1. We want surface close to points.

2. Network of Bezier Patches can be described as multi-input function:

$$\vec{S}(u, v, f, \{\vec{P}_f\}) = \sum_{\vec{P}_{ij} \in \{\vec{P}_f\}} \overbrace{b_{i,n}(u)\, b_{j,n}(v)}^{\text{bernstein polynomials}} \vec{P}_{i,j}$$

($\vec{P}_{i,j}$: Bez pt, local one)

- u, v: params on 1 patch
- f: which patch
- $\{\vec{P}_f\}$: a number of Bez points related to patch f

$$[\vec{P} \Leftrightarrow \vec{B}^{Bez}]$$

3. We have datapoints $\vec{d}_k$ with assigned values u_k, v_k, f_k. We want to minimize the distance e_k between datapoint $\vec{d}_k$ and point on surface $\vec{S}(u_k, v_k, f_k, \{\vec{P}_{f_k}\})$

$$\text{distance } e_k^2 = \| \vec{d}_k - \underbrace{\vec{S}(u_k, v_k, f_k, \{\vec{P}_{f_k}\})}_{\vec{S}_k} \|_2^2$$

4. For fixed values of u_k, v_k (and f_k), we can treat $(b_{i,n}(u_k) b_{j,n}(v_k))$ as constants c_{ijk}. Let's also call surface evaluation $\vec{S}$ at u_k, v_k, f_k $\vec{S}_k$

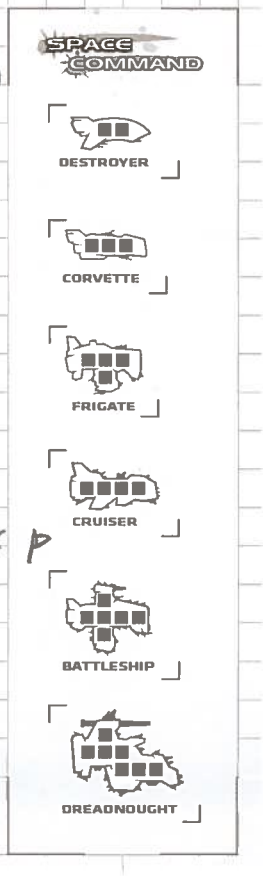
$$\vec{S}_k = \sum \underbrace{b_{i,n}(u_k) b_{j,n}(v_k)}_{c_{ijk}} \vec{P}_{i,j} =$$

$$= \sum c_{ijk} \vec{P}_{i,j} \Rightarrow \text{Reindex } \{i,j\} \text{ to a combined index } p$$

$$= \sum c_{pk} \vec{P}_p \equiv \underbrace{C}_{\text{coef matrix } C} \vec{P}$$

($\vec{P}$: Bezier point matrix)

looks like matrix-vector!



5. ... Which means that distance now looks like

$$\text{distance } e_k^2 = \| \vec{d}_k - C_{k:} \vec{\vec{P}} \|_2^2$$

[choose row k for data point k]

$\Rightarrow$ standard Least-squares problem!!

BUT: not yet continous/smooth.

Peter's scheme 2nd step: [after Doo-Sabin]

6. Peter's scheme says: Start with [after Doo-Sabin] points $\vec{Q}$. Create Bezier points $\vec{P}$ as linear combinations of $\vec{Q}$ and you'll get smoothness.

$$\Rightarrow \vec{P}_p = \sum a_{pl} \vec{Q}_l$$

(a_{pl}: linear coef from Peters scheme for $\vec{P}_p$ on $\vec{Q}_l$)

Again, this gives matrix-vector!

$$\vec{\vec{P}} = A \vec{\vec{Q}}$$

Combined least squares

7. We get for surface point $\vec{s}_k$ now

[a] $$\vec{s}_k = C_{ki} \vec{P} = C_{ki} A \vec{Q}$$

(insert from 6) (makes smooth!)

