Polyhedral geometry 3

Computational Visual Design (CVD-Lab), DIA, “Roma Tre” University, Rome, Italy

Computational Graphics 2012

Section 1

Examples and Exercises

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)

1. geometric calculus in FL-style

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

> *A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)*

1. geometric calculus in FL-style
2. dimension independence

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

> *A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)*

1. geometric calculus in FL-style
2. dimension independence
3. dynamic typing

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

> *A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)*

1. geometric calculus in FL-style
2. dimension independence
3. dynamic typing
4. higher-level operators

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

> *A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)*

1. geometric calculus in FL-style
2. dimension independence
3. dynamic typing
4. higher-level operators
5. arity: always 1 (number of arguments of functions)

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

> *A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)*

1. geometric calculus in FL-style
2. dimension independence
3. dynamic typing
4. higher-level operators
5. arity: always 1 (number of arguments of functions)
6. small set of predefined functionals

PLaSM Basics

PLaSM = Geometric extension of the FP / FL languages by Backus (IBM Research)

> *A. Paoluzzi, V. Pascucci and M. Vicentino: Geometric Programming: A Programming Approach to Geometric Design. ACM Transactions on Graphics 14(3): 266-306 (1995)*

1. geometric calculus in FL-style
2. dimension independence
3. dynamic typing
4. higher-level operators
5. arity: always 1 (number of arguments of functions)
6. small set of predefined functionals
7. names of functions: all-caps

PLaSM Basics (AA: Apply-to-All)

```
AA(SUM) [[1,2,3],[4,5,6]]
=> [6,15]
```

PLaSM Basics (DISTL: DISTribute-Left)

```
DISTL [2,[1,2,3]]
// => [[2,1],[2,2],[2,3]]

DISTL [2,[]]
// => []
```

PLaSM Basics (TRANS: TRANSpose)

```
TRANS [[1,2,3],[10,20,30],[100,200,300]]
// => [[1,10,100],[2,20,200],[3,30,300]]

TRANS [[1,2,3,4,5],[10,20,30,40,50]]
// => [[1,10],[2,20],[3,30],[4,40],[5,50]]

TRANS [[],[]]
// => []
```

PLaSM Basics (arithmetic ops)

```
MUL [3,4]
// => MUL [3,4] = 12

MUL [[1,2,3],[4,5,6]]
// => MUL [[1,2,3],[4,5,6]] = [4, 10, 18]
```

```
SUM [3,4]
// => SUM [3,4] = 7

SUM [[1,2,3],[4,5,6]]
// => SUM [[1,2,3],[4,5,6]] = [5, 7, 9]
```

PLaSM Basics (product scalar by vector)

```
PROD [3,[1,2,3]]
// => PROD [3,[1,2,3]] = [3, 6, 9]

PROD [4,[10,20,30]]
// => PROD [4,[10,20,30]] = [40, 80, 120]
```

Plasm.js: Exercise 1 (INNERPROD)

The inner (or scalar) product of $a, b \in \mathbb{R}^m$ is a number

$$\mathtt{INNERPROD} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R} : (u, v) \mapsto \sum_{i=1}^{m} \mathbf{u}_i \mathbf{v}_i$$

```
INNERPROD = ([u, v]) -> SUM MUL [u, v]

u = [1,2,3]
v = [10,20,30]
INNERPROD [u, v]
// => 140
```

Plasm.js: Exercise 2 (VECTNORM)

The norm of a vector $a \in \mathbb{R}^m$ is a number.

$$\texttt{VECTNORM} : \mathbb{R}^m \to \mathbb{R} : v \mapsto \sqrt{\sum_{i=1}^{m} \mathbf{v}_i^2}$$

```
VECTNORM = (v) -> Math.sqrt SUM MUL [v, v]

a = [1,2,3]
VECTNORM a
// => 3.7416573867739413
```

Plasm.js: Exercise 3 (UNITVECT)

The unit vector is a function

$$\texttt{UNITVECT} : \mathbb{R}^m \to \mathbb{R}^m : v \mapsto \frac{v}{\|v\|}$$

```
UNITVECT = (v) -> PROD [1/(VECTNORM v), v]

v = [1,2,3]
UNITVECT v
// => [0.2672612, 0.5345224, 0.8017837]

VECTNORM UNITVECT v
// => 1
```

Plasm.js: Exercise 4 (SUM)

SUM adds n vectors in $\mathbb{R}^m$, i.e. the columns of a matrix in $\mathbb{R}^m_n$:

```
a = [1,2,3]
a
// => [1, 2, 3]

b = [10,20,30]
b
// => [10, 20, 30]
```

```
SUM [a,b]
// => SUM [a,b] = [11, 22, 33]
```

Plasm.js: Exercise 5 (SUM)

```
a = [1..10]
a
// => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

b =  (10*k for k in [1..10])
b
// => [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

SUM [a,b]
// => [11, 22, 33, 44, 55, 66, 77, 88, 99, 110]
```

```
c =  (100*k for k in [1..10])
SUM [a,b,c]
// => [111, 222, 333, 444, 555, 666, 777, 888, 999, 1110]
```

