

Digital Micrography

Ron Maharik

Mikhail Bessmeltsev

University of British Columbia

Alla Sheffer

University of British Columbia

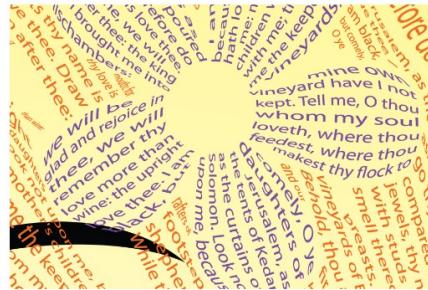
INRIA Rhône-Alpes

Ariel Shamir

The Interdisciplinary Center

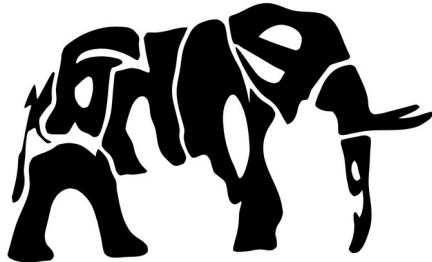
Nathan Carr

Adobe Systems Incorporated



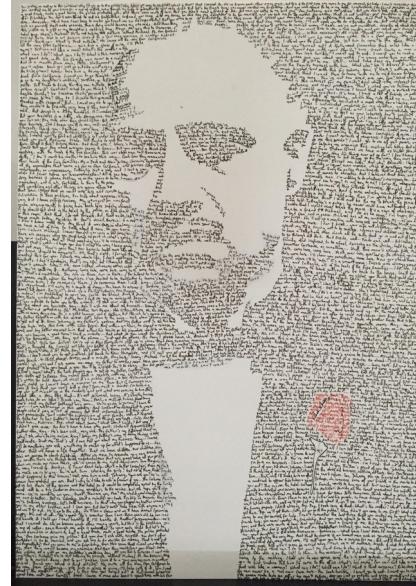
“Micrography”?

Calligrams



Calligraphic Packing by Jie Xu Craig S. Kaplan 2007

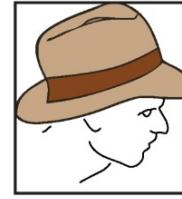
Micrograms



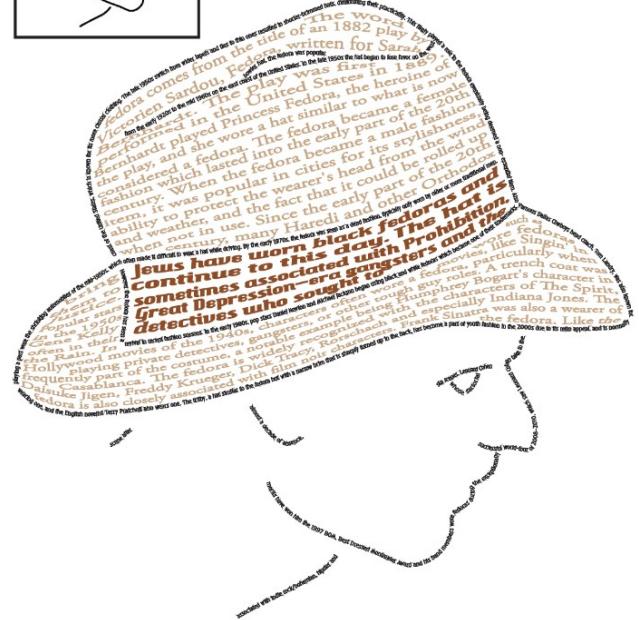
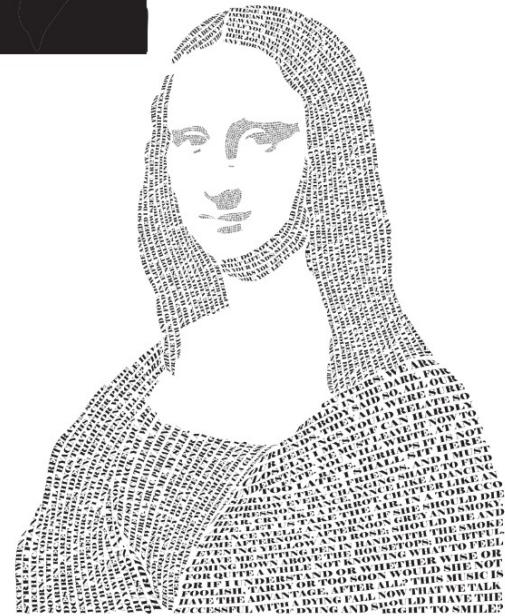
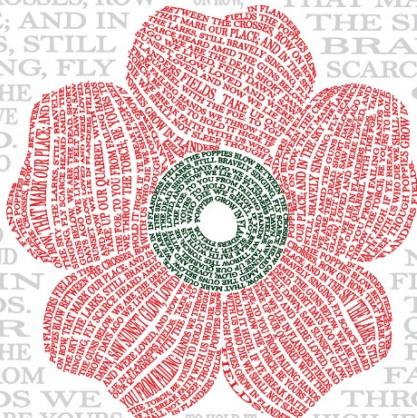
Vector images + Text



Microgram



IN FLANDERS FIELDS THE POPPIES BLOW BETWEEN
THE CROSSES, ROW ON ROW, THAT MARK OUR
PLACE; AND IN LARKS, STILL SINGING, FLY
AMID THE GUNS
BRAVELY SCARCE HEARD
GUNS ARE THE SHORT WE LIVED,
SAW GLOW, WERE NOW WE
FLANDERS TAKE UP QUARREL
FOE: TO FAILING
THROW THE HIGH. IF YE BREAK
FAITH WITH US WHO DIE WE SHALL NOT SLEEP,
THOUGH POPPIES GROW IN FLANDERS FIELDS.



Readable Text

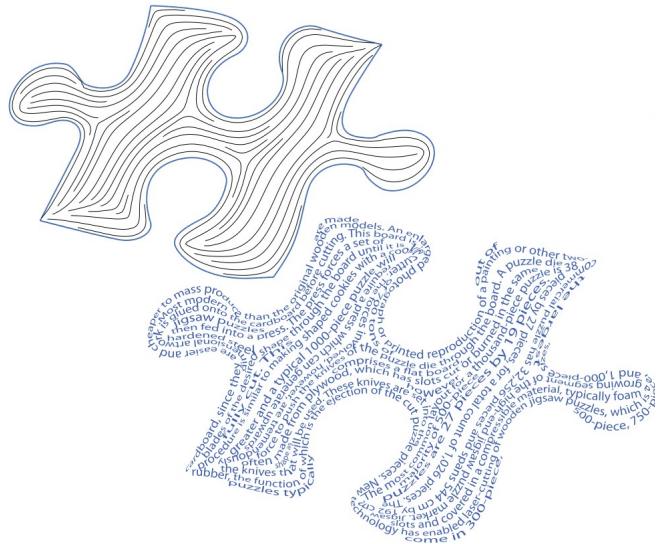
L
OR
EM IP
SUM D
OLOR SIT
AMET CO
NSECU
R ADIPISCIN
G ELIT, SED D
O EUISMOD TE
MPOR INCIDU
NT UT IURE DOL
T DOLORE MAGNA
ALIQUA. UT ENIM A
D MINIM VENIAM, Q
UIS NOSTRUD EXERC
ITATIUS LABORUM DOLAB
ORIS NISI UT ALIQUA E
X EA COMMODO CONSE
QUAT. DUS AUTE IRURE
DOLOR IN PR
EHENDIT
RIT T DOLUPIT. T DOL
ESSE CILLUM D
OLOR E
U FUGIT NULL
A PARAT
UR EXCEPTEU
R SINT Q
CCOCAT CIPID
ATATI ON
PRODESI SUT
IN ALP
QUI OFFICIA DES
ERUNT MO
LLIT ANIM ID EST
ATIS UNDE
SIT PERSICI
DAMNATI FUS
DOLUPATATEM ACCUS
OREMOUE LAUDANT
EM APERIAM, EAQUE I
LLO INVENTORE VERT
ARCHITECTUS VI
EXPLICABO. NEMO ENIM
EM QUA VOLUPTA SIT AS
DIT AUT FUGIT. SED QUA CON
NI DOLOREM IPS
NESCIENT. NEQUE P
ORRO QUISQUAM EST, QUI DOLOREM IPS
T AMET, CONSEQUETUR, ADIPISCİ VELIT, SED QUA NON NUMQUAM
RA INCIDUNT LABORE ET DOLORE MAGNAM ALIQUAM QUÆRA
T ENIM AD MINIMA VENIAM, QUI NODUSUM EXERCITATIONEM U
SUSCIPIT. APERIAM, NISI UT ALIQUID EST, DOLOREM OJOI CO
QUIS AUTEM VEL EUM IURE REPREHENDERIT QUI IN EA V
ATE VELIT ESSE QUAM NIL, MOLESTIAE CONSEQU
ATUR, VEL ILLUM QUI DOLOREM EUM FUGIA
T QUO VOLUPTA NULLA PARAT
UR?

