Computational Graphics: Lecture 12

Alberto Paoluzzi

Tue, Apr 28, 2015

Pyplasm model implementation of curves

```
def circle(r=1):
    x = lambda p: r*COS(p[0])
    y = lambda p: r*SIN(p[0])
    return CONS([x,y])
```

Pyplasm model implementation of curves

```
def sincos(r=1):
    x = lambda p: r*COS(p[0])
    y = lambda p: r*SIN(p[0])
    z = lambda p: PROD([x(p),y(p)])
    return [x,y,z]
```

Pyplasm model implementation of curves

```
def circle(r=1):
    def circle0(p):
        u = p[0]
        x = r*COS(u)
        y = r*SIN(u)
        z = x*y
        return [x,y,z]
    return circle0
```

Pyplasm model implementation of curves

```
def dom(interval=1,dmin=0,dmax=1,steps=32):
    interval = interval*(dmax - dmin)
    return T(1)(dmin)(INTERVALS(interval)(steps))

VIEW(MAP(circle(1))(dom(1)))
VIEW(MAP(circle(1))(dom(2*PI)))
VIEW(MAP(circle(1))(dom(-PI,PI)))
VIEW(MAP(circle(1))(dom(-PI/2,3*PI/2)))
```

Notable surface classes

Profile product surfaces

```
def PROFILEPRODSURFACE (args):
    profile_fn,section_fn = args

    def map_fun(point):
        u,v,w=point
        profile,section=profile_fn(point),section_fn(point)
        ret=[profile[0]*section[0],profile[0]*section[1],profi
        return ret
    return map_fun
```

Pyplasm model implementation of curves

```
if __name__ == "__main__":
    alpha=BEZIER(S1)([[0.1,0,0],[2,0,0],[0,0,4],[1,0,5]])
    beta =BEZIER(S2)([[0,0,0],[3,-0.5,0],[3,3.5,0],[0,3,0]])
    plasm_config.push(1e-4)
    domain = EMBED(1)(PROD([ Hpc(Grid([20*[1./20]])),  Hpc(Gri
    out = Plasm.Struct([GMAP(alpha)(domain),MAP(beta )(domain)
    plasm_config.pop()
    VIEW(out)
```

Ruled surfaces

```
def RULEDSURFACE (args):
    alpha_fn , beta_fn = args

    def map_fn(point):
        u,v,w = point
        alpha,beta=alpha_fn(point),beta_fn(point)
        ret=[0.0 for i in range(len(alpha))]
        for K in range(len(ret)): ret[K]=alpha[K]+v*beta[K]
        return ret
    return map_fn
```

Pyplasm model implementation of curves

```
if __name__ == "__main__":
    alpha= lambda point: [point[0],point[0],        0 ]
    beta = lambda point: [    -1,      +1, point[0] ]
    #domain= T([1,2])([-1,-1])(Plasm.power(INTERVALS(2)(10),IN
    domain = EMBED(1)(PROD([ Hpc(Grid([10*[2./10]])),  Hpc(Gri
    domain = MAT([[1,0,0,0],[-1,1,0,0],[-1,0,1,0],[0,0,0,1]])(
    plasm_config.push(1e-4)
    VIEW(GMAP(RULEDSURFACE([alpha,beta]))(domain))
    plasm_config.pop()
```

ROTATIONALSURFACE surfaces

```
def ROTATIONALSURFACE (args):
    profile = args

    def map_fn(point):
        u,v,w = point
        f,h,g = profile(point)
        ret=[f*math.cos(v),f*math.sin(v),g]
        return ret
    return map_fn
```

Pyplasm model implementation of curves

```
if __name__ == "__main__":
    profile=BEZIER(S1)([[0,0,0],[2,0,1],[3,0,4]]) # defined in
    plasm_config.push(1e-4)
    #domain=Plasm.power(INTERVALS(1)(10),INTERVALS(2*PI)(30))
    domain = EMBED(1)(PROD([ Hpc(Grid([10*[1./10]])),  Hpc(Gri
    out = GMAP(ROTATIONALSURFACE(profile))(domain)
    plasm_config.pop()
    VIEW(out)
```

CYLINDRICALSURFACE surfaces

```
def CYLINDRICALSURFACE (args):
    alpha_fun    = args[0]
    beta_fun     = CONS(AA(K)(args[1]))
    return RULEDSURFACE([alpha_fun, beta_fun])
```