Plasm.js: Exercise 6 (MATSUM)

Write a function that adds any two matrices $[A], [B]$ (compatible by sum). both $[A], [B]$ must belong to the same linear space $\mathbb{R}^m_n$

```
MATSUM = (args) -> AA(AA(SUM)) AA(TRANS) TRANS args
```

```
A = [ [1,2,3], [4,5,6], [7,8,9] ]
B = [ [10,20,30], [40,50,60], [70,80,90] ]

MATSUM [A,B]
// => [ [11,22,33], [44,55,66], [77,88,99] ]

MATSUM [A,B,A]
// => [ [12,24,36], [48,60,72], [84,96,108] ]

MATSUM [A,B,B,A]
// => [ [22,44,66], [88,110,132], [154,176,198] ]
```

Plasm.js: Exercise 7 (MATPROD)

Write a function that multiplies two matrices (compatible by product)

Remember that

$$A \in \mathbb{R}^m_n, \quad B \in \mathbb{R}^n_p, \quad \text{and} \quad C = AB \in \mathbb{R}^m_p,$$

with

$$C = \left(\ c^i_j\ \right) = \left(\ \mathbf{A^i B_j}\ \right), \qquad 1 \leq i \leq m, 1 \leq j \leq p,$$

where $\mathbf{A^i}$ is the i-th row of $\mathbf{A}$, and $\mathbf{B_j}$ is the j-th column of $\mathbf{B}$.

Plasm.js: Exercise 7 (MATPROD) – Solution

Write a function that multiplies two compatible matrices

```
MATPROD = ([A,B]) ->
    AA(AA(INNERPROD)) AA(DISTL) DISTR [A, TRANS B]
```

```
A = [[1,2,3],[4,5,6],[7,8,9]]
B = [[1,2,3],[4,5,6],[7,8,9]]
MATPROD [A,B]
// => [ [30, 36, 42], [66,81,96], [102,126,150] ]
```

```
C = [[1,2,3],[4,5,6]]
D = [[1,2],[4,5],[7,8]]
MATPROD [C,D]
// => [ [30,36], [66,81] ]
```

Plasm.js: Exercise 8 (some array operators)

Look at some PLaSM operators on arrays

```
REPEAT(3) 0    # Repeat
// => [0,0,0]
REPEAT(3) [0,1]
// => [ [0,1], [0,1], [0,1] ]

REPLICA(3) [0,1]  # REPeat LIst & CAtenate
// => [ 0,1, 0,1, 0,1 ]
```

```
AR [ [0,0,0], 1 ]  # Append Rigth
// => [0,0,0,1]

AL [ 1, [0,0,0] ] # Append Left
// => [1,0,0,0]
```

Plasm.js: Exercise 9 (identity array)

Write a function to generate the $n \times n$ identity matrix $[\mathbf{I}]$

```
IDNT = (n) ->
    MAT(n,n) AR [REPLICA(n-1)(AL [1, REPEAT(n) 0]), 1]

IDNT 3
// => [ [1,0,0], [0,1,0], [0,0,1] ]
```

note that:

```
MAT(2,3)([1,2,3,4,5,6])
// => [ [1,2,3], [4,5,6] ]

MAT(2,3)("ABCDEF")
// => [ ["A","B","C"], ["D","E","F"] ]
```

Basic use of PLaSM.js

```
# Execution initialization.
# The parameter `rn` is the dimension of the point space.
# Set `rn` to 2 for a 2D example.
rn = 3

# `points` to be classified are randomly generated, and th
# via the translation method `.t()`
points = randomPoints(rn, m=2000*Math.pow(2,rn), scale=8).

# The `object` to draw is initialized to the empty array
object = SimplicialComplex [points, AA(list) [0...points.le

# The `draw` method of the `viewer` is applied to the `obj
model = viewer.draw object
```

Setting the environment for PLaSM.js

1. Download `plasm.js` from the repository:

```
git clone git://github/ ...
```

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>test</title>
</head>
```

Setting the environment for PLaSM.js

1. Download `plasm.js` from the repository:

```
git clone git://github/ ...
```

1. Move in the `test` directory, and set `index.html` to this content:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>test</title>
</head>
```

continue . . .