[7b. Whole formulation: create $\vec{S}$ from all points, $\vec{D}$ likewise

$$\vec{S} = CA\vec{Q}$$

find $\vec{Q}$ such that

$$E^2 = \| \vec{D} - \vec{S} \|_2^2 = \| \vec{D} - CA\vec{Q} \|_2^2$$

is minimized.

The Algorithm

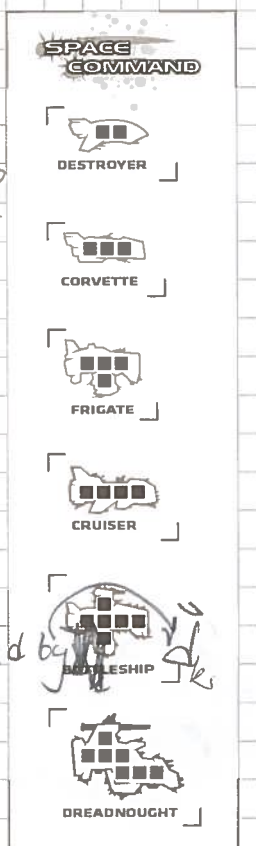
8. What do we need to do to find $\{\vec{P}_f\}$?

a.) sort $\vec{d}_k$ into matrix $\vec{D}$

b.) sort $\vec{Q}_l$ into matrix $\vec{Q}$

c.) build matrix C

i.) sort parameters (u_k, v_k, f_k) into same as is used by $\vec{d}_k$



8c) ii) Evaluate bernstein polynomials $b_{i,n}(u_k) b_{j,n}(v_k)$ to get c_{ijk}

iii) sort it into C by making the reindexing $\{i,j,f\} \to p$

d.) calculate A

Peter's scheme

i.) Find the local formulation of a_{pl}: a_{pl} is only nonzero for a few l for every p / for a few p for every l.

+) For ordinary pts in 3x3 patches

++) For extraordinary pts in 4x4 patches

ii) Check sortings of $\vec{P}_p$ and $\vec{Q}_l$ to put a_{pl} inthe right place in the matrix.

e.) Combine into CA. [can be done matrixless until here]

f.) Solve least-squares for $\vec{\vec{Q}}$.

g.) Find $\vec{\vec{P}} = A\vec{\vec{Q}}$

i.) Extract points $P_{i,j}$ in f from the indexing $\{p\} \leftarrow \{i,j,k\}$

j) Output to CAD - processing/plotting.

Matlab algorithm:

runningScript: calls loadingScript that loads and prepares input data ([a.) & b.) and c.) i)]
- creates CA, solves and outputs.

loadingScript: loads files + preprocesses data.

a.): determined by order in .csv file [from parametrisation] - sanitized by scaleAwayPa
b.) is done partly by Anna's algorithm, this is checked and changed in sortAB1B2VIndices
c. i) also done by structure of parametrisation files

runningScript calls createGlobalControlMeshCoefs to build CA (steps c) - e), main part of algorithm)

→ first build local matrices of A for the extraordinary (bicubic) cases [since they're complicated] for re-use in building CA. → create BicubicCoefMatrices [biquad/regular is easy → call function to build for each data point SHOULD BE CHANGED] [d. i)]

- second, Execute main CA-building loop:

Construct one row at a time $\Longleftrightarrow$ one datapoint in $\vec{D}$.

- check its parameters to find which Quad and ordinary/extraordinary + scale/rotate quad-params to patch-params = createLocalParamsExtraordinaryPatch
- call getExtraOrdCornerIndexMask / get3x3ControlPointIndexMask to get the appropriate connections: steps c.iii) and d.ii) [using data from Anna's Algorithm]
- call getBicubicBezierPointCoefs / getBezierPointCoefs to create appropriate row of C [by evaluating Bernstein polys for local coords] step c.ii)
- multiply together for the elts of CA steps d.ii), e)

end CA-building-loop

then: runningScript solves LinearLeastSq. by

$$\vec{\vec{Q}} = (CA) \backslash \vec{D} \qquad (f.))$$

then $\vec{P} = A\vec{Q}$ and output is done in plotting scripts by building each row of A as needed, just like in createGlobalControlMeshCoefs by:

- call get...IndexMask to get index connections
- call getBiquadraticPatchCoefs / use matrices from createBicubicCoefMatrices to get coefs in A

h.) , i.)

