list(set(cell)) for cell in CV
            if len(set(cell))==len(cell)]
    return V, CV
```

Identify close or coincident points (2/2)

Create a dictionary with key the point location

```
larCircle(1)(4)
>>> ([[1.0, 0.0],
   [6.123233995736766e-17, 1.0],
   [-1.0, 1.2246467991473532e-16],
   [-1.8369701987210297e-16, -1.0],
   [1.0, -2.4492935982947064e-16]],
  [[0, 1], [1, 2], [2, 3], [3, 4]])
```

Identify close or coincident points (2/2)

Create a dictionary with key the point location

```
larCircle(1)(4)
>>> ([[1.0, 0.0],
  [6.123233995736766e-17, 1.0],
  [-1.0, 1.2246467991473532e-16],
  [-1.8369701987210297e-16, -1.0],
  [1.0, -2.4492935982947064e-16]],
 [[0, 1], [1, 2], [2, 3], [3, 4]])
```

design decision: return either the original points of the key values?

```
vertDict = defaultdict(list)
for k,v in enumerate(V): vertDict[vcode(v)].append(k)
print vertDict
>>> defaultdict(<type 'list'>, {'[1., 0.]': [0, 4],
'[0., -1.]': [3], '[0., 1.]': [1], '[-1., 0.]': [2]})
print CV
>>> [[0, 1], [1, 2], [2, 3], [3, 0]]
```

2D primitives

larCircle: circle centered in the origin (1/2)

```
def larCircle(radius=1.):
    def larCircle0(shape=36):
        domain = larIntervals([shape])([2*PI])
        V,CV = domain
        x = lambda V : [radius*COS(p[0]) for p in V]
        y = lambda V : [radius*SIN(p[0]) for p in V]
        return larMap([x,y])(domain)
    return larCircle0
```

```
model = checkModel(larCircle(1)())
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
```

larCircle: circle centered in the origin (2/2)

```
model = checkModel(larCircle(1)())
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
print AA(len)(model)
>>> [37, 36]                # by virtue of my (bad?) design choice
print model[1]
>>> [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7],
    ... [32, 31], [32, 33], [33, 34], [34, 35], [0, 35]]
```

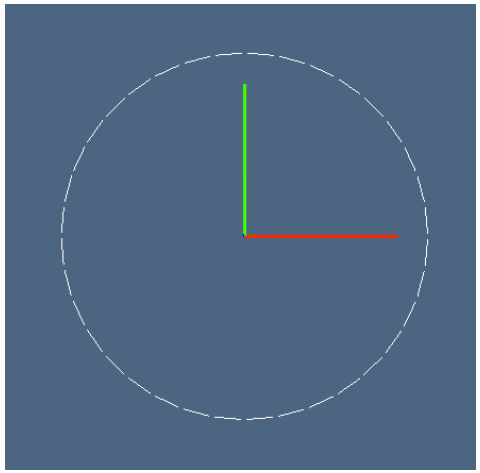


Figure: Circle centered in the origin, with unit radius and default shape

larDisk: disk centered in the origin (1/2)

```
def larDisk(radius=1.):
    def larDisk0(shape=[36,1]):
        domain = larIntervals(shape)([2*PI,radius])
        V,CV = domain
        x = lambda V : [p[1]*COS(p[0]) for p in V]
        y = lambda V : [p[1]*SIN(p[0]) for p in V]
        return larMap([x,y])(domain)
    return larDisk0
```

larDisk: disk centered in the origin (1/2)

```
def larDisk(radius=1.):
    def larDisk0(shape=[36,1]):
        domain = larIntervals(shape)([2*PI,radius])
        V,CV = domain
        x = lambda V : [p[1]*COS(p[0]) for p in V]
        y = lambda V : [p[1]*SIN(p[0]) for p in V]
        return larMap([x,y])(domain)
    return larDisk0
```

```
V,CV1 = larDisk(1)([4,1])
len(CV1)
>>> 8
V,CV1 = checkModel(larDisk(1)([4,1]))
len(CV1)
>>> 4
```

larDisk: disk centered in the origin (2/2)