Pyplasm model implementation of curves

```
if __name__ == "__main__":
    alpha=BEZIER(S1)([[1,1,0],[-1,1,0],[1,-1,0],[-1,-1,0]])
    #Udomain=INTERVALS(1)(20)
    #Vdomain=INTERVALS(1)(6)
    #domain=Plasm.power(Udomain,Vdomain)
    domain = EMBED(1)(PROD([ Hpc(Grid([20*[1./20]])),  Hpc(Gri
    fn = CYLINDRICALSURFACE([alpha,[0,0,1]])
    VIEW(GMAP(fn)(domain))
```

CONICALSURFACE surfaces

```
def CONICALSURFACE (args):
    apex=args[0]
    alpha_fn   = lambda point: apex
    beta_fn = lambda point: [ args[1](point)[i]-apex[i] for i
    return RULEDSURFACE([alpha_fn, beta_fn])
```

Pyplasm model implementation of curves

```
if __name__ == "__main__":
    #domain=Plasm.power(INTERVALS(1)(20),INTERVALS(1)(6))
    domain = EMBED(1)(PROD([ Hpc(Grid([20*[1./20]])),  Hpc(Gri
    beta = BEZIER(S1)([ [1,1,0],[-1,1,0],[1,-1,0],[-1,-1,0] ])
    out = GMAP(CONICALSURFACE([[0,0,1],beta]))(domain)
    VIEW(out)
```

Profile product surfaces

```
def CUBICHERMITE (U):
    def CUBICHERMITE0 (args):
        p1_fn , p2_fn , s1_fn , s2_fn = args
        def map_fn(point):
            u=U(point);u2=u*u;u3=u2*u
            p1,p2,s1,s2=[f(point) if callable(f) else f for f
            ret=[0.0 for i in range(len(p1))]
            for i in range(len(ret)):
                ret[i]+=(2*u3-3*u2+1)*p1[i] + (-2*u3+3*u2)*p2
            return ret
        return map_fn
    return CUBICHERMITE0
```

Pyplasm model implementation of curves

```
if __name__ == "__main__":

    domain=INTERVALS(1)(20)
    out=Plasm.Struct([
        MAP(CUBICHERMITE(S1)([[1,0],[1,1],[ -1,  1],[ 1,0]]))(c
        MAP(CUBICHERMITE(S1)([[1,0],[1,1],[ -2,  2],[ 2,0]]))(c
        MAP(CUBICHERMITE(S1)([[1,0],[1,1],[ -4,  4],[ 4,0]]))(c
        MAP(CUBICHERMITE(S1)([[1,0],[1,1],[-10,10],[10,0]]))(c
    ])
    VIEW(out)

    c1=CUBICHERMITE(S1)([[1  ,0,0],[0  ,1,0],[0,3,0],[-3,0,0]]
    c2=CUBICHERMITE(S1)([[0.5,0,0],[0,0.5,0],[0,1,0],[-1,0,0]]
    sur3=CUBICHERMITE(S2)([c1,c2,[1,1,1],[-1,-1,-1]])
    #plasm_config.push(1e-4)
    #domain=Plasm.power(INTERVALS(1)(14),INTERVALS(1)(14))
```

Pyplasm model implementation of curves

```
def beta(p):
    """ cross-section curve """
    u = p[0]
    x = COS(u)
    y = SIN(u)
    z = 0
    return [x,y,0]

VIEW(MAP(beta)(dom(2*PI)))
```

Pyplasm model implementation of curves

```
def alpha(p):
    """ profile curve """
    c = bezier([[0,0,0],[4,0,0],[6,0,2],[0,0,4],[2,0,5]])
    x = c(p)[0]
    y = c(p)[1]
    z = c(p)[2]
    return [x,y,z]


VIEW(MAP(alpha)(dom(1)))
```

Pyplasm model implementation of curves

```
def profileProduct(alpha,beta):
    def profileProduct0(p):
        u,v = p
        x = beta([u])[0] * alpha([v])[0]
        y = beta([u])[1] * alpha([v])[0]
        z = alpha([v])[2]
        return [x,y,z]
    return profileProduct0

domain = PROD([dom(interval=2*PI,steps=32),dom(steps=12)])

VIEW(GMAP(profileProduct(alpha,beta))(domain))
```