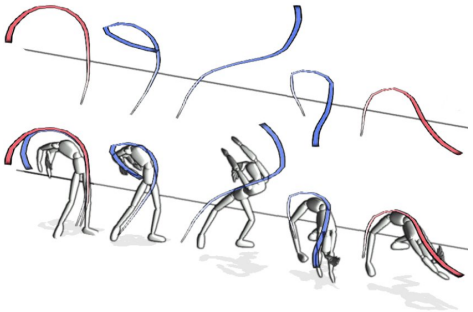
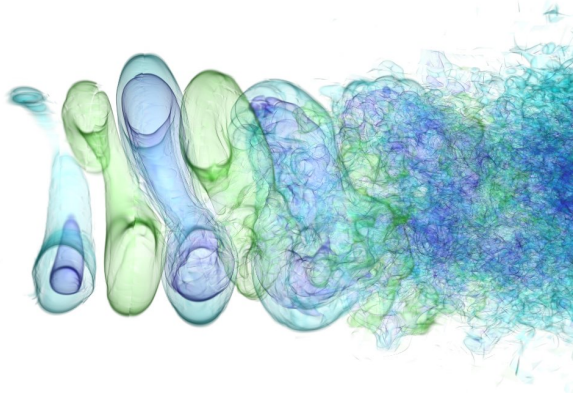
Computer Graphics & Scientific Visualization

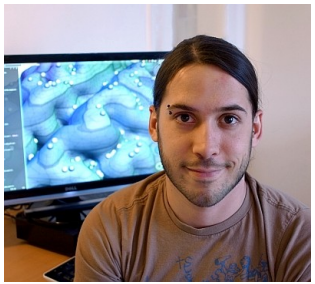
MPRI 2-39





Your teachers

Julien Tierny
julien.tierny@lip6.fr



CNRS,
Sorbonne Univ. UPMC

Scientific Visualization

Marie-Paule Cani
marie-paule.cani@polytechnique.edu

Ecole Polytechnique

3D Modeling, Computer Animation

Damien Rohmer
damien.rohmer@polytechnique.edu

Ecole Polytechnique

3D Modeling, Computer Animation

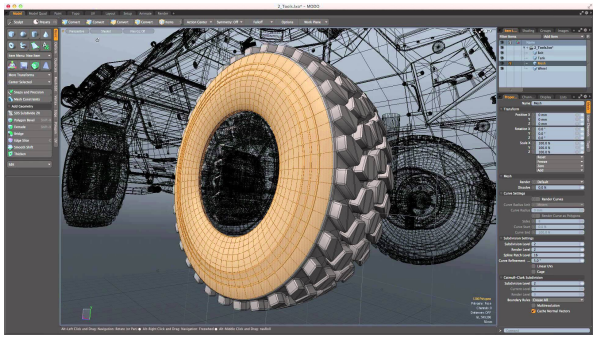
Your background

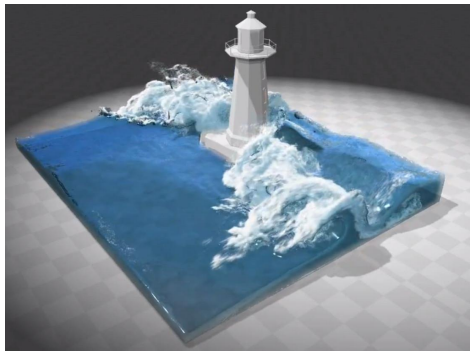
Previous experience in

- Programmation?
- Computational geometry (Delaunay, Voronoi) ?
- Computer graphics (rendering, parametric curves) ?

Project involving programmation or theoretical study of research article?

General introduction to Computer Graphics (CG)



