Computational Graphics: Lecture 8

The CVDlab Team

Tue, Mar 18, 2014

Outline: LAR1

1. Simplicial mapping
2. Examples of MAP
3. References

Simplicial mapping

Simplicial mapping: definition

Definition

A simplicial map is a map between simplicial complexes with the property that the images of the vertices of a simplex always span a simplex.

Simplicial mapping: definition

Definition

A simplicial map is a map between simplicial complexes with the property that the images of the vertices of a simplex always span a simplex.

Remarks

Simplicial maps are determined by their effects on vertices
for a precise definition of Simplicial Map look at Wolfram MathWorld

MAP operator in plasm

Map operator

```
MAP(fun)(domain)
```

MAP operator in plasm

Map operator

`MAP(fun)(domain)`

Semantics

1. domain (HPC value) is decomposed into a simplicial complex

MAP operator in plasm

Map operator

`MAP(fun)(domain)`

Semantics

1. domain (HPC value) is decomposed into a simplicial complex
2. fun (a simplicial function) is applied to the domain vertices

MAP operator in plasm

Map operator

`MAP(fun)(domain)`

Semantics

1. domain (HPC value) is decomposed into a simplicial complex
2. fun (a simplicial function) is applied to the domain vertices
3. the mapped domain is returned

MAP examples: 1-sphere (S^1) and 2-disk (D^2)

```
def sphere1(p): return [COS(p[0]), SIN(p[0])] # point function
def domain(n): return INTERVALS(2*PI)(n)      # generator of domain decomp
VIEW( MAP(sphere1)(domain(32)) )              # geometric value (HPC type)

def disk2D(p):                                                   # point function
    u,v = p
    return [v*COS(u), v*SIN(u)]                                  # coordinate functions
domain2D = PROD([INTERVALS(2*PI)(32), INTERVALS(1)(3)])  # 2D domain decompos
VIEW( MAP(disk2D)(domain2D) )
VIEW( SKELETON(1)(MAP(disk2D)(domain2D)) )
```

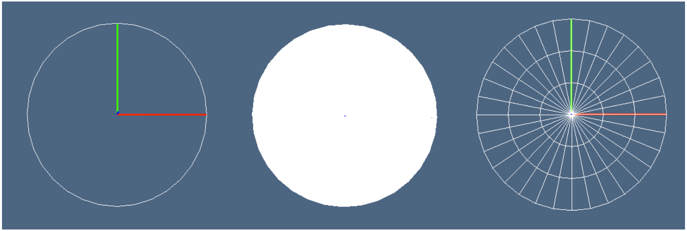
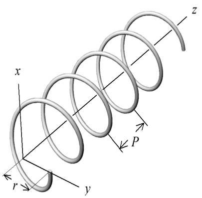


Figure : (a) sphere S^1 (b) disk D^2; (c) 1-skeleton.

MAP examples: 1-helix

```
def helix(r):                                           # point function
    def helix0(pitch):
        def helix1(p):
            return [r*COS(p[0]), r*SIN(p[0]), (pitch/(2*PI))*p[0]]
        return helix1
    return helix0

def domain(len,n): return INTERVALS(len)(n)     # generator of domain decomp
VIEW( MAP(helix(0.2)(0.05))(domain(10*(2*PI),320)) )    # geometric value (HPC type
```



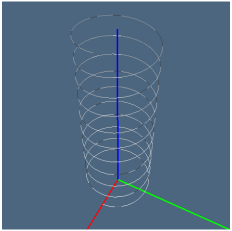


Figure : (a) sphere S^1 (b) disk D^2; (c) 1-skeleton.

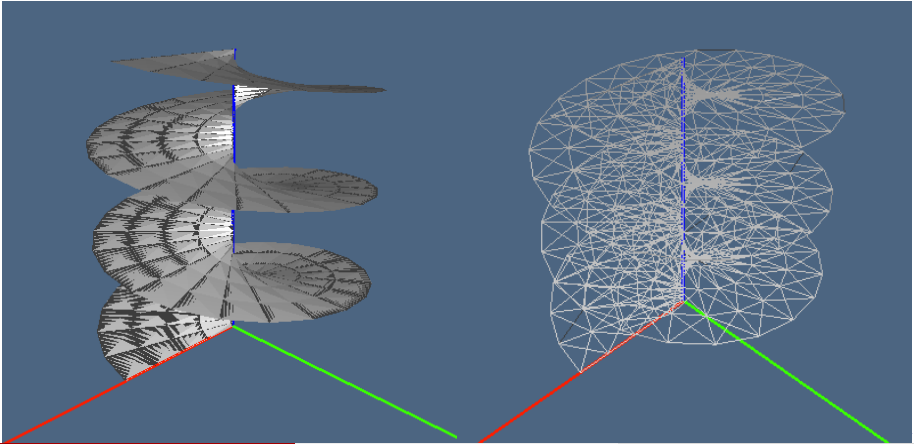
MAP examples: 2-helicoid

```
def helicoid(radius):                          # point function
    def helix0(pitch):
        def helix1(p):
            a,r = p
            return [radius*r*COS(a), -radius*r*SIN(a), (pitch/(2*PI))*a]
        return helix1
    return helix0
fun = helicoid(0.5)(0.333)
dom = PROD([INTERVALS(2*PI*3)(64), INTERVALS(1)(4)])
pol = MAP(fun)(dom)

VIEW( pol )                                    # geometric value (HPC type)
VIEW( SKELETON(1)(pol) )                       # geometric value (HPC type)
```