LOREM IPSU
DOLOR SIT AMET, CONSEQUETUR, ADIPISCING EIT SED DO EUISMOD T
UT LABORE ET DOLORE MAGNAM ALIQUAM QUÆRA C
NOSTRUD EXERCITATION ULLAMCO LABORUM DOL
DOLOR IN REPHEHENDIT, SUNT QUA VOLUPTA VEL DOL
CUPIDITATEM, SED QUID, SIT QUA VOLUPTA VEL DOL
REM APERIAM, EAQUE IPSA QUA AB ILLO ASPERNARE, SED Q
WPSAM VOLUPTATEM QUA VOLUPTA QUASQ
NEQUI NESCIENT. NEQUE PORRO QUISQUAM EST, QUI D
LAFERA ET DOLORE MAGNAM ALIQUAM QUÆRA C
MINIMA VENIAM, QUI NODUSUM EXERCITATIONEM U
SUSCIPIT. APERIAM, NISI UT ALIQUID EST, DOLOREM OJOI CO
QUIS AUTEM VEL EUM IURE REPREHENDERIT QUI IN EA V
ATE VELIT ESSE QUAM NIL, MOLESTIAE CONSEQU
ATUR, VEL ILLUM QUI DOLOREM EUM FUGIA
T QUO VOLUPTA NULLA PARAT
UR?

LOREM IPSU
DOLOR SIT AMET, CONSEQUETUR, ADIPISCING EIT SED DO EUISMOD T
UT LABORE ET DOLORE MAGNAM ALIQUAM QUÆRA C
NOSTRUD EXERCITATION ULLAMCO LABORUM DOL
DOLOR IN REPHEHENDIT, SUNT QUA VOLUPTA VEL DOL
CUPIDITATEM, SED QUID, SIT QUA VOLUPTA VEL DOL
REM APERIAM, EAQUE IPSA QUA AB ILLO ASPERNARE, SED Q
WPSAM VOLUPTATEM QUA VOLUPTA QUASQ
NEQUI NESCIENT. NEQUE PORRO QUISQUAM EST, QUI D
LAFERA ET DOLORE MAGNAM ALIQUAM QUÆRA C
MINIMA VENIAM, QUI NODUSUM EXERCITATIONEM U
SUSCIPIT. APERIAM, NISI UT ALIQUID EST, DOLOREM OJOI CO
QUIS AUTEM VEL EUM IURE REPREHENDERIT QUI IN EA V
ATE VELIT ESSE QUAM NIL, MOLESTIAE CONSEQU
ATUR, VEL ILLUM QUI DOLOREM EUM FUGIA
T QUO VOLUPTA NULLA PARAT
UR?