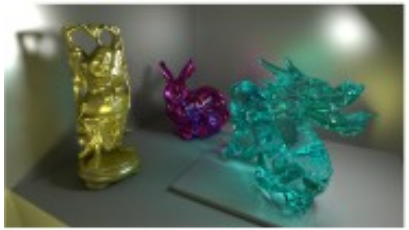


Sub-fields of CG

Modeling

Animation

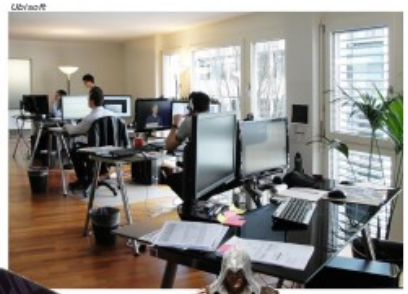
Rendering



Some applications related to CG

Entertainment: Video game

Ubisoft



Ubisoft

Naughty Dog

Nvidia

Ubisoft

The Crew

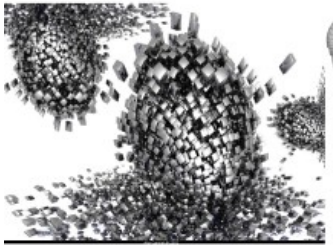
http://eelgod.deviantart.com/

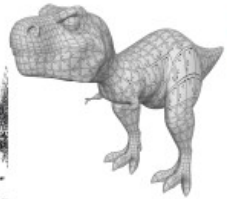
Entertainment: Cinema, VFX

Autodesk Maya

Weta

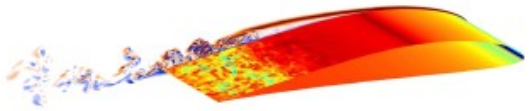
Matrix

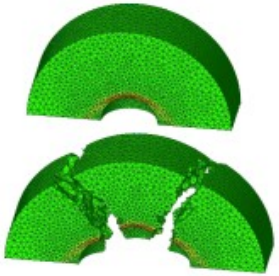


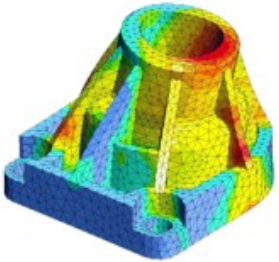


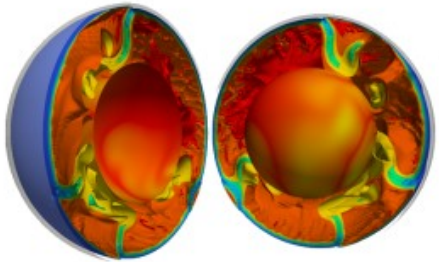
PTex

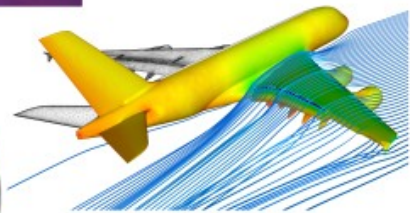
Scientific computing