MAP examples: 2-helicoid incorrect!! WHY?

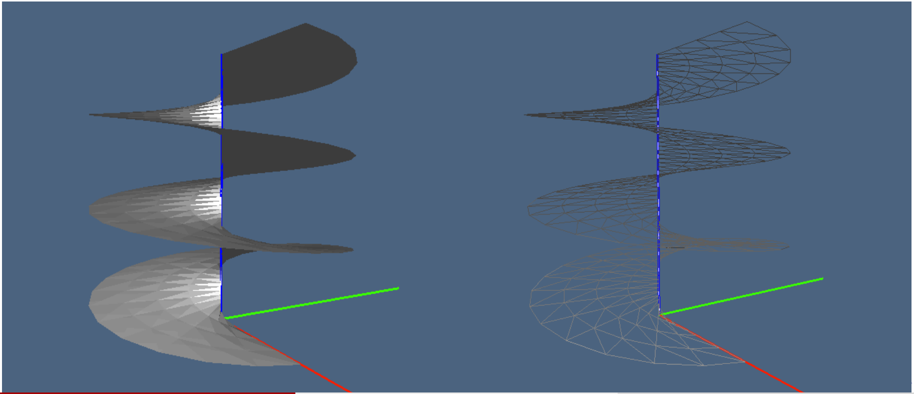
```
dom1Da, dom1Db = INTERVALS(2*PI*3)(64), INTERVALS(1)(4)
dom = PROD([ dom1Da, dom1Db ])          # Cartesian product of intervals
VIEW( MAP(fun)(dom) )                   # geometric value (HPC type)
VIEW( SKELETON(1)(MAP(fun)(dom)) )      # geometric value (HPC type)
```



MAP examples: 2-helicoid correct!!

```
def domain(shape):                    # simplicial decomposition of domain
    def domain0(size):
        return S([1,2])(size)(GRID(shape))
    return domain0

dom = domain([64,4])([2*PI*3,1])      # provided via GRID primitive
```



Examples of MAP

Mapping a function over the vertices of a domain

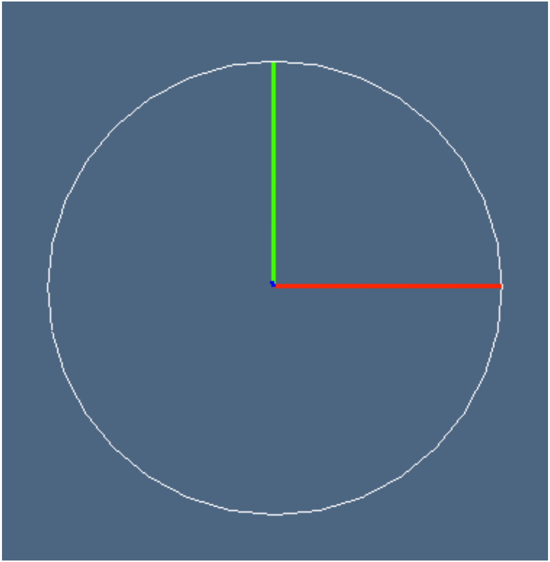
function of point returning a list of coordinate functions

```
def circle(p):
    alpha = p[0]
    return [COS(alpha), SIN(alpha)]
```

primitive constructor `INTERVALS(x)(n)` of a simplicial decomposition of the $[0, x]$ interval into n subintervals

```
obj = MAP(circle)(INTERVALS(2*PI)(32))
VIEW(obj)
```