```
model = checkModel(larDisk(1)([36,4]))
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
```

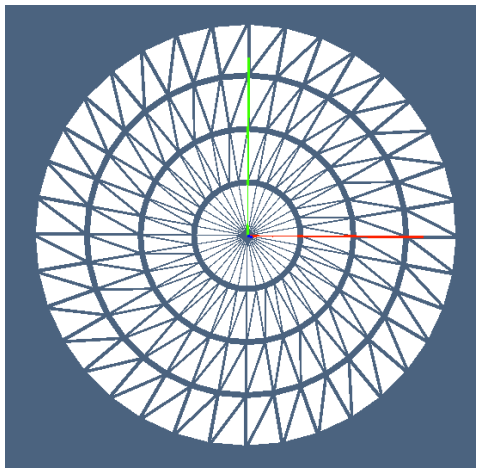


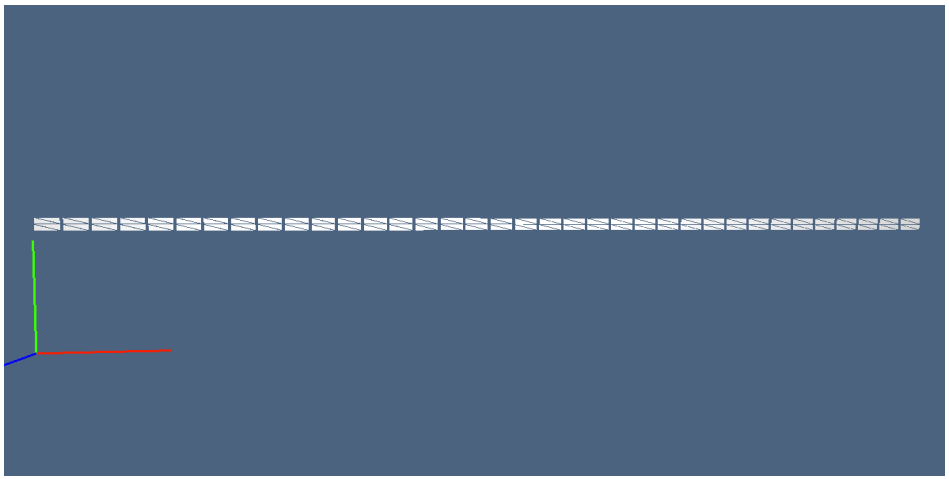
Figure: Disk centered in the origin, with unit radius and given shape

larRing: ring centered in the origin (2/2)

```
def larRing(params):
    r1,r2 = params
    def larRing0(shape=[36,1]):
        V,CV = larIntervals(shape)([2*PI,r2-r1])
        V = translatePoints(V,[0,r1])
        domain = V,CV
        x = lambda V : [p[1] * COS(p[0]) for p in V]
        y = lambda V : [p[1] * SIN(p[0]) for p in V]
        return larMap([x,y])(domain)
    return larRing0
```

larRing: ring centered in the origin (2/2)

```
model = checkModel(larRing([.9, 1.])([36,2]))
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
```



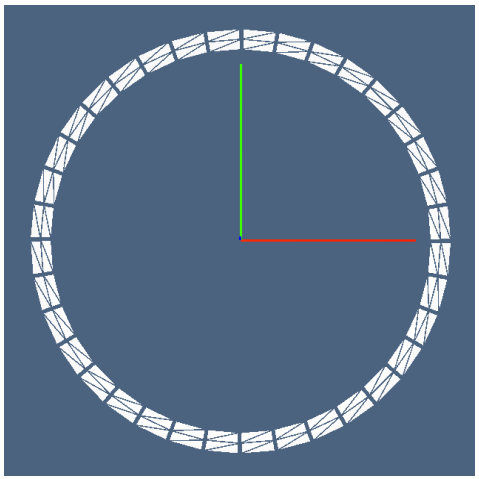


Figure: 2D ring centered in the origin, with given radius and shape

3D primitive surfaces

larCylinder: cylinder about the z-axis (1/2)

```
def larCylinder(params):
    radius,height= params
    def larCylinder0(shape=[36,1]):
        domain = larIntervals(shape)([2*PI,1])
        V,CV = domain
        x = lambda V : [radius*COS(p[0]) for p in V]
        y = lambda V : [radius*SIN(p[0]) for p in V]
        z = lambda V : [height*p[1] for p in V]
        mapping = [x,y,z]
        model = larMap(mapping)(domain)
        # model = makeOriented(model)
        return model
    return larCylinder0
```

larCylinder: cylinder about the z-axis (2/2)