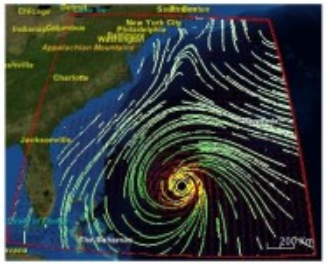


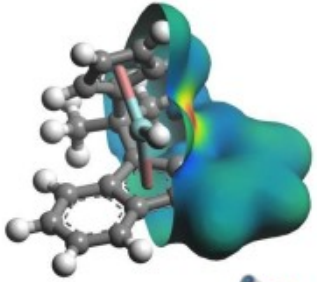


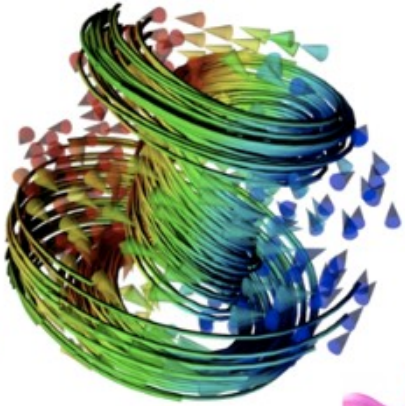


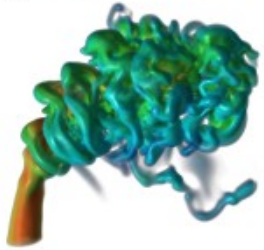


Visualization

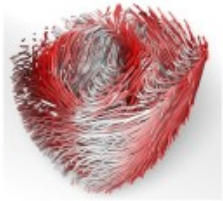


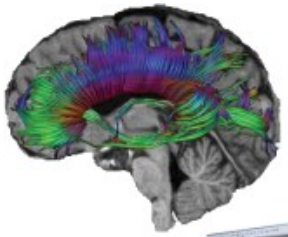


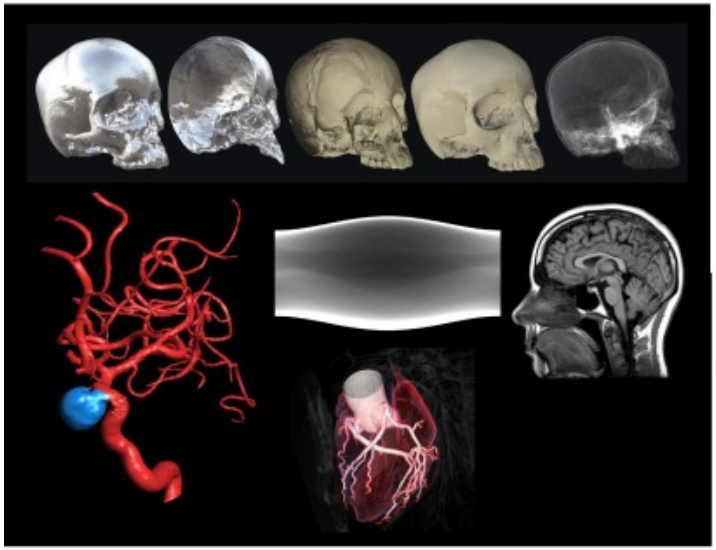


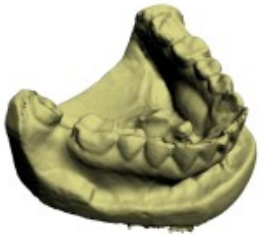


Medical imaging

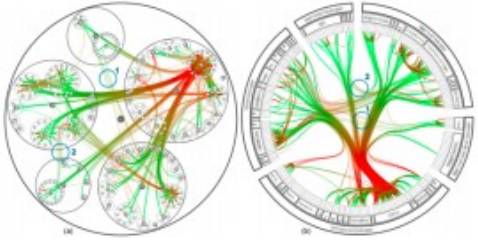


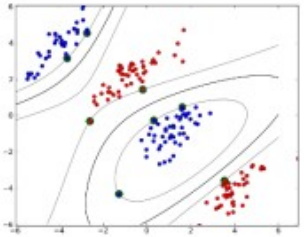


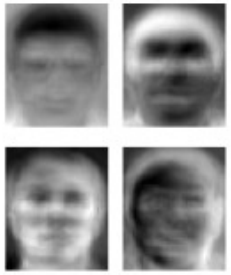


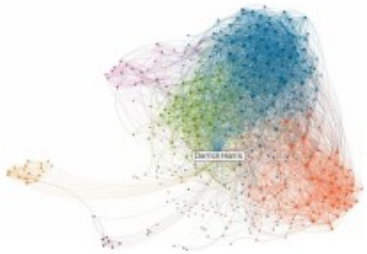


Data analysis

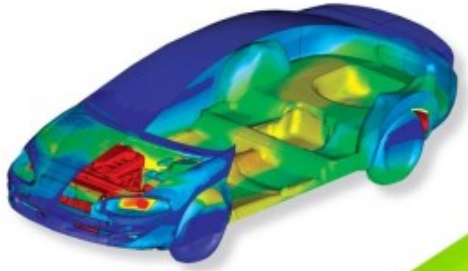


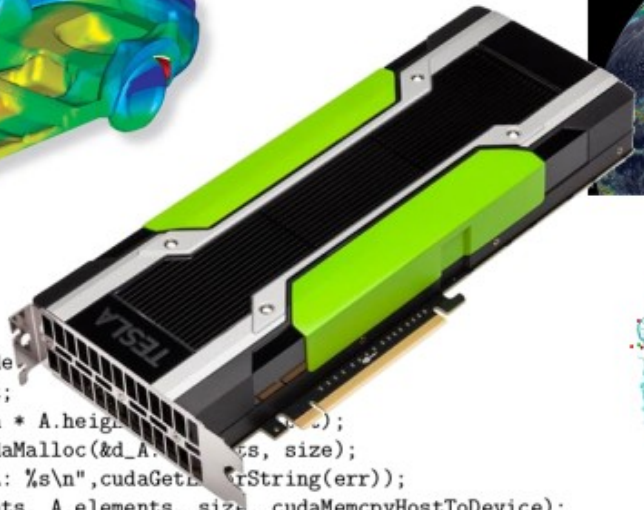


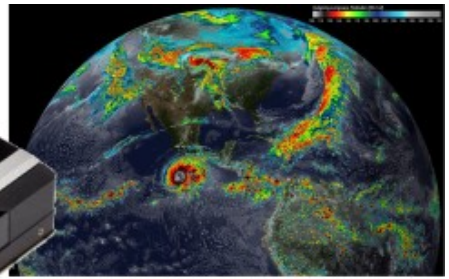


