

# Lar Integrals

## Introduzione

Questo modulo implementa un metodo di integrazione finita di polinomi del tipo:

$$x^\alpha y^\beta z^\gamma \quad (1)$$

Vengono implementate due funzioni **II** e **III** che permettono di fare rispettivamente l'integrale di superficie e di volume di polinomi del tipo specificato.

L'integrale di superficie viene calcolato facendo la somma degli integrali dei triangoli. I triangoli devono essere ottenuti triangolando opportunamente la superficie.

La funzione **TT** permette di calcolare l'integrale sul singolo triangolo.

L'integrale di volume si ottiene facilmente grazie al Teorema della Divergenza; questo teorema permette di calcolare un integrale di volume da un integrale di superficie.

È importante notare che tutti i domini, sia 2D che 3D, sono definiti in 3 dimensioni.

## Descrizione funzioni di integrazione

La funzione `function M(alpha::Int, beta::Int)::Float64` calcola la seguente formula:

$$II^{\alpha\beta} = \frac{1}{\alpha+1} \sum_{h=0}^{\alpha+1} \binom{\alpha+1}{h} \frac{(-1)^h}{h+\beta+1} \quad (2)$$

Che con  $\alpha = 0$  e  $\beta = 0$  si riduce al calcolo dell'area del triangolo con vertici  $w_o = (0, 0)$ ,  $w_a = (1, 0)$  e  $w_b = (0, 1)$ , pari a  $\frac{1}{2}$ .

La seguente funzione:

```
function TT(tau::Array{Float64,2},
           alpha::Int, beta::Int, gamma::Int, signedInt::Bool=false)
    vo,va,vb = tau[:,1],tau[:,2],tau[:,3]
    a = va - vo
    b = vb - vo
    s1 = 0.0
    for h=0:alpha
        for k=0:beta
            for m=0:gamma
                s2 = 0.0
                for i=0:h
                    s3 = 0.0
                    for j=0:k
                        s4 = 0.0
                        for l=0:m
                            s4 += binomial(m,l) * a[3]^(m-l) * b[3]^l *
                                M(h+k+m-i-j-l, i+j+1)
                        end
                        s3 += binomial(k,j) * a[2]^(k-j) * b[2]^j * s4
                    end
                    s2 += binomial(h,i) * a[1]^(h-i) * b[1]^i * s3;
                end
                s1 += binomial(alpha,h) * binomial(beta,k) * binomial(gamma,m) *
                    vo[1]^(alpha-h) * vo[2]^(beta-k) * vo[3]^(gamma-m) * s2
            end
        end
    end
    c = cross(a,b)
    if signedInt == true
        return s1 * norm(c) * sign(c[3])
    else
        return s1 * norm(c)
    end
end
```

Permette di calcolare l'integrale di un triangolo implementando la seguente formula:

$$\begin{aligned}
 II_{\tau}^{\alpha\beta\gamma} = & II^{uv} |a \times b| \cdot \\
 & \cdot \sum_{h=0}^{\alpha} \binom{\alpha}{h} x_0^{\alpha-h} \sum_{k=0}^{\beta} \binom{\beta}{k} y_0^{\beta-k} \sum_{m=0}^{\gamma} \binom{\gamma}{m} z_0^{\gamma-m} \cdot \\
 & \cdot \sum_{i=0}^h \binom{h}{i} a_x^{h-i} b_x^i \sum_{j=0}^k \binom{k}{j} a_v^{k-j} b_y^j \sum_{l=0}^m \binom{m}{l} a_z^{m-l} b_z^l
 \end{aligned} \tag{3}$$

Dato il triangolo  $\tau$ , come array di array di vertici di tre dimensioni  $v_o = (x_o, y_o, z_o)$ , estraiamo i vertici  $v_o, v_a, v_b$ . Estratti i vertici calcoliamo i vettori  $a$  e  $b$  e l'equazione parametrica del piano di inclusione del triangolo vale  $p = v_o + u * a + v * b$ .