Then, output !!

Suggestion:

Instead of (building &) using large matrices to keep track of the different indexings in the $\vec{D}-C_{\vec{P}}-A-\vec{Q}$ chain, let the objects/vertices keep track of their neighbours and then this information can be extracted as needed in steps c.i), iii), d. ii), e.), g.) and i.).

Categorisation of Peter's scheme functions

M - necessary for MATLAB's way of handling data
[indexing etc. part, could be omitted in an OO treatment]

E - essential part of algorithm

B - bezier

AP - plotting

T - testing

Alphabetically

bernstein - BE

bezier - P

bezier_interactive -

bezierRational -

bicubicPatchCoefTest -BT

bincoeff -(E)

checkB1B2OrientationReversal - M(E)

checkB1B2Reversal

checkB1B2Reversal_opt

createBezierPointMatrices - M(E)B

createBicubicCoefMatrices - BE

createCoefsMatrix -

createExtraOrdCoefsMatrix - MBT

createGlobalControlMeshCoefs - E

createLocalParamsExtraordinary - MEB

createQuadPatchIndices - MT

createTorusParams - T

example -

exampleCircle -

extraOrdCorner - BTPP

ExtraOrdExtend - BTP

ExtraOrdTorusTest - BTP

get3x3ControlPointIndexMask - E(M)

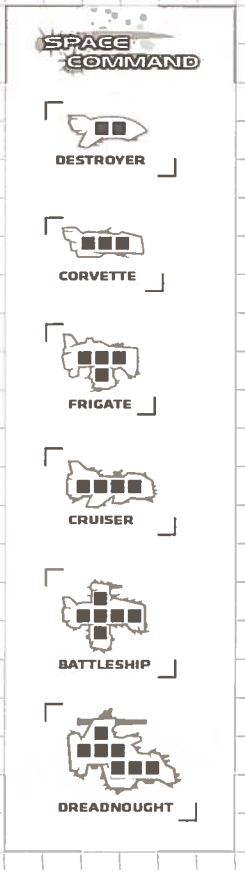
{ getBezierPointCoefs -BE
{ getBicubicBezierPointCoefs -BE