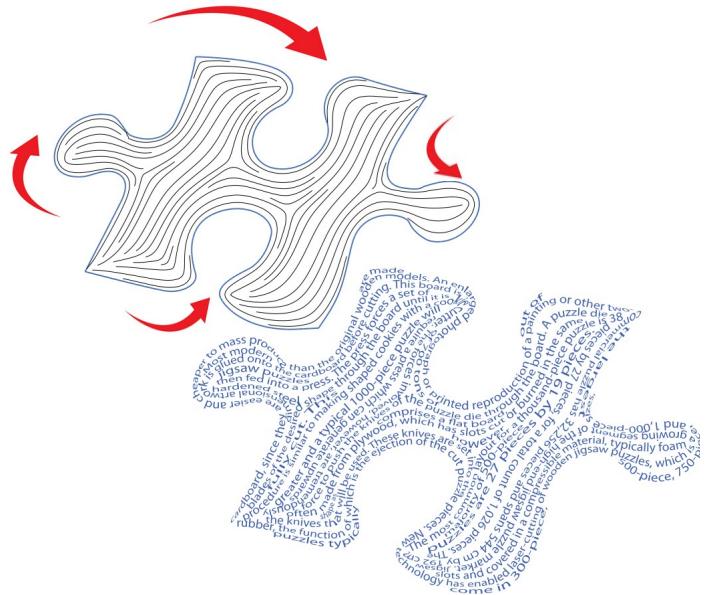
First attempt

Forcing alignment to boundaries



First attempt

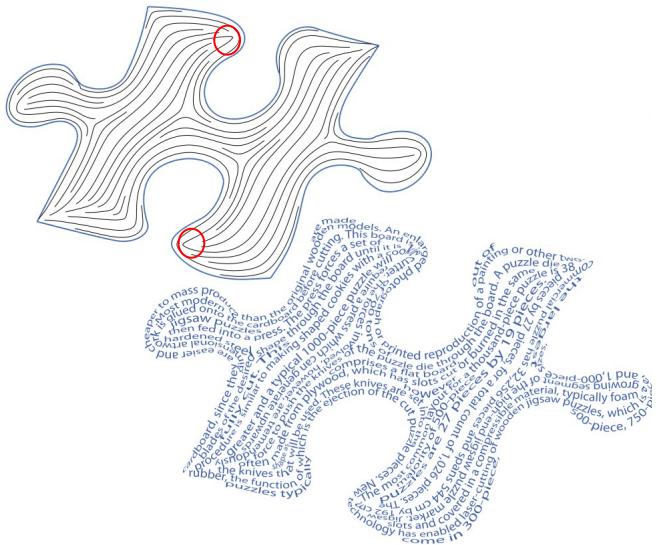
Forcing alignment to boundaries



No orientation

First attempt

Forcing alignment to boundaries

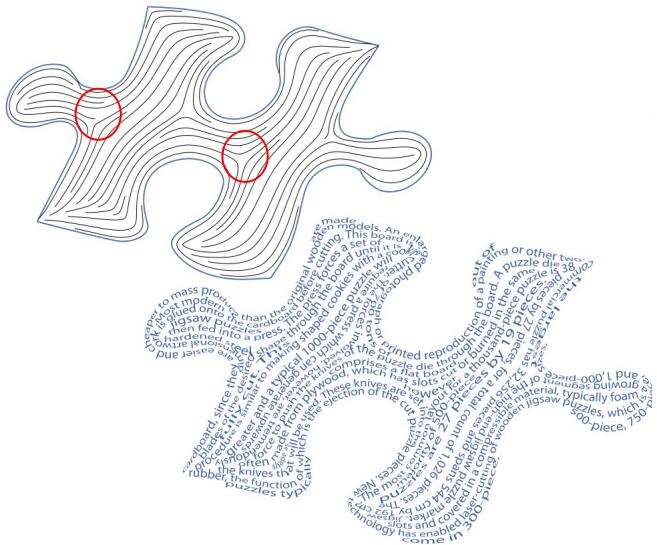


No orientation

High Curvature

First attempt

Forcing alignment to boundaries

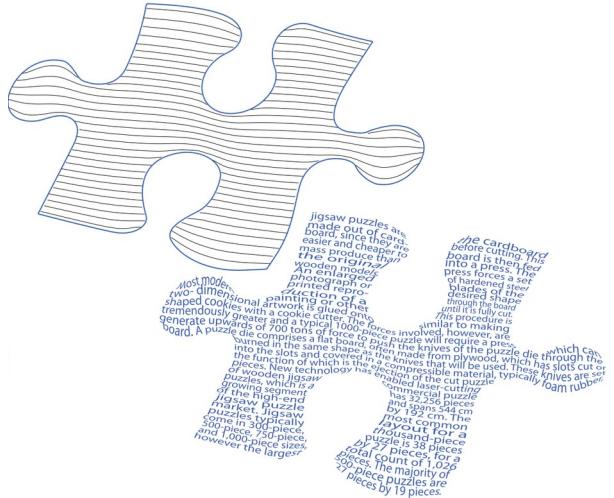


No orientation

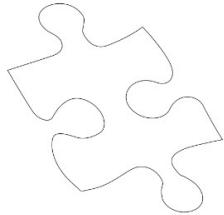
High Curvature

Singularities

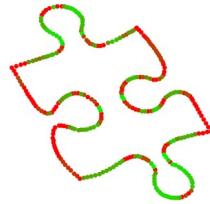
With smart alignment constraints



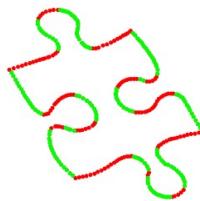
Algorithm pipeline



A



B



C

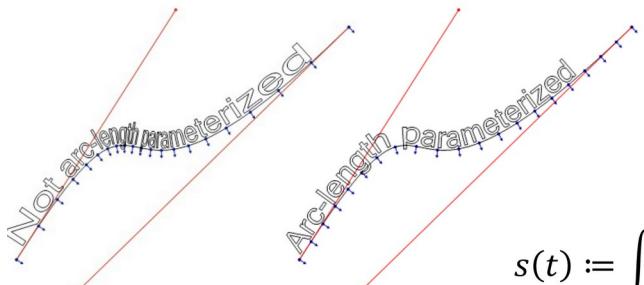


D

Algorithm pipeline

PARAMETERIZATION BY ARC LENGTH

<http://www.planetclegg.com/projects/WarpingTextToSplines.html>

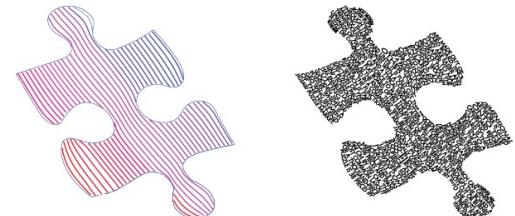


$$s(t) := \int_{t_0}^t \|\gamma'(t)\| dt$$

$$t(s) := \text{inverse of } s(t)$$

$$\gamma(s) := \gamma(t(s))$$

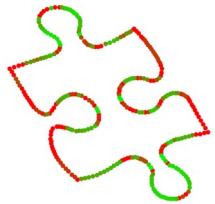
Constant-speed parameterization



D

Boundary condition

Orthogonal vs Aligned



B

Edge weights

$$w_{ij} = F_a(a_{ij}) \cdot F_d(d_{ij})$$

Each colored vertex has an label weight going from 0 (red) to 1 (green)

$$F_a(a_{ij}) = \tanh\left(\frac{4}{\pi}\left(|\frac{\pi}{2} - a_{ij}| - \frac{\pi}{4}\right)\right) \quad F_d(d_{ij}) = e^{-\tilde{d}_{ij}^2/\sigma_d^2}$$

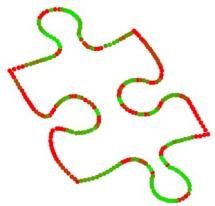
Where : a_{ij} = angle between normal of vertices i,j

d_{ij} = distance between vertices i,j

$$\sigma_d = 10$$

Boundary condition

Orthogonal vs Aligned



B

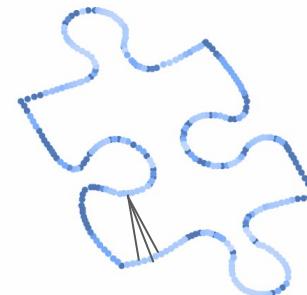
Vertice attraction weights

$$\alpha_i = F_a^+(a_i) \cdot F_d^+(d_i)$$

$$F_a^+ = \max(0, F(a_i))$$

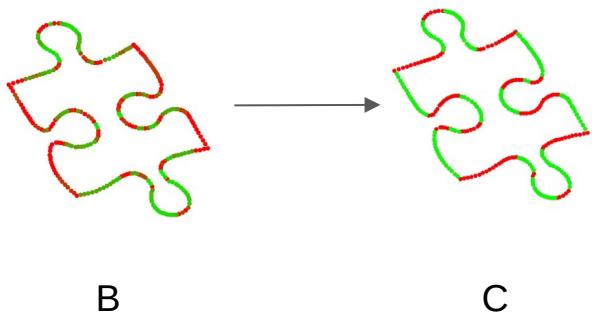
$$\sigma_d = 20$$

Each colored vertex has an label weight going from 0 (red) to 1 (green)



Boundary condition

Orthogonal vs Aligned



Each colored vertex has an label weight going from 0 (red) to 1 (green)

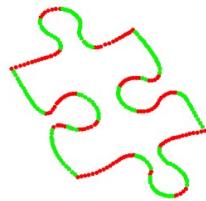
Vertice attraction weights

$$\min \sum_{ij} w_{ij} (l_i - l_j)^2 + \omega \sum_i \alpha_i (1 - l_i)$$

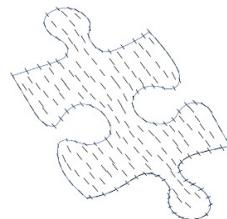
subj. to $0 \leq l_i \leq 1$

can be negative

Vector field



C



D

1. Triangulate shape
2. Compute vector field
3. Trace text lines

Vector field

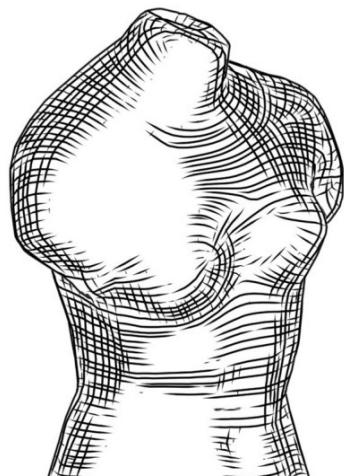
Rotational Symmetry Field Design on Surfaces