Possiamo notare come  $s4$  corrisponda all'ultima sommatoria e l'ultimo termine della sommatoria corrisponda alla funzione  $M$ (con  $u = (h+k+m)^\vee(i+j+l)$  e  $v = (i+j+l)$ ),  $s3$  alla penultima sommatoria,  $s2$  alla terz'ultima sommatoria e  $s1$  alle prime tre sommatorie. Alla fine viene eseguito il prodotto vettoriale fra  $a$  e  $b$ .

Le funzioni:

- `function II(P::LAR, alpha::Int, beta::Int, gamma::Int, signedInt=false)::Float64`
- `function III(P::LAR, alpha::Int, beta::Int, gamma::Int)::Float64`

Permettono di calcolare rispettivamente l'integrale doppio e l'integrale triplo di un polinomio che ha come dominio un poligono; l'integrale viene calcolato attraverso la somma delle aree dei triangoli (funzione  $TT$ ) ottenuti dalla suddivisione della sua superficie.

## Esempi

Un esempio di calcolo della superficie può essere il seguente (da cui sono stati omessi gli output):

```
julia> V, VV, EV, FV = makeSurface(10)
```

```
julia> model = (V, (VV, EV, FV))
```

```
julia> P = V, FV
```

```
julia> Plasm.view(Plasm.hpc_exploded(model)(1.1,1.1,1.1))
```

```
julia> surface(P)
```

```
90.0
```