```
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = checkModel(larCylinder([.5,2.])([32,1]))
```

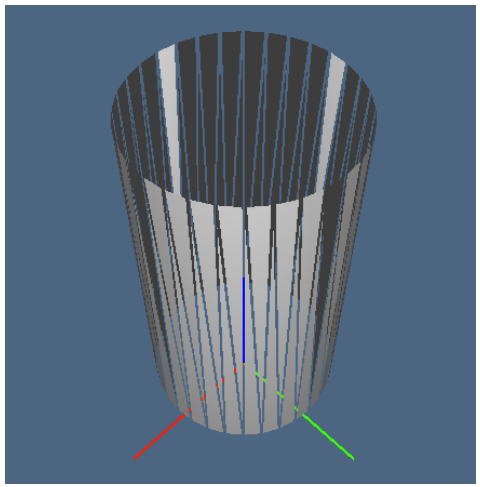


Figure: Cylinder about the z-axis, with basis centered in the origin, of given radius, height, shape.

larSphere: spherical surface centered in the origin (1/2)

```
def larSphere(radius=1):
    def larSphere0(shape=[18,36]):
        V,CV = larIntervals(shape)([PI,2*PI])
        V = translatePoints(V,[-PI/2,-PI])
        domain = V,CV
        x = lambda V : [radius*COS(p[0])*COS(p[1]) for p in V]
        y = lambda V : [radius*COS(p[0])*SIN(p[1]) for p in V]
        z = lambda V : [radius*SIN(p[0]) for p in V]
        return larMap([x,y,z])(domain)
    return larSphere0

V,CV = checkModel(larSphere()())
AA(len)((V,CV))
>>> [703, 1224]
(18 * 36) * 2 - (36 * 2)
>>> 1224
```

larSphere: spherical surface centered in the origin (2/2)

```
model = checkModel(larSphere(1)())
VIEW(STRUCT(MKPOLS(model)))
```

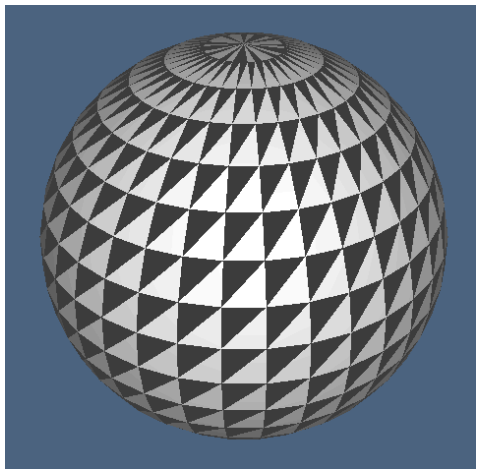


Figure: Sphere centered in the origin, with given radius and shape

larToroidal: toroidal surface centered in the origin (1/2)

```
def larToroidal(params):
    r,R = params
    def larToroidal0(shape=[24,36]):
        domain = larIntervals(shape)([2*PI,2*PI])
        V,CV = domain
        x = lambda V : [(R + r*COS(p[0])) * COS(p[1]) for p in V]
        y = lambda V : [(R + r*COS(p[0])) * SIN(p[1]) for p in V]
        z = lambda V : [-r * SIN(p[0]) for p in V]
        return larMap([x,y,z])(domain)
    return larToroidal0
```

larToroidal: toroidal surface centered in the origin (2/2)

```
model = checkModel(larToroidal([0.5,1])())
VIEW(STRUCT(MKPOLS(model)))
```

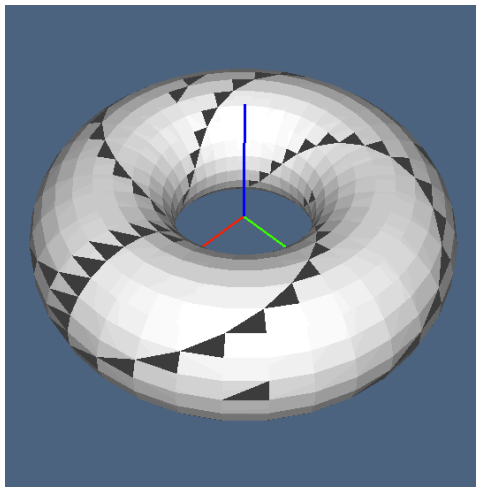


Figure: Toroidal surface centered in the origin, with given radiuses and shape

larCrown: Circolar crown surface centered in the origin (1/2)

Half-toroidal surface of given radiuses

```
def larCrown(params):
    r,R = params
    def larCrown0(shape=[24,36]):
        V,CV = larIntervals(shape)([PI,2*PI])
        V = translatePoints(V,[-PI/2,0])
        domain = V,CV
        x = lambda V : [(R + r*COS(p[0])) * COS(p[1]) for p in V]
        y = lambda V : [(R + r*COS(p[0])) * SIN(p[1]) for p in V]
        z = lambda V : [-r * SIN(p[0]) for p in V]
        return larMap([x,y,z])(domain)
    return larCrown0
```

larCrown: Circolar crown surface centered in the origin (2/2)