Jonathan Palacios*

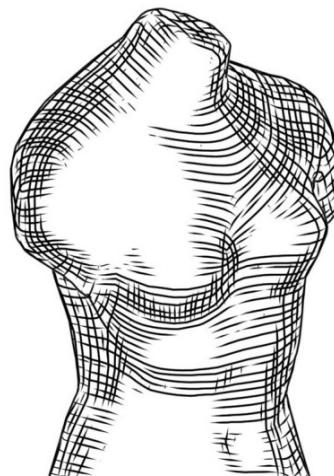
Oregon State University

Eugene Zhang*

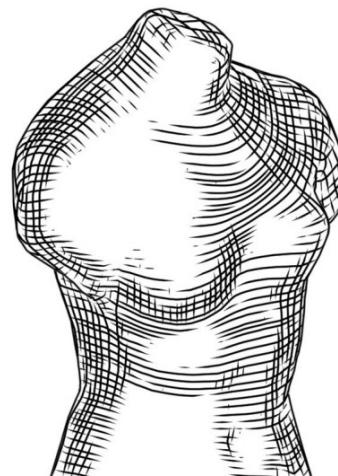
Oregon State University



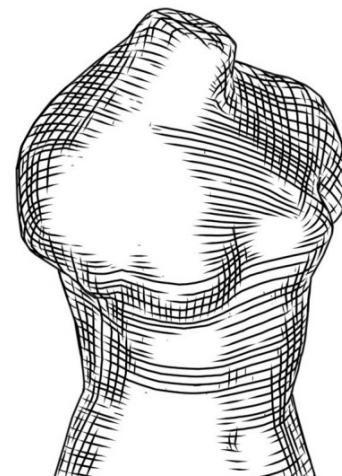
(a)



(b)

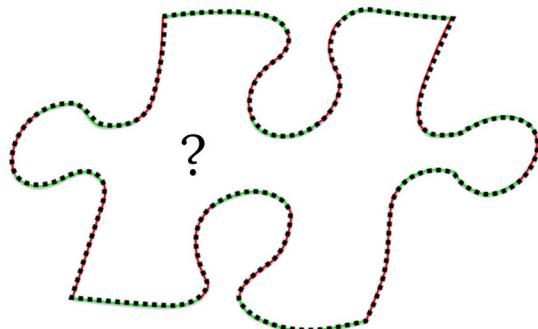


(c)



(d)

Inside?



Inside?

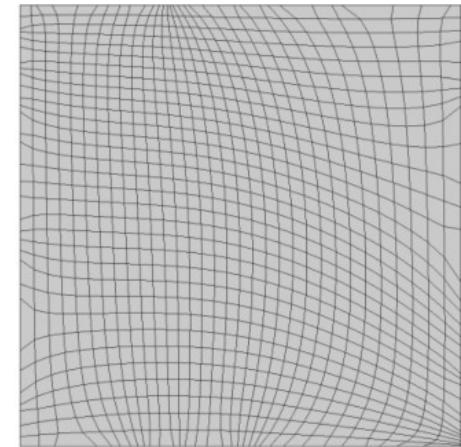
- Smoothest interpolation of boundary values
- Laplace equation with Dirichlet boundary conditions
- Discretization?
- Representation?

$$\Delta u = 0$$
$$u \Big|_{\partial\Omega} = v$$

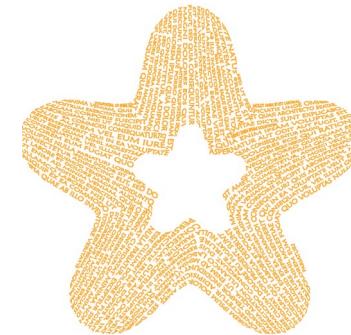
Laplacian smoothing!

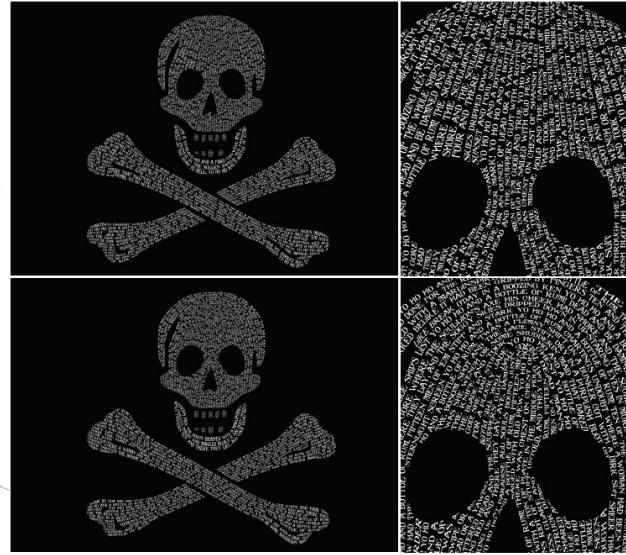
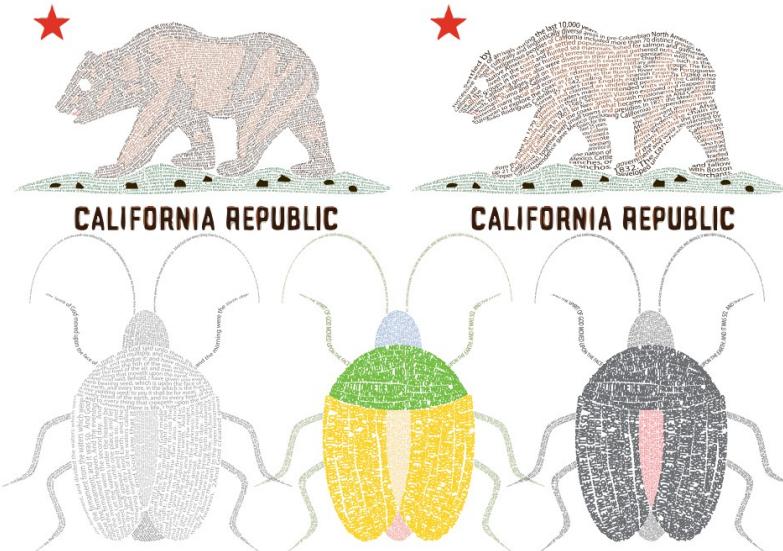
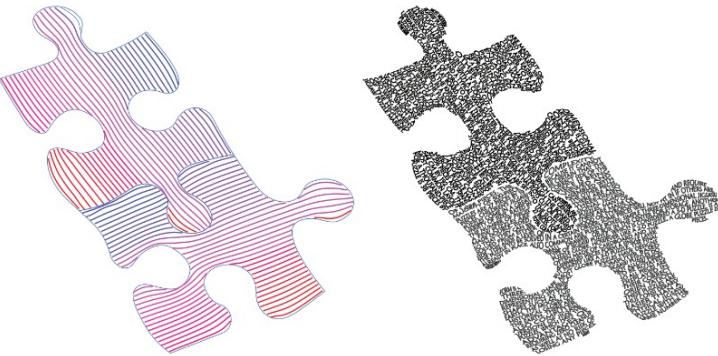