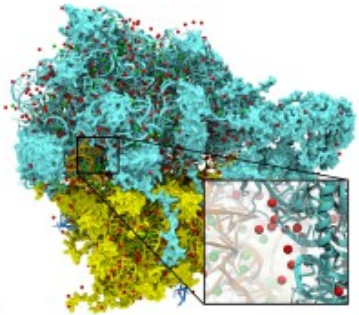


HPC

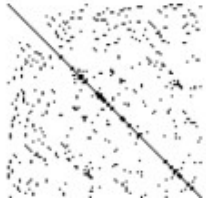




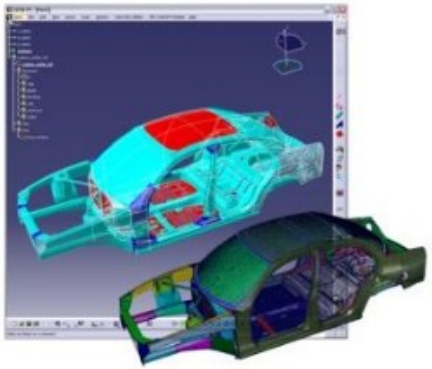


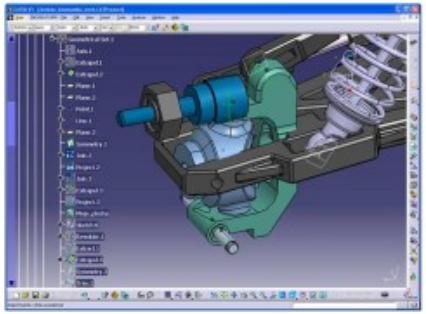


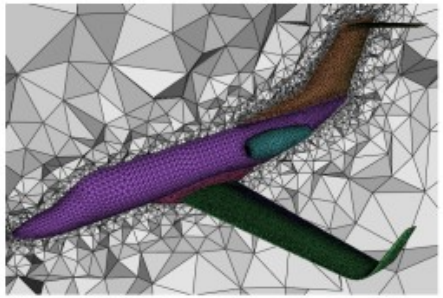
```
Matrix d_A;
d_A.width = d_A.stride
d_A.height = A.height;
size_t size = A.width * A.heig
cudaError_t err = cudaMalloc(&d_A.
printf("CUDA malloc A: %s\n",cudaGet
cudaMemcpy(d_A.elements, A.elements, size, cudaMemcpyHostToDevice);

Matrix d_B;
d_B.width = d_B.stride = B.width;
d_B.height = B.height;
size = B.width * B.height * sizeof(float);
err = cudaMalloc(&d_B.elements, size);
printf("CUDA malloc B: %s\n",cudaGetErrorString(err));
```