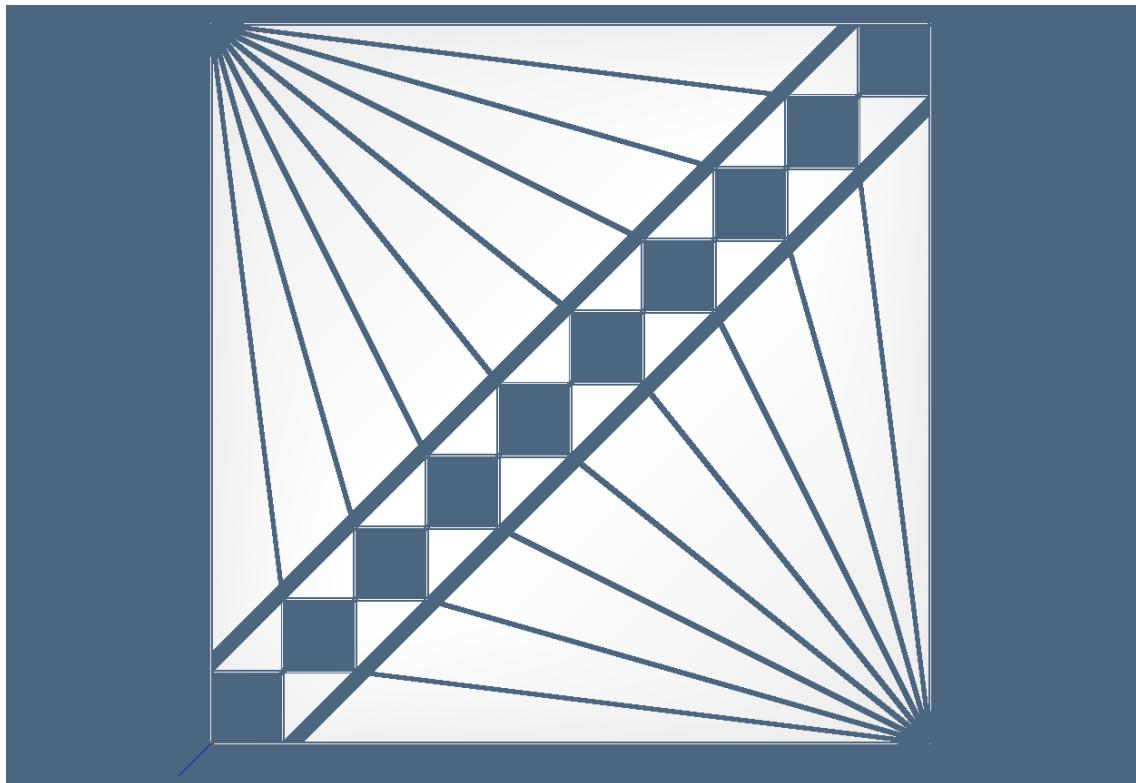


Figure 1: Quadrato

Come riferimento per il calcolo del volume si è preso il modello 3D *Stanford bunny*:

```
julia> V, EV, FV = obj2lar("bunny.obj")  
  
julia> EV = EV[1] julia> FV = FV[1]  
  
julia> VV = [[k] for k=1:size(V,2)]  
  
julia> model = (V, (VV, EV, FV))  
  
julia> P = V, FV  
  
julia> Plasm.view(V, FV)
```

```
julia> volume(P)  
-1.8818908358435506e-5
```

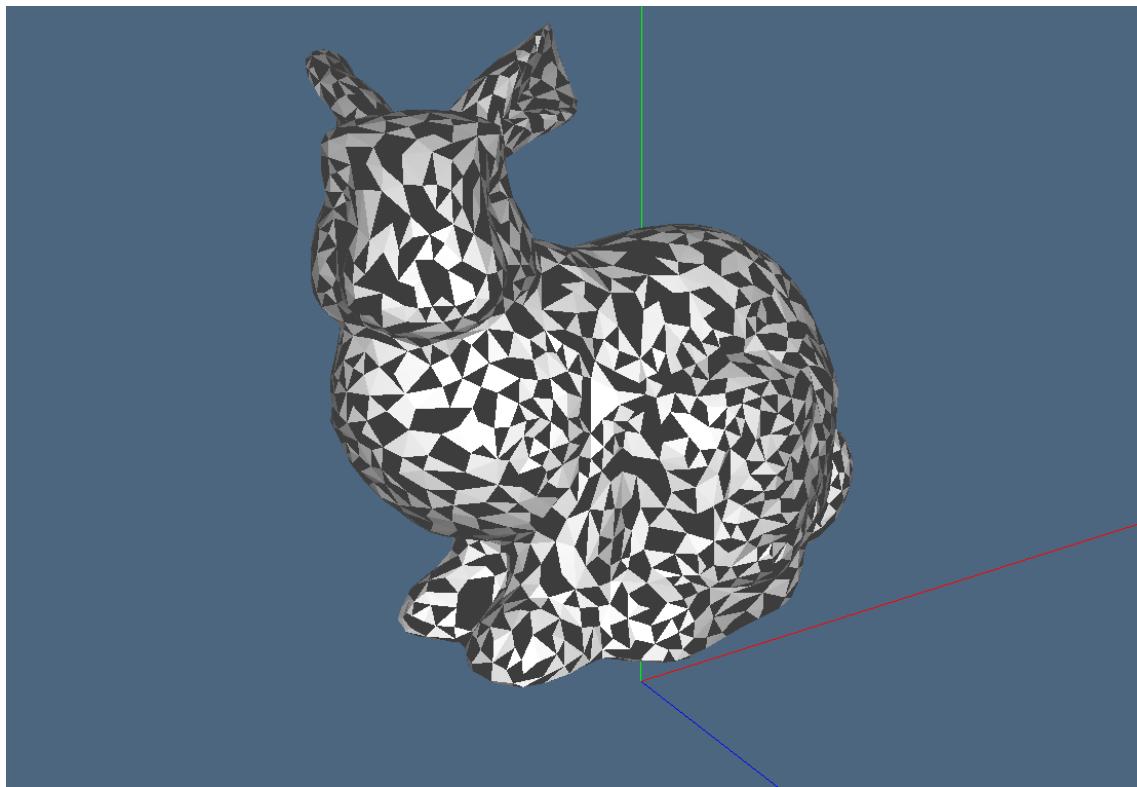


Figure 2: Stanford Bunny