(a) before smoothing

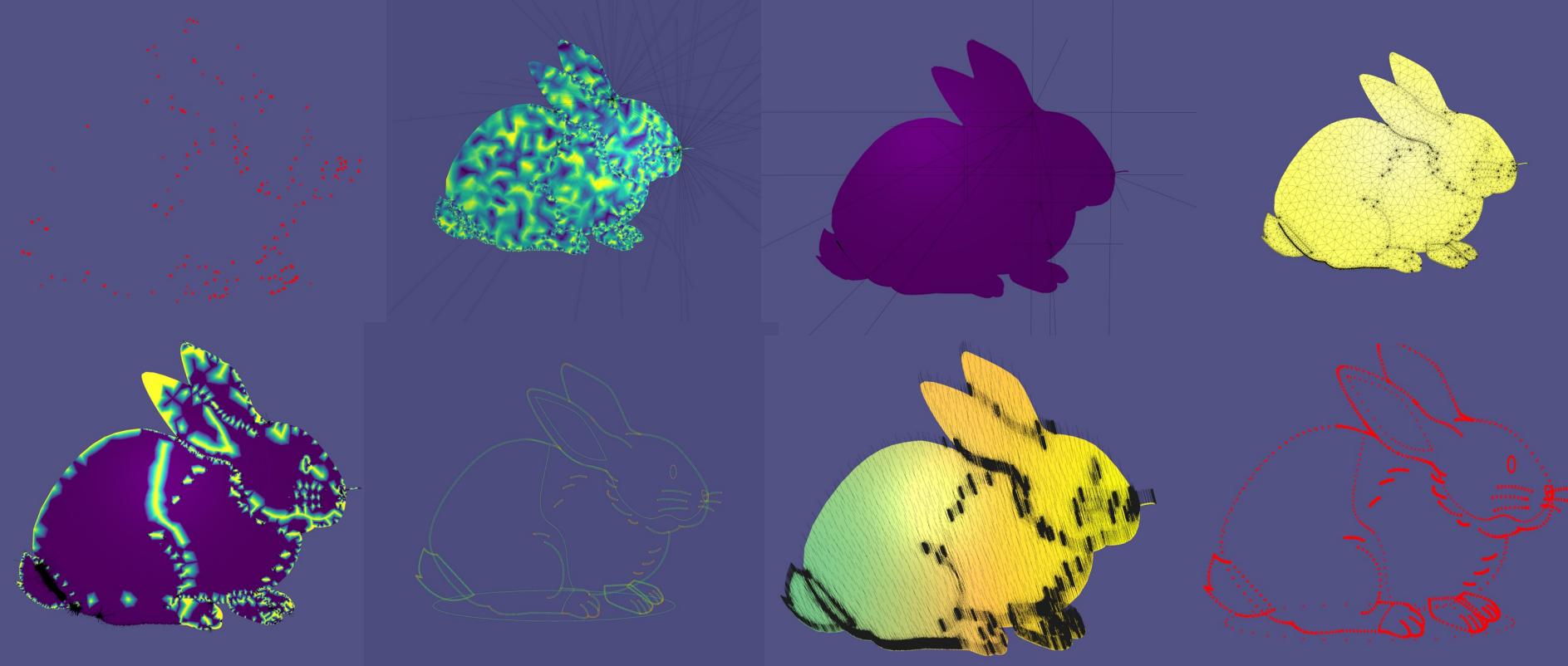


(b) after smoothing





Implementation

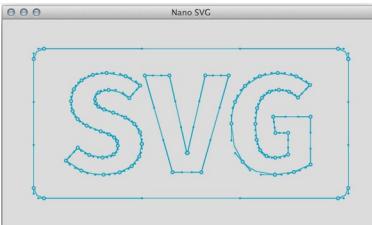


Rendering points on the screen

SVG files

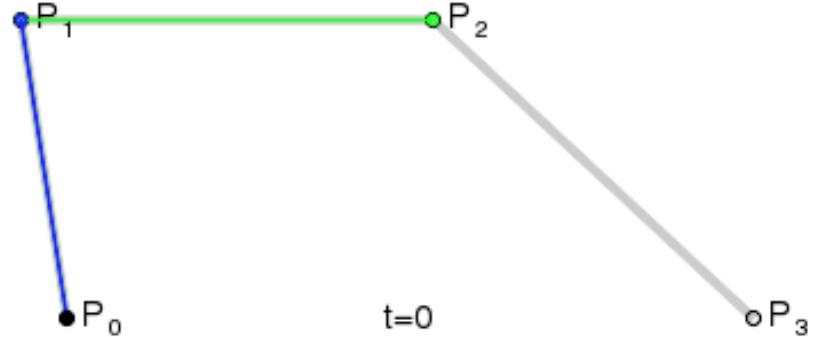
```
struct NSVGimage* image;
image = nsvgParseFromFile("../bunnyhd.svg", "px", 96);
std::vector<Eigen::MatrixXd> curves;
std::vector<Eigen::MatrixXi> edges;
for (auto shape = image->shapes; shape != NULL; shape = shape->next) {
    for (auto path = shape->paths; path != NULL; path = path->next) {
        for (auto i = 0; i < path->npts-1; i += 3) {
            {
                float* p = &path->pts[i*2];
                bezierToVertex(5, p, curves, edges, path->closed);
            }
        }
    }
}
```

```
for(double i=0;i<=dotsPerCurve;i++)
{
    double t = i/dotsPerCurve;
    double x = pow((1.0-t),3) * p[0] + 3.0 * t * pow((1.0-t),2) * p[2] + 3.0*pow(t,2)*(1.0-t) * p[4] + pow(t,3) * p[6];
    double y = pow((1.0-t),3) * p[1] + 3.0 * t * pow((1.0-t),2) * p[3] + 3.0*pow(t,2)*(1.0-t) * p[5] + pow(t,3) * p[7];
```



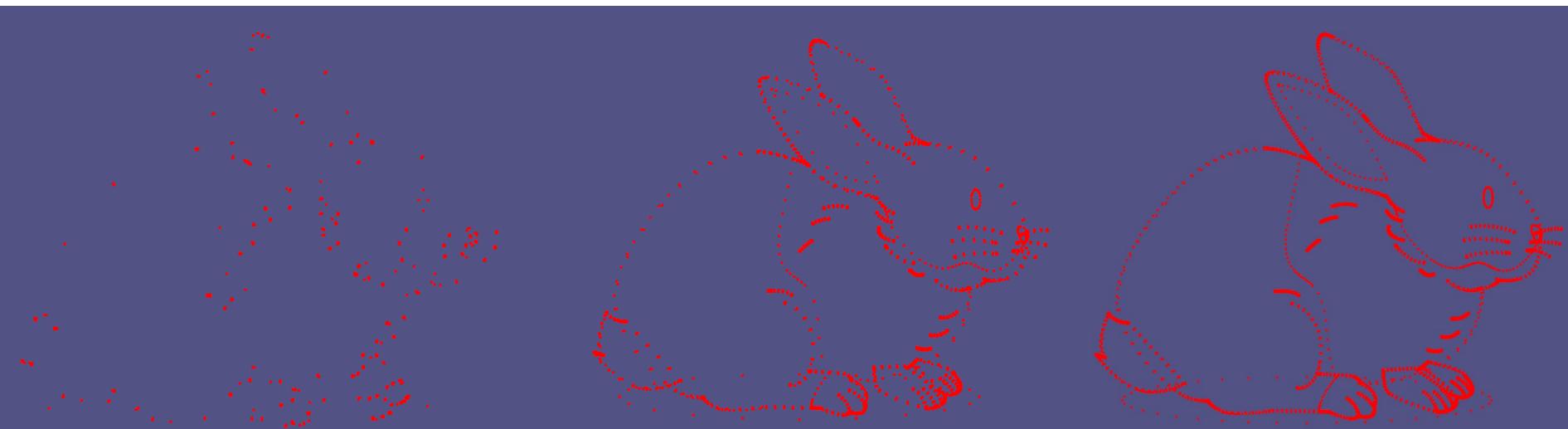
Tool : NanoSVG

Cubic Bezier Curves



$$\mathbf{B}(t) = (1-t)^3 \mathbf{P}_0 + 3(1-t)^2 t \mathbf{P}_1 + 3(1-t)t^2 \mathbf{P}_2 + t^3 \mathbf{P}_3, \quad 0 \leq t \leq 1.$$