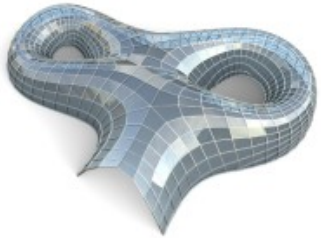
Design, CAD



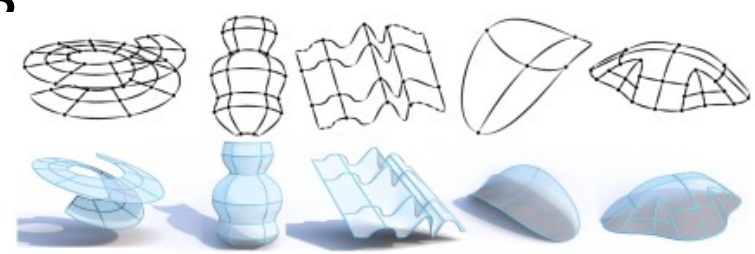


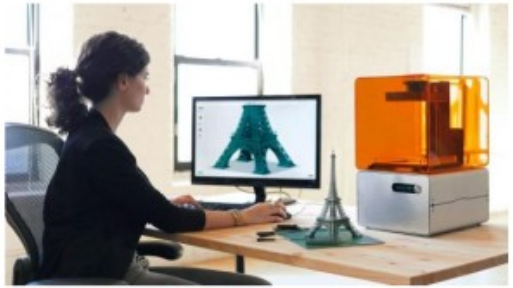


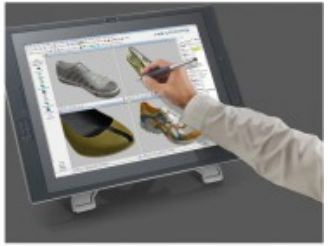
Design: Architecture



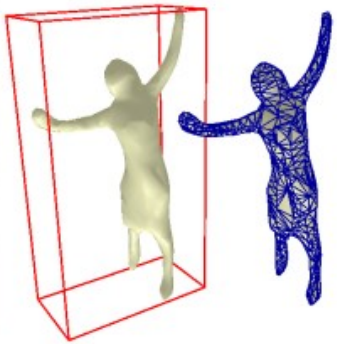
Design: Prototyping







3D reconstruction



Cultural heritage



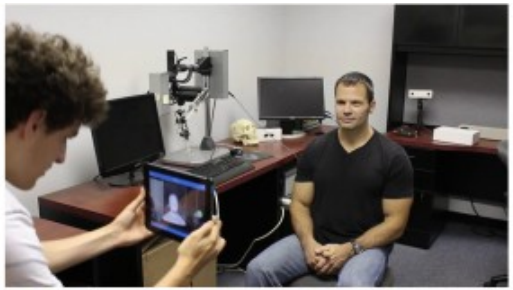
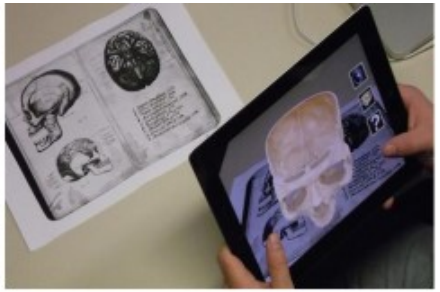







Virtual reality

Augmented reality




What we are going to see in details in this module

The computational methods to **model**, **animate** and **visualize** 3D virtual contents.

- Basic concepts
- State of the art research methods

Note: CG is a recent topic
(started ~60 for CAD, rapid extension after 2000 with video games, animation cinema, VFX, now 3D printing)

Some current challenges:
Expressive modeling, animation authoring, complex & uncertain data visualization.

Sequencing of the module

1st period

- 13/09: Introduction + Modeling with meshes (D. Rohmer)
- 20/09: Mesh representations (J. Tierny)
- 27/09: Scalar field visualization (J. Tierny)
- 04/10: Modeling with scalar fields (M.-P. Cani)
- 11/10: Procedural vs. Interactive modeling (M.-P. Cani)
- 18/10: Expressive design: Sculpting and sketching 3D shapes (M.-P. Cani)
- 08/11: Descriptive animation technique (D. Rohmer)
- 15/11: Research seminar (you present a research article)

Sequencing of the module

2nd period

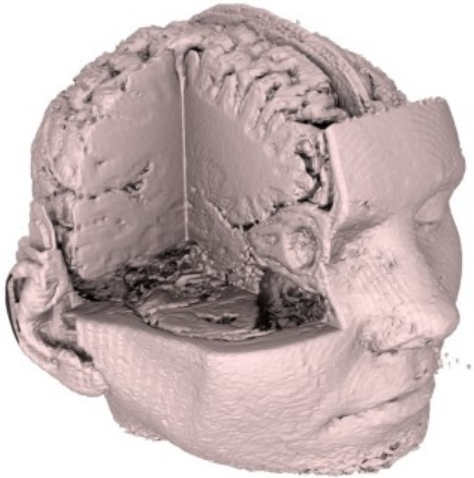
- 06/12 Physically based animation (D. Rohmer)
- 13/12 Layered models for natural phenomena (M.-P. Cani)
- 20/12 Vector field visualization (J. Tierny)
- 10/01 Character animation 1: Motion synthesis and control (M.-P. Cani)
- 17/01 Character animation 2: Skin, clothes and hair (D. Rohmer)
- 24/01 Tensor field visualization (J. Tierny)
- 31/01 Advanced techniques in visualization (J. Tierny)
- 14/02 Project seminar (you present your work)

3D Content Representation & Modeling

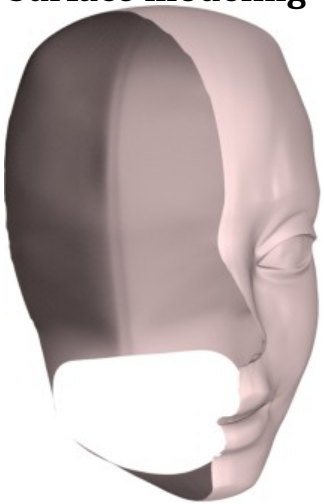
Different representations of 3D virtual content
Their common use case