Mapping a function over the vertices of a domain



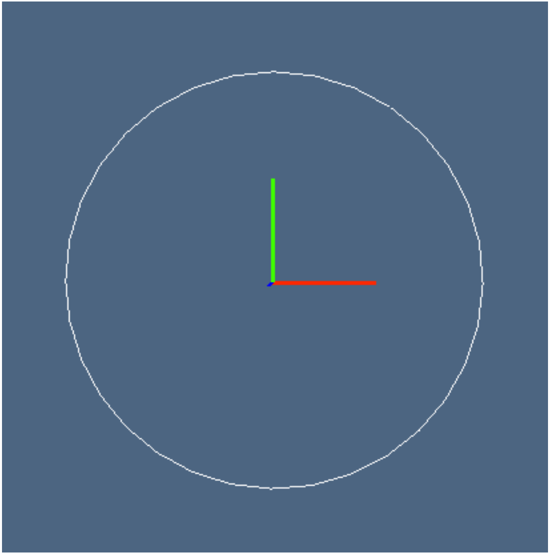
Mapping an higher-level function

`circle(r)(p)` is now parameterized by the *r* value

```
def circle(r):
    def circle0(p):
        alpha = p[0]
        return [r*COS(alpha), r*SIN(alpha)]
    return circle0

obj = MAP(circle(2))(INTERVALS(2*PI)(32))
VIEW(obj)
```

Mapping an higher-level function



Mapping an higher-level function

`dom(n)` is now parameterized by the *n* values

```
def dom(n):
    return INTERVALS(2*PI*n)(24*n)
```

`spiral(pitch,n)(p)` is now parameterized by the *pitch*, *n* values

```
def spiral(pitch,n):
    def spiral0(p):
        alpha = p[0]
        return [COS(alpha), SIN(alpha), alpha*pitch*n/(2*PI*n)
    return spiral0

obj = MAP(spiral(0.2,5))(dom(5))
VIEW(obj)
```

Mapping an higher-level function

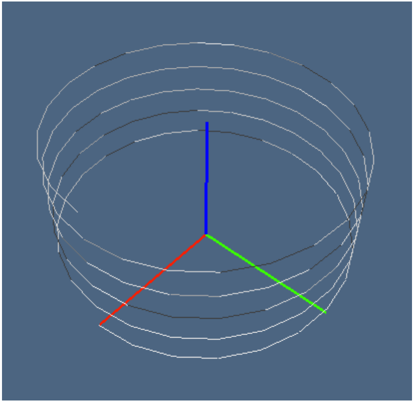


Figure : Spiral curve in 3D (3 coordinate functions)

Mapping a 2D domain

The domain dom2D $= [0, 2\pi] \times [0, 1]$ is the Cartesian product of two 1D intervals

```
dom2D = PROD([INTERVALS(2*PI)(24), INTERVALS(1)(1)])
VIEW(dom2D)
```

It is useful to look at its 1-skeleton

```
VIEW(SKELETON(1)(dom2D))
```

Mapping a 2D domain

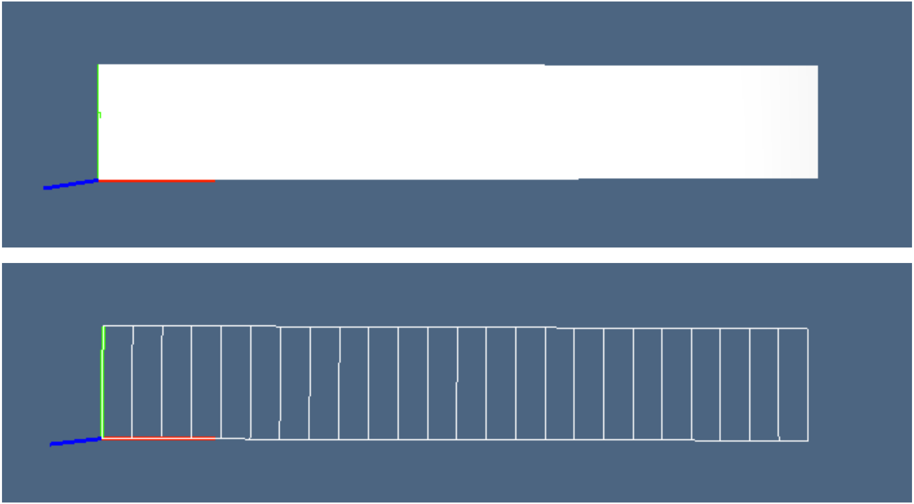


Figure : $\texttt{dom2D} = [0, 2\pi] \times [0, 1]$ and its 1-skeleton

2D/3D spiral surface/solid

p$\in \mathbb{E}^2$ contains two coordinates

```
def spiral(p):
    alpha,r = p
    return [r*COS(alpha), r*SIN(alpha), alpha/(2*PI)]

obj = MAP(spiral)(dom2D)
VIEW(obj)
```