Rendering points on the screen

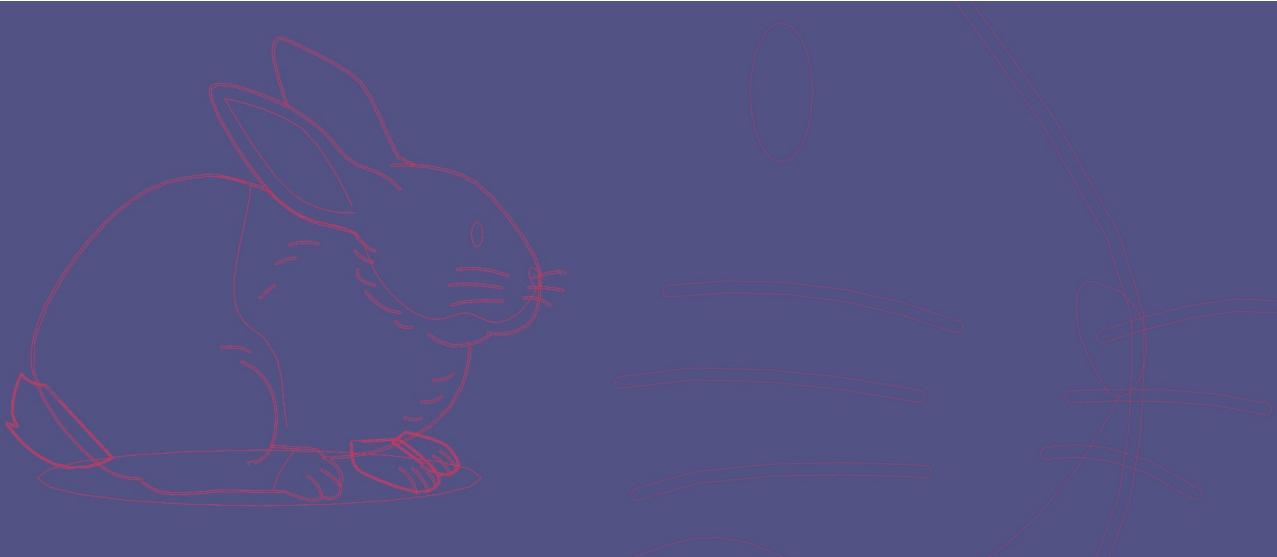


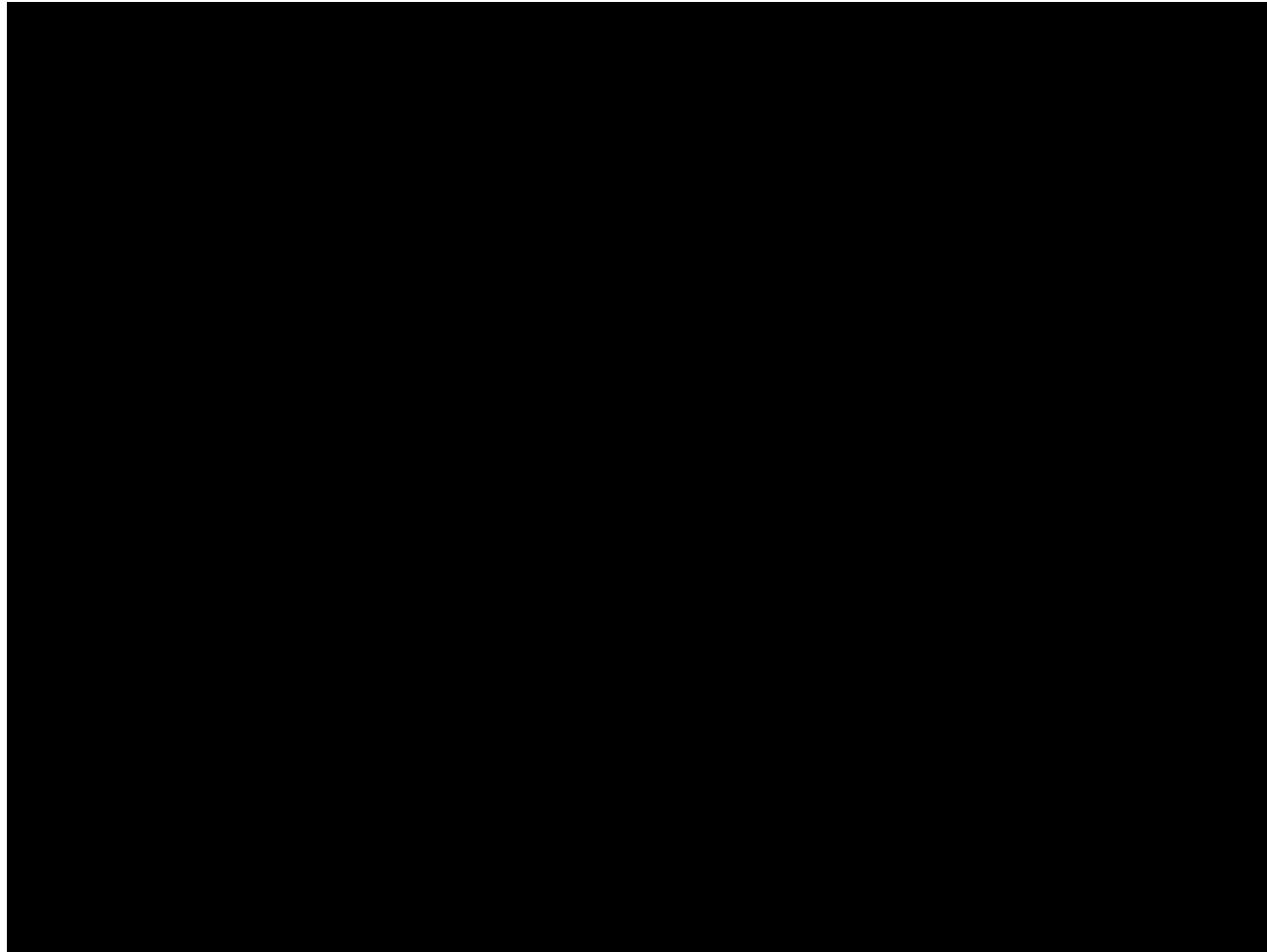
1 point per
curve

5 points per curve

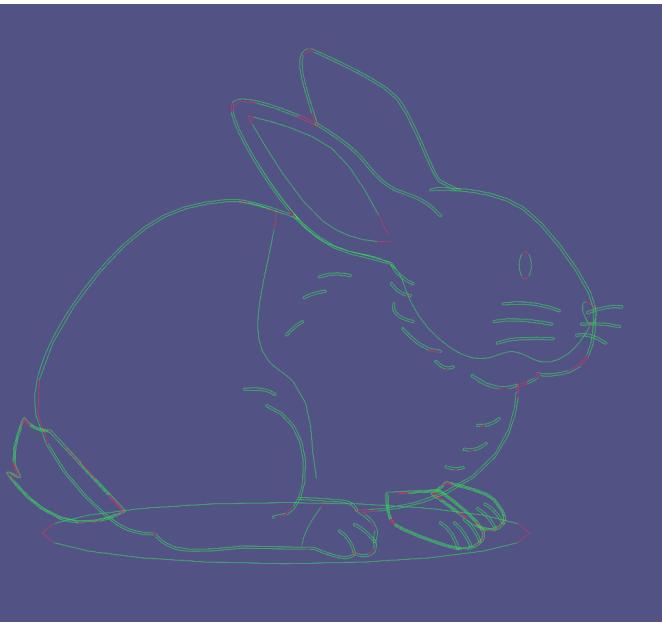
10 points per curve

Edges





Calculating edge weights



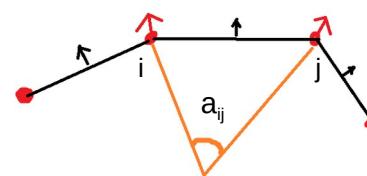
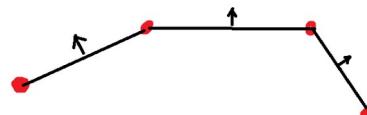
negative : red