Modeling 3D objects for Graphics application

Volume modeling



Surface modeling



- Fast GPU rendering, low memory footprint
- Focus on visible part

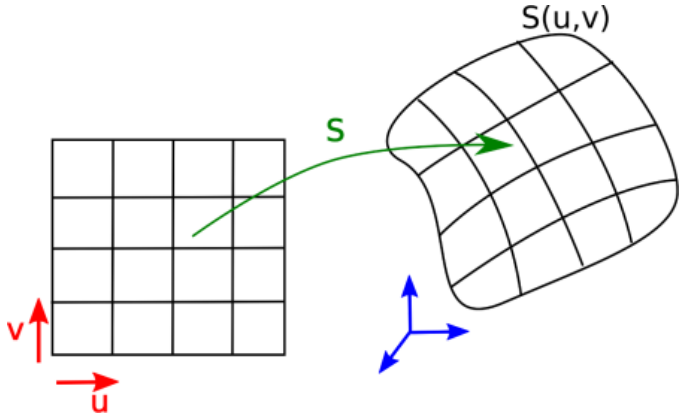
=> focus on modeling surfaces in CG

Note: Objective of Graphics applications != CAD/medical simulation
Note: We will learn how to visualize acquired volume data

Two modeling approaches: Explicit / Implicit

Explicit representation

$(x, y, z) = S(u, v)$



+ *Neighborhood information*

Implicit representation

$\{(x, y, z) \mid F(x, y, z) = 0\}$

[Sherstyuk, 98-88]

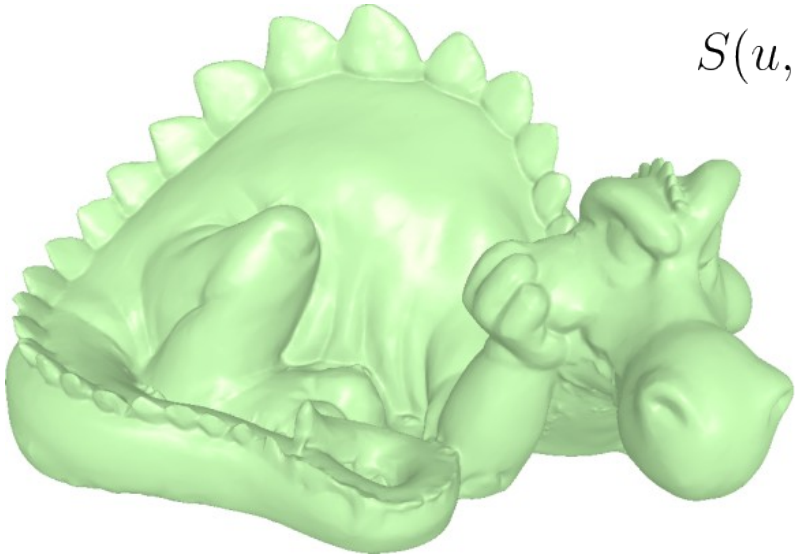
+ *Topological modification*

Today: The explicit representation

(but 27/09+04/10: visualizing and modeling field functions)

Problematic of surface representation

What is the function of this dragon ?



$$S(u, v) = ?$$

Explicit surface representation

I. Meshes

II. Parametric surfaces

III. Subdivision surfaces

IV. Point cloud

Mesh

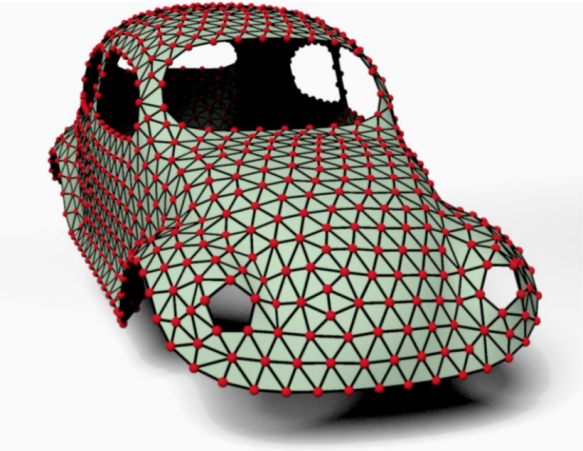
Simplest possible representation: set of triangles

$$S_i : \begin{cases} \mathcal{D} \subset \mathbb{R}^2 & \to \mathbb{R}^3 \\ (u, v) & \mapsto S_i(u, v) = u\overrightarrow{AB} + v\overrightarrow{AC} + \overrightarrow{OA} \end{cases}$$