2D/3D spiral surface/solid

p$\in \mathbb{E}^2$ contains two coordinates

```
def spiral(p):
    alpha,r = p
    return [r*COS(alpha), r*SIN(alpha), alpha/(2*PI)]

obj = MAP(spiral)(dom2D)
VIEW(obj)
```

p$\in \mathbb{E}^3$ contains two coordinates

```
dom1D = INTERVALS(1)(1)
dom3D = INSR(PROD)([INTERVALS(2*PI)(24), dom1D, dom1D])
def spiral(p):
    alpha,r,h = p
    return [r*COS(alpha), r*SIN(alpha), h*alpha/(2*PI)]

obj = MAP(spiral)(dom3D)
VIEW(obj)
```

2D/3D spiral surface/solid

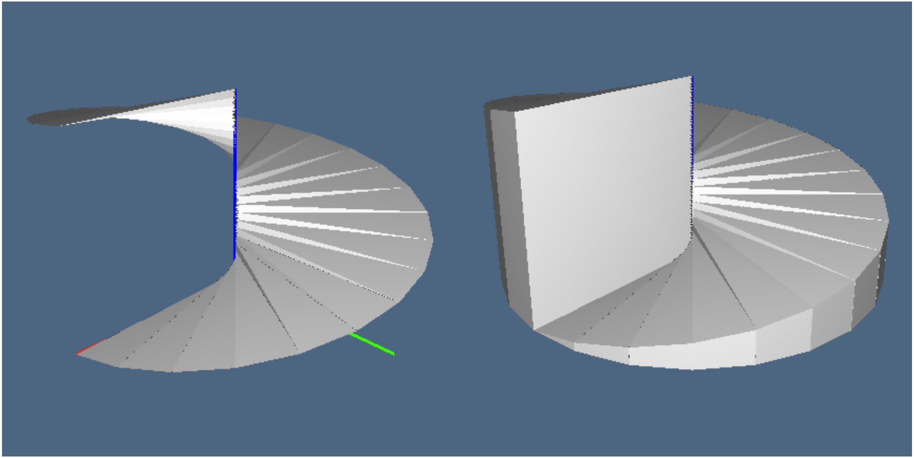


Figure : spiral surface and solid spiral

3D solid spiraloid

Two surface functions $\mathbb{E}^3 \to \mathbb{E}^2$ are given

```
dom3D = INSR(PROD)([INTERVALS(2*PI)(24), dom1D, dom1D])
def spiral1(p):
    alpha,r,h = p
    return [r*COS(alpha), r*SIN(alpha), alpha/(2*PI)]
def spiral2(p):
    alpha,r,h = p
    return [r*COS(alpha), r*SIN(alpha), alpha/(2*PI) + 0.1]
```

3D solid spiraloid

Two surface functions $\mathbb{E}^3 \to \mathbb{E}^2$ are given

```
dom3D = INSR(PROD)([INTERVALS(2*PI)(24), dom1D, dom1D])
def spiral1(p):
    alpha,r,h = p
    return [r*COS(alpha), r*SIN(alpha), alpha/(2*PI)]
def spiral2(p):
    alpha,r,h = p
    return [r*COS(alpha), r*SIN(alpha), alpha/(2*PI) + 0.1]
```

The mapping function is a transfinite interpolation of two surface functions

```
obj = STRUCT([MAP(spiral1)(dom3D), MAP(spiral2)(dom3D)])
VIEW(obj)

obj = MAP(BEZIER(S3)([spiral1,spiral2]))(dom3D)
VIEW(obj)
```

3D solid spiraloid

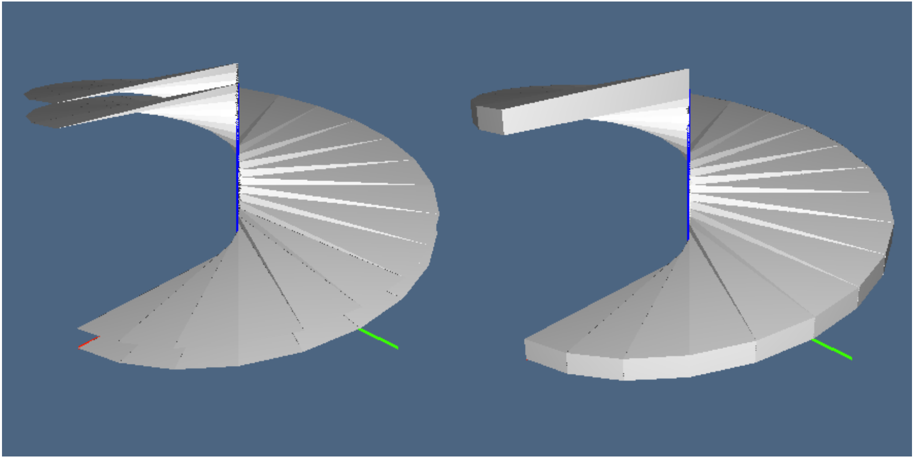


Figure : (a) The two generating surfaces and (b) the solid spiraloid obtained by interpolation

References

References

GP4CAD book