positive : green

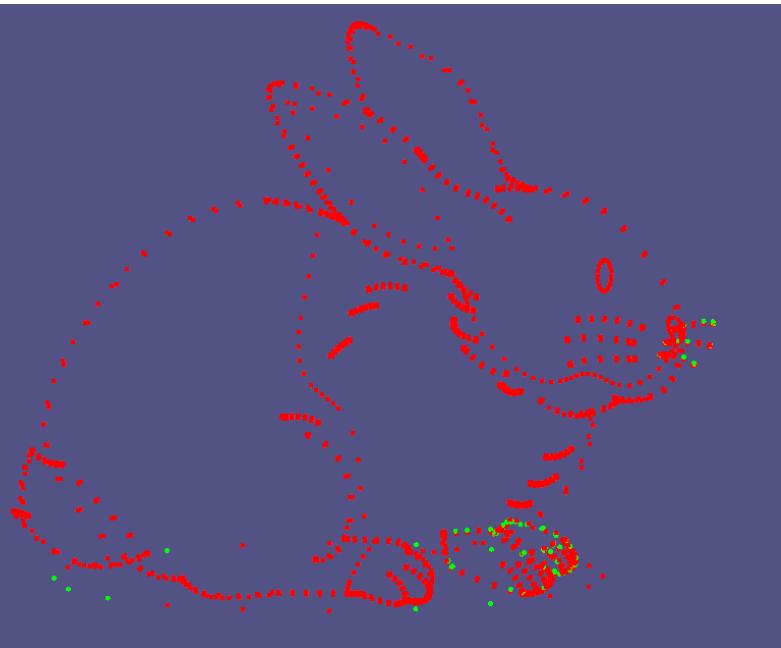
$$w_{ij} = F_a(a_{ij}) \cdot F_d(d_{ij})$$

$$F_a(a_{ij}) = \tanh\left(\frac{4}{\pi}\left(|\frac{\pi}{2} - a_{ij}| - \frac{\pi}{4}\right)\right)$$

$$F_d(d_{ij}) = e^{-\tilde{d}_{ij}^2/\sigma_d^2}$$



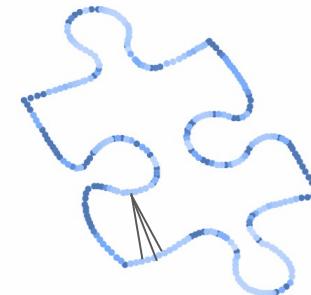
Calculating vertex attraction weights



$$\alpha_i = F_a^+(a_i) \cdot F_d^+(d_i)$$

$$F_a^+ = \max(0, F(a_i))$$

$$\sigma_d = 20$$



`igl::ray_box_intersect(V.row(i), Vnormal, box)`

Find local minimum

$$\min \sum_{ij} w_{ij}(l_i - l_j)^2 + \omega \sum_i \alpha_i(1 - l_i)$$

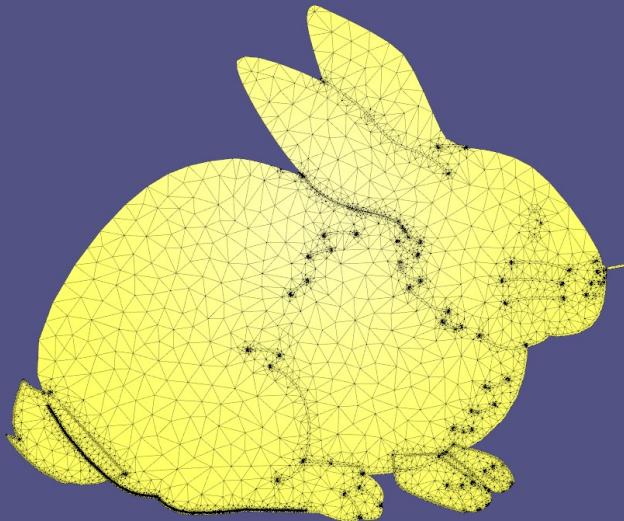
subj. to $0 \leq l_i \leq 1$

```
auto W = calculateEdgeWeights(V,E,VtoEnormals);
auto alpha = calculateVertexAttraction(V,VtoEnormals);
auto label = Eigen::VectorXd{V.rows()}.setConstant(0.5); //final labeling, starts at 0.5
double Wsum = 0.0;
//edge weight sum
for(int i=0;i<E.rows();i++)
{
    auto p1 = E.row(i).coeff(0,0);
    auto p2 = E.row(i).coeff(0,1);

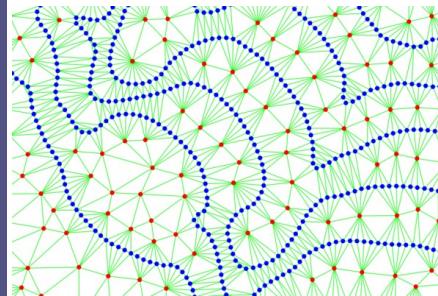
    Wsum += W[i] * pow(label[p1]-label[p2],2.0);
}
double alphasum = 0.0;
//vertex weight sum
for(int i=0;i<V.rows();i++)
{
    alphasum += alpha[i] * (1-label[i]);
}

//minimise here
auto total = Wsum + alphasum;
```

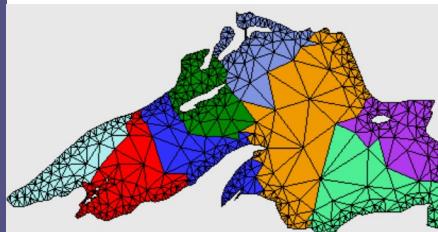
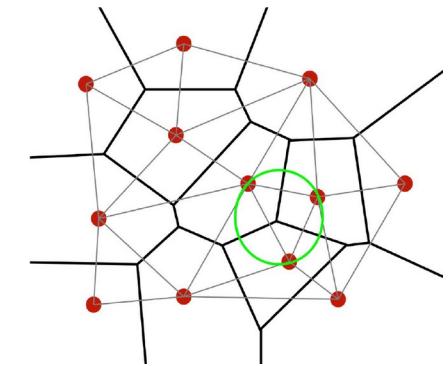
Triangulation