$$\mathcal{D} : (u, v) \in [0, 1]^2, 0 \leq u + v \leq 1$$

$$S = \bigcup_i S_i$$

Union of linear functions



Vertex

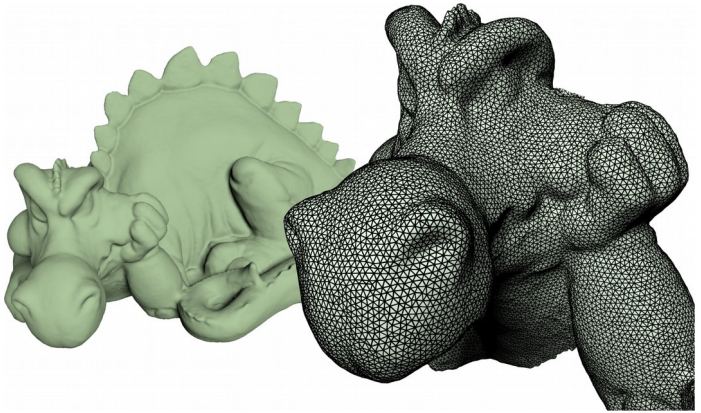
$$\mathcal{V} = (v_1, \ldots, v_N)$$

Edge

$$\mathcal{E} = (e_1, \ldots, e_{N_e}) \in (\mathcal{V}^2)^{N_e}$$

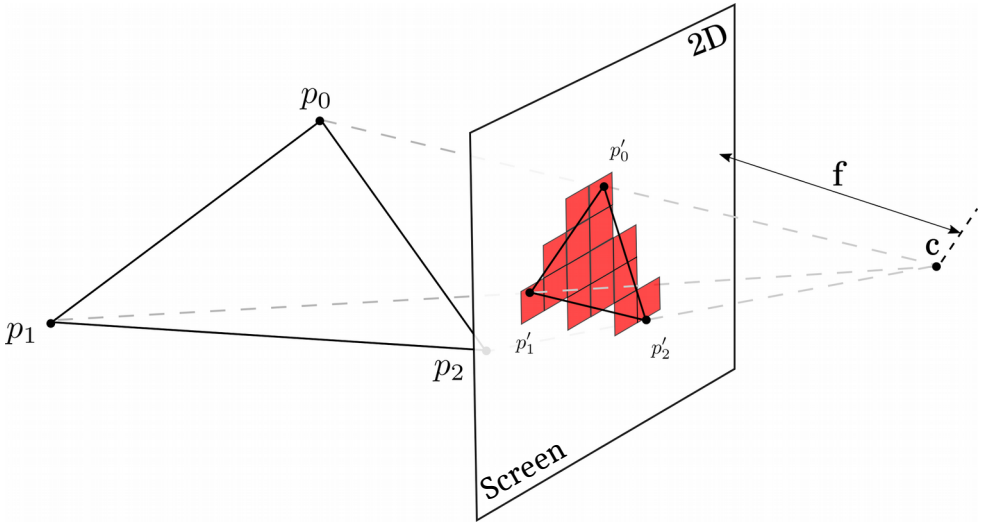
Face

$$\mathcal{F} = (f_1, \ldots, f_{N_f}) \in (\mathcal{V}^3)^{N_f}$$

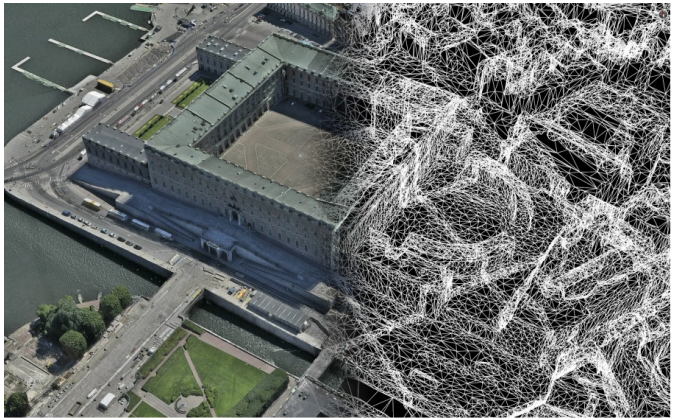


Triangular meshes = Common base representation

Triangle is the only surface that GPU is able to render natively



=> Need to convert any representation to triangle meshes for final rendering

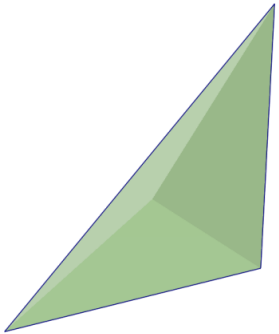


$$p' = \mathrm{P}\ p$$

P: projection matrix

Mesh encoding

ex. Tetrahedron



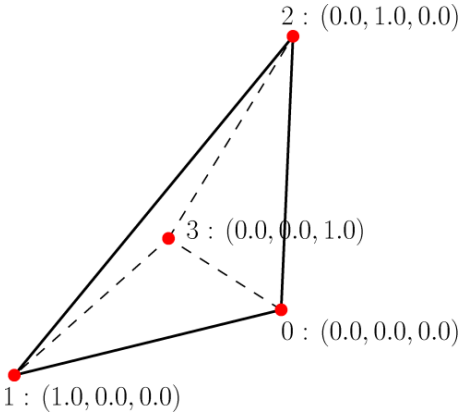
1st solution: “Polygon soup”