1. continue . . .

```
<body>
  <script src="../support/numeric.js"></script>
  <script src="../support/three.js"></script>
  <script src="../support/detector.js"></script>
  <script src="../support/request-animation-frame.js"></sc
  <script src="../support/stats.js"></script>
  <script src="../lib/enhancedtrackballlightcontrols.js"><
  <script src="../lib/plasm.js"></script>
  <script src="../lib/plasm-init.js"></script>
  <script src="../lib/simplexn.js"></script>
  <script src="../test/jourfile.js"></script>
</body>
</html>
```

Plasm.js: Exercise 11 (VECTPROD)

the vector product $\mathbf{w}$ of vectors in $\mathbb{R}^3$ id defined as the function

$$\mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3 : (\mathbf{u}, \mathbf{v}) \mapsto \det \begin{pmatrix} \mathbf{e}_0 & \mathbf{e}_1 & \mathbf{e}_2 \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{pmatrix}$$

Therefore we can write, for the vector product of two 3D vector:

```
VECTPROD = ([u,v]) ->
    w = new Array(3)
    w[0] = u[0]*v[1] - u[1]*v[0]
    w[1] = u[2]*v[0] - u[0]*v[2]
    w[2] = u[0]*v[2] - u[2]*v[0]
    w

VECTPROD [[1,0,0], [0,1,0]]
\\ => [0,0,1]
VECTPROD [[1,1,0], [0,1,0]]
\\ => [0,0,1]
```

Plasm.js: Exercise 12 (1/4)

```
randomPoints = (m, sx=1, sy=1) ->
    point = () -> [Math.random() * sx, Math.random() * sy]
    new PointSet( point() for k in [0...m] )

points = randomPoints(200, 2*Math.PI, 2)
obj = new SimplicialComplex points.verts, AA(LIST) [0...points.m

model = viewer.draw obj
```

Plasm.js: Exercise 12 (2/4)

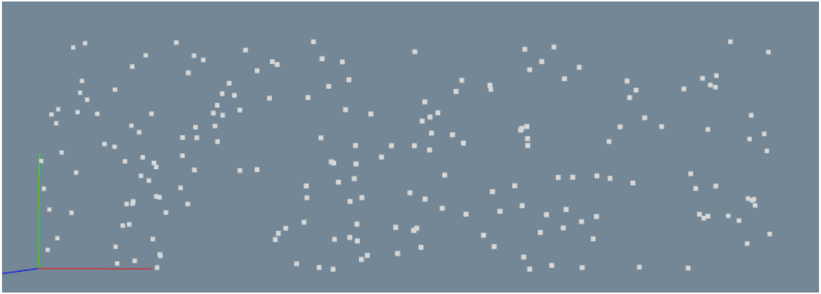

Figure: 200 random points in $[0, 2\pi] \times [0, 2] \subset \mathbb{E}^2$

Plasm.js: Exercise 12 (3/4)

coordinate functions

```
x = ([u,v]) -> v * Math.cos u
y = ([u,v]) -> v * Math.sin u
obj = MAP([ x,y ]) obj

model = viewer.draw obj
```

Plasm.js: Exercise 12 (4/4)

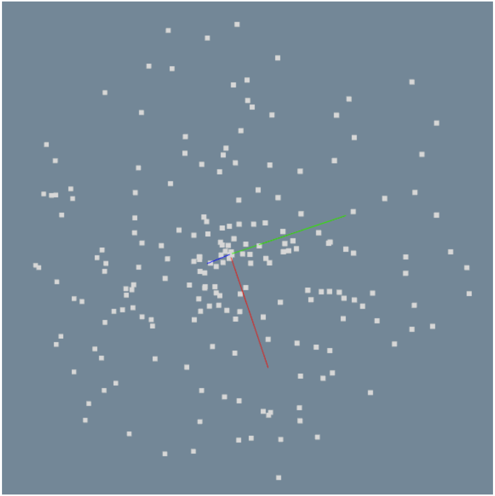

Figure: 200 random points within the 2D "ball" of radius 2

Plasm.js: Exercise 13 (4/4)

Map the previously constructed 2D random points to affine coordinates

Plasm.js: Exercise 14 (implement the Gift wrapping in 2D)

Of course, you need to draw your input data and your partial geometric constructions, in order to write and to test your algorithm . . . :o)

Current plasm.js Library

AA
AL
APPLY
AR
BIGGER
BIGGEST
BOUNDARY
BUTLAST
CART
CAT
CENTROID
CIRCLE
CLONE
CODE
COMP
CONS
CUBE
CUBOID
CYLSOLID
CYLSURFACE
DISK
DISTL
DISTR
DIV
EMBED
EXPLODE
EXTRUDE
FIRST
FREE
Graph
GRAPH
HELIX
ID
IDNT
IDNT
INNERPROD
INSL
INSR
INTERVALS
INV
ISFUN
ISNUM
K
LAST
LEN
LINSPACE1D
LINSPACE2D
LINSPACE3D
LIST
MAP
MAT
MATPROD
MATSUM
MUL
PointSet
POLYLINE
POLYMARKER
PRECISION
PRINT
PROD
PROGRESSIVE_SUM
QUADMESH
R
REPEAT
REPLICA
REVERSE
S
S0
S1
S2
S3
S4
SET
SIMPLEX
SIMPLEXGRID
SimplicialComplex
SKELETON
SMALLER
SMALLEST
SORTED
SUB
SUM
T
TAIL
Topology
TORUSSOLID
TORUSSURFACE
TRANS
TREE
TRIANGLEARRAY
TRIANGLEFAN
TRIANGLESTRIP
UNITVECT
VECTNORM
VECTPROD