Generate Steiner points



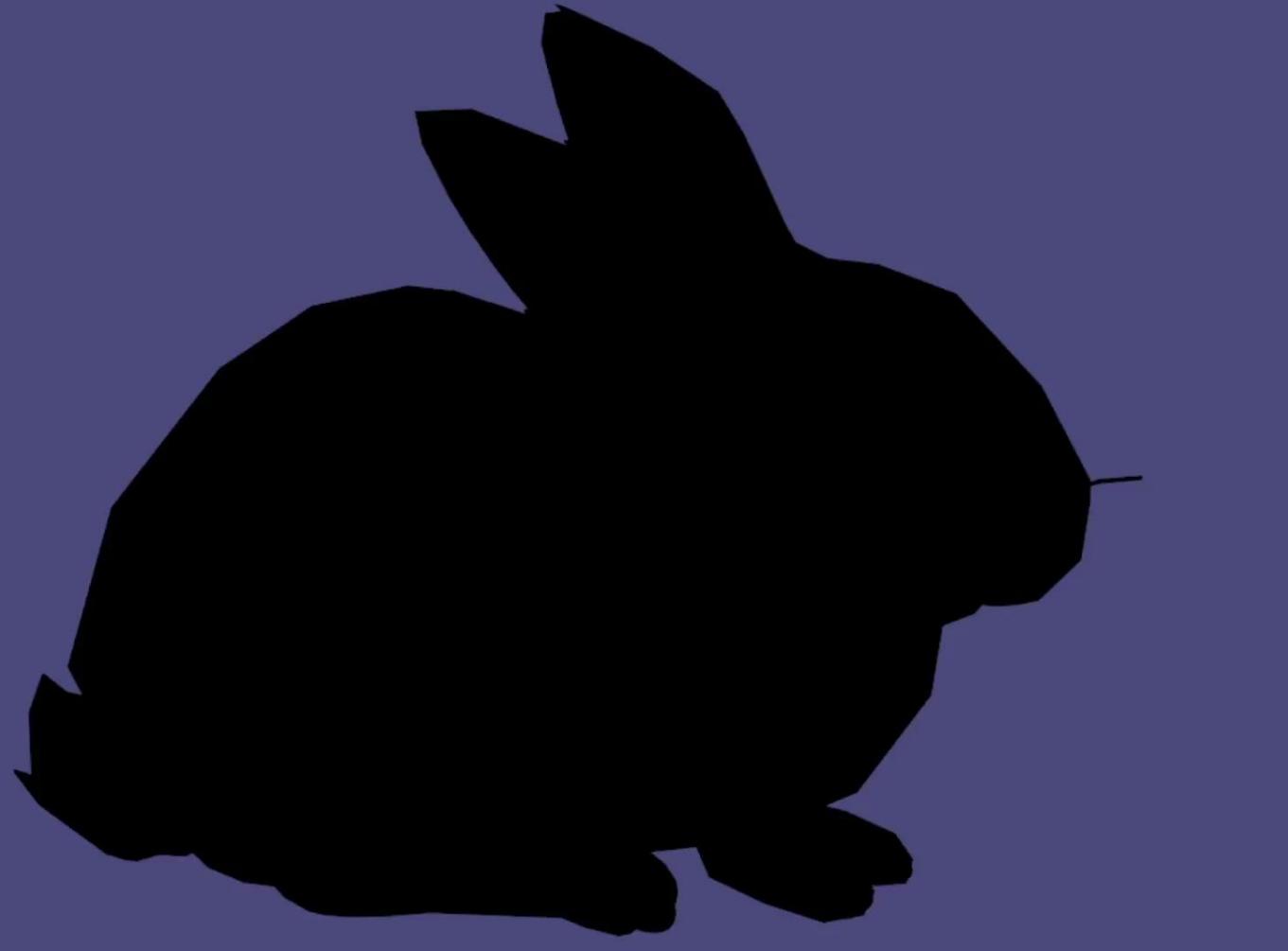
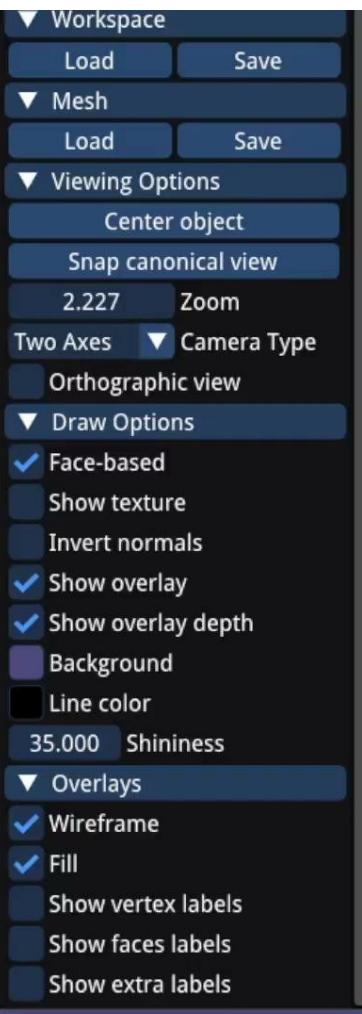
Perform Delaunay triangulation



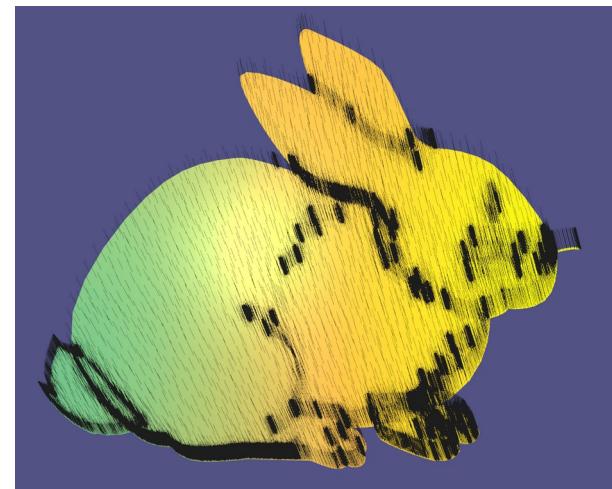
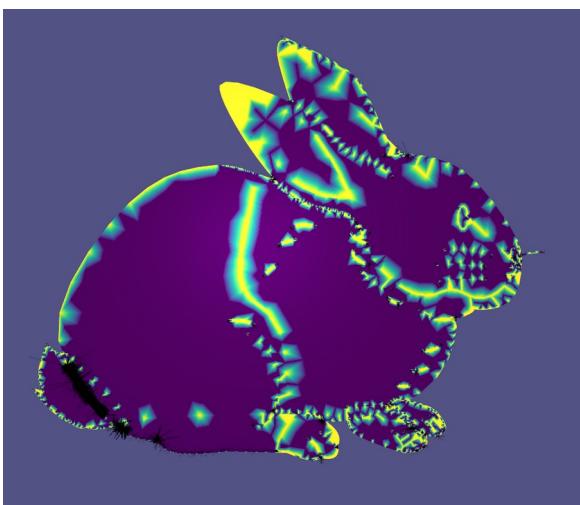
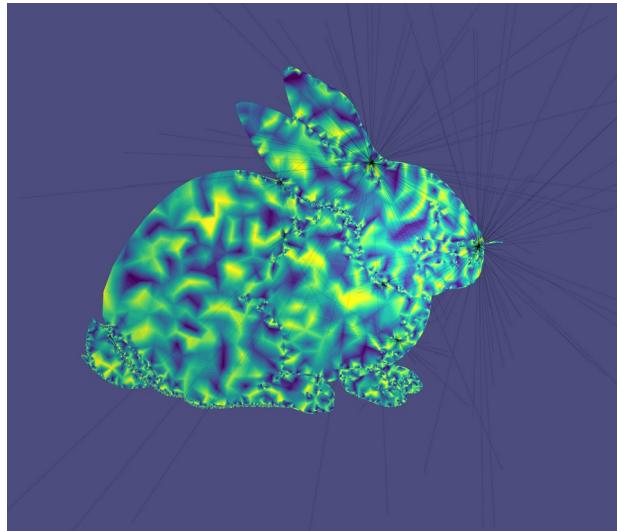
triangle

A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator.

Jonathan Richard Shewchuk
Computer Science Division
University of California at Berkeley
Berkeley, California 94720-1776
jrs@cs.berkeley.edu



Exploring vector fields with Libigl



Conclusion?