```
model = checkModel(larCrown([0.125,1])([8,48]))
VIEW(STRUCT(MKPOLS(model)))
```

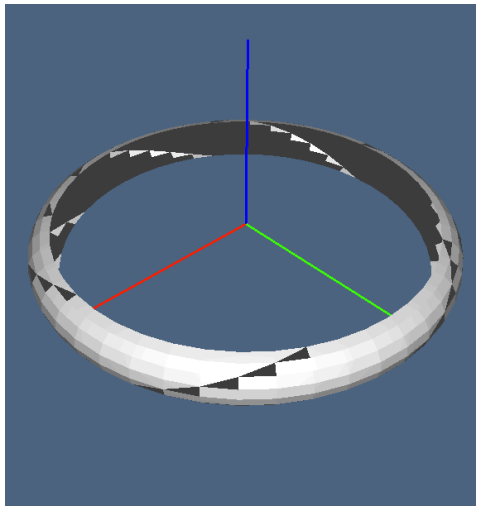


Figure: Circolar crown surface centered in the origin, with unit radius and given shape

3D primitive solids

larBall: solid 3D disk centered in the origin (1/2)

```
def larBall(radius=1):
    def larBall0(shape=[18,36]):
        V,CV = checkModel(larSphere(radius)(shape))
        return V,[range(len(V))]
    return larBall0
```

larBall: solid 3D disk centered in the origin (1/2)

```
def larBall(radius=1):
    def larBall0(shape=[18,36]):
        V,CV = checkModel(larSphere(radius)(shape))
        return V,[range(len(V))]
    return larBall0
```

```
model = checkModel(larSphere(1)())
AA(len)(model)
>>> [703, 0]              # BUG:  to solve

pol = JOIN(STRUCT(MKPOLS(model)))
pol
>>> <pyplasm.xgepy.Hpc; proxy of <Swig Object of type
    'std::tr1::shared_ptr< Hpc > *' at 0x1106d8330> >
```

larBall: solid 3D disk centered in the origin (2/2)

```
model = checkModel(larBall(1)([18,36]))
VIEW(STRUCT(MKPOLS(model)))
```

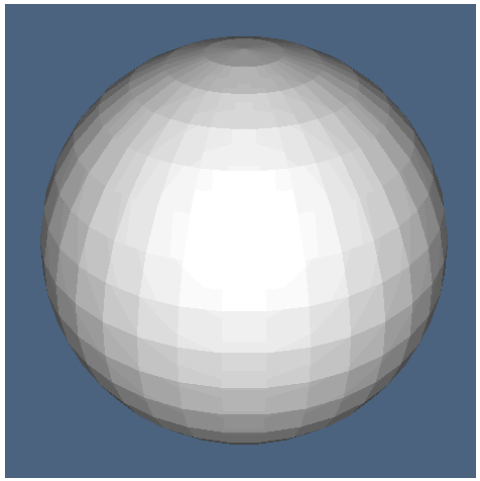


Figure: Solid disk centered in the origin, with given radius and shape

larRod: solid Cylinder centered in the origin (2/2)

```
def larRod(params):
    radius,height= params
    def larRod0(shape=[36,1]):
        V,CV = checkModel(larCylinder(params)(shape))
        return V,[range(len(V))]
    return larRod0
```

larRod: solid Cylinder centered in the origin (2/2)

```
def larRod(params):
    radius,height= params
    def larRod0(shape=[36,1]):
        V,CV = checkModel(larCylinder(params)(shape))
        return V,[range(len(V))]
    return larRod0
```

```
V,CV = larRod([.25,2.])([32,1])
len(CV)
>>> 1
print CV,        # single LAR cell!
[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,
47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65]]
```

larRod: solid Cylinder centered in the origin (2/2)

```
model = larRod([.25,2.])([32,1])
VIEW(STRUCT(MKPOLS(model)))
```

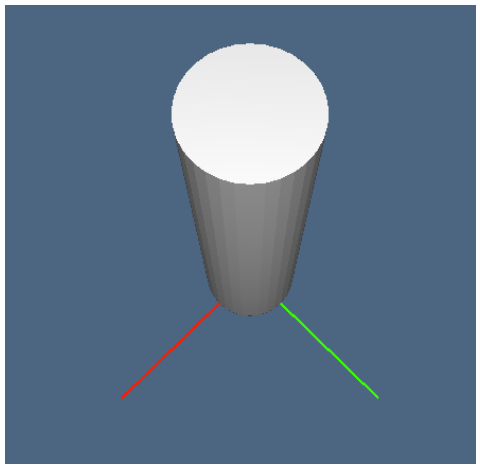


Figure: Solid cylinder about the z-axis, with given radius, height and shape

larPizza: solid 3D disk centered in the origin (1/2)

Solid pizza of given radiuses