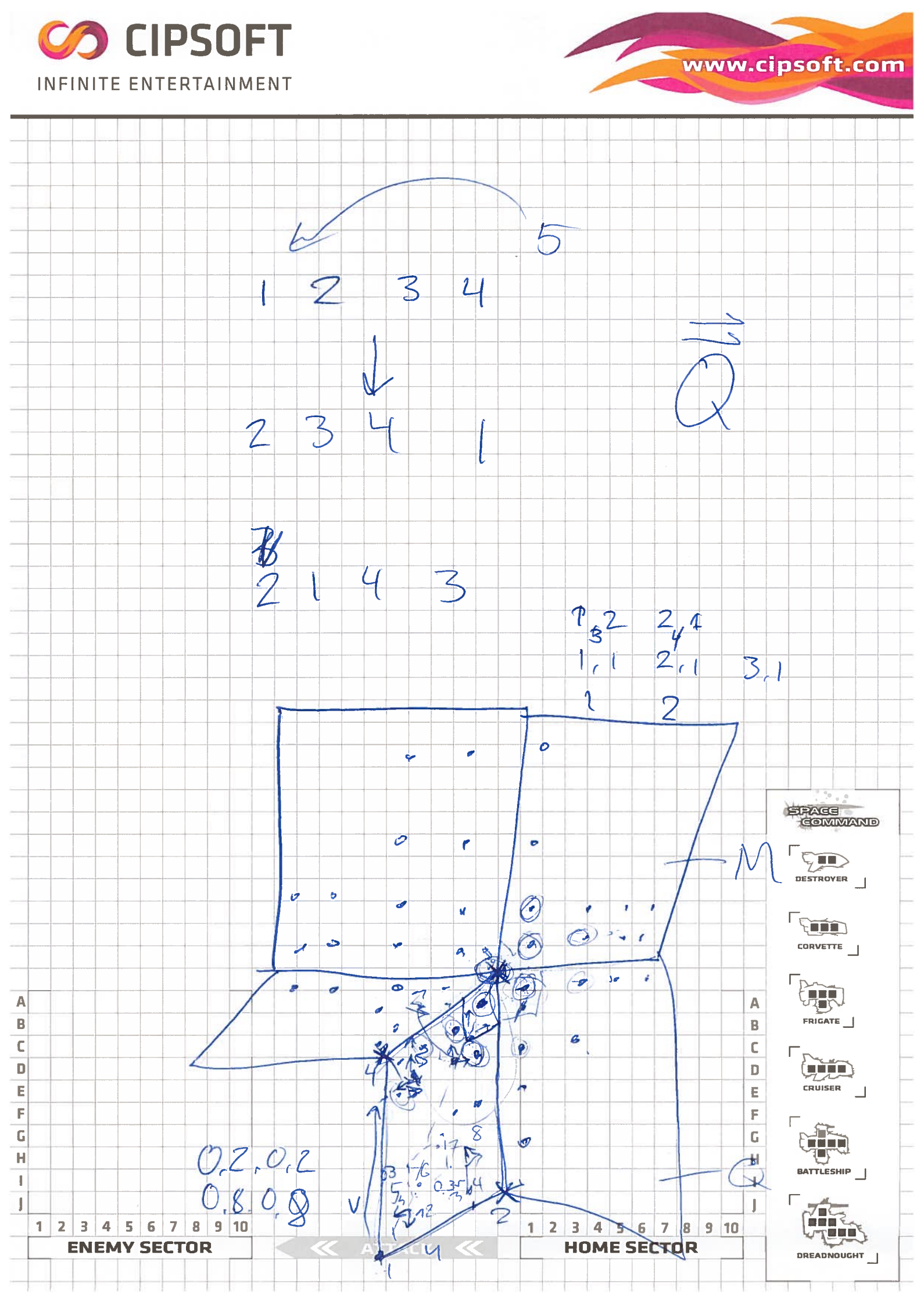
(getBicubicPatch) -B(E)TP
getBicubicPatchIndex -BETP
getBiqubicPatch -BETP
getBiqubicPatchCoefs -BTE

{ getCellAlongEdge - M(E)
{ getCellsAlongEdge - M(E)

getExtraOrdCornerIndexMask - ME
getNeighboursSharedEdge -M(E)
(getNumOfEdgesMeeting) -M(E)
getNumOfEdgesMeetingMatlab -M(E)
getPatchPointOnQuad -P
getVertexIndex -
loadingScript -M((E))

multiplyVerts	-T
petersPlot	-TP
plotAllHairsOnQuad	-P
plotAsBsCs	-P
plotBezierSurfaceWhole	-P
plotDataPoints	-P
plotLinesOnQuad	-P
plotOneQuadWhole	-P
plotPatchesOnQuad	-P
plotPatchPointsOnQuad	-P
plotPoints	-TB
plotQuad	-P
plotQuadOrQuadsWithHairs	-PT
projectOnQuadrant	-E
rotateB2	-MT
rotateB2FromB1	-MT(E)
rotateB2Matlab	-MT
runningScript	-EMTP...
scaleAwayParameters	-T(E)
scaleParameters	-T
shiftToRegularPointsFormat	-M
sortABIB2IVIndices	-M(E)T
sortBIB2Indices	-MT
throwAwayParameters	-T
toUsPointScript	-TP




5
1 2 3 4
2 3 4 1
Q
2 1 4 3
1,1 2,1 3,1
1 2
M
Q
0,2 0,2
0,8 0,8
A B C D E F G H I J
1 2 3 4 5 6 7 8 9 10
ENEMY SECTOR
ATTACK
HOME SECTOR
SPACE COMMAND
DESTROYER
CORVETTE
FRIGATE
CRUISER
BATTLESHIP
DREADNOUGHT