```
def larPizza(params):
    r,R= params
    def larPizza0(shape=[24,36]):
        V,CV = checkModel(larCrown(params)(shape))
        return V,[range(len(V))]
    return larPizza0
```

larPizza: solid 3D disk centered in the origin (2/2)

```
model = larPizza([0.05,1])([8,48])
VIEW(STRUCT(MKPOLS(model)))
```

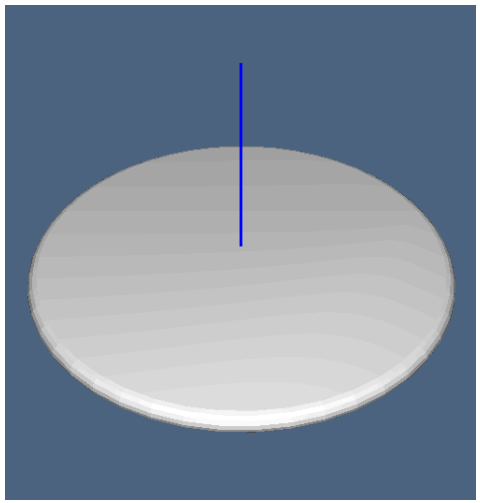


Figure: Solid 3D disk centered in the origin, with given radius and shape

larTorus: solid 3D torus centered in the origin (1/2)

```
def larTorus(params):
    r,R = params
    def larTorus0(shape=[24,36,1]):
        domain = larIntervals(shape)([2*PI,2*PI,r])
        V,CV = domain
        x = lambda V : [(R + p[2]*COS(p[0])) * COS(p[1]) for p in V]
        y = lambda V : [(R + p[2]*COS(p[0])) * SIN(p[1]) for p in V]
        z = lambda V : [-p[2] * SIN(p[0]) for p in V]
        return larMap([x,y,z])(domain)
    return larTorus0
```

larTorus: solid 3D torus centered in the origin (1/2)

```
def larTorus(params):
    r,R = params
    def larTorus0(shape=[24,36,1]):
        domain = larIntervals(shape)([2*PI,2*PI,r])
        V,CV = domain
        x = lambda V : [(R + p[2]*COS(p[0])) * COS(p[1]) for p in V]
        y = lambda V : [(R + p[2]*COS(p[0])) * SIN(p[1]) for p in V]
        z = lambda V : [-p[2] * SIN(p[0]) for p in V]
        return larMap([x,y,z])(domain)
    return larTorus0

model = checkModel(larTorus([0.5,1])())
AA(len)(model)
>>> [1850, 2592]
model[1]
>>> [[0,25,925,926],[25,925,926,950],[25,950,926,951],[0,25,926,927],
    ... ... ...
    [25,951,926,927],[952,25,951,927],[0,947,1822,1823],[0,947,948,1823],
    [0,1800,875,1823],[0,1800,948,1823],[0,1800,948,925]]
```

larTorus: solid 3D torus centered in the origin (2/2)

```
model = checkModel(larTorus([0.5,1])())
VIEW(STRUCT(MKPOLS(model)))
```

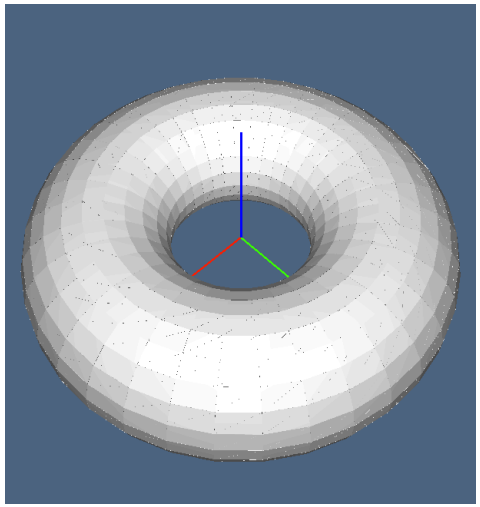


Figure: Disk centered in the origin, with unit radius and given shape

References

GP4CAD book