[(0,0,0), (1,0,0), (0,0,1),
(0,0,0), (0,0,1), (0,1,0),
(0,0,0), (0,1,0), (1,0,0),
(0,1,0), (0,0,1), (1,0,0)]

2nd solution: geometry/connectivity

Geometry: [(0,0,0), (1,0,0), (0,1,0), (0,0,1)]
Connectivity: [(0,1,3), (0,3,2), (0,2,1), (1,2,3)]

Standard (exchange format, GPU compatible)



Note next week - More advanced mesh encoding data structure (neighborhood query)

Example of ASCII mesh file format

file.off

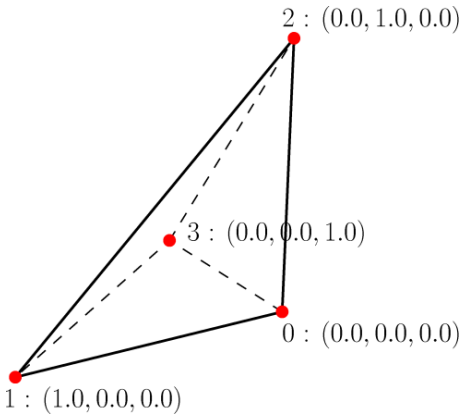
```
OFF
4 4 6
0 0 0
1 0 0
0 1 0
0 0 1
0 1 3
0 3 2
0 2 1
1 2 3
```

OFF
Nvertex Nface Nedge(0)
all vertices coords
all triangle index

file.obj

```
v 0 0 0
v 1 0 0
v 0 1 0
v 0 0 1
f 1 2 4
f 1 4 3
f 1 3 2
f 2 3 4
```

v vertex coords
f triangle index (1-indexed)



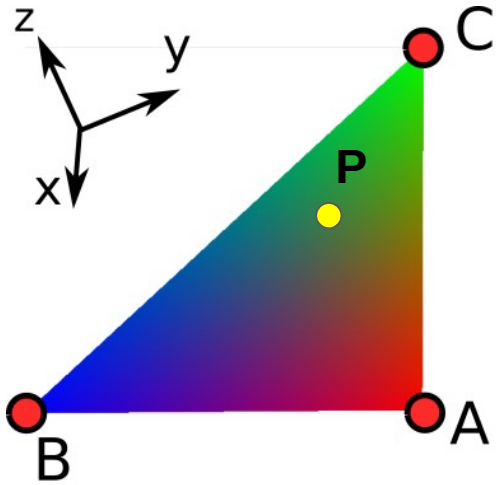
Vertex parameter

Each vertex can be associated with some parameters (normal, color, texture coordinates, etc)

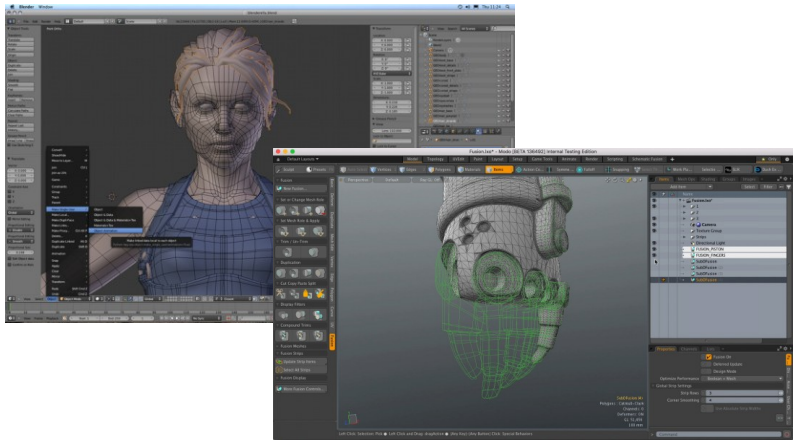
The values are continuously spread over the surface of the triangle using barycentric/linear interpolation

$$v_P = \alpha\, v_A + \beta\, v_B + \gamma\, v_C$$

(α, β, γ) are barycentric coordinates



Use of mesh representations



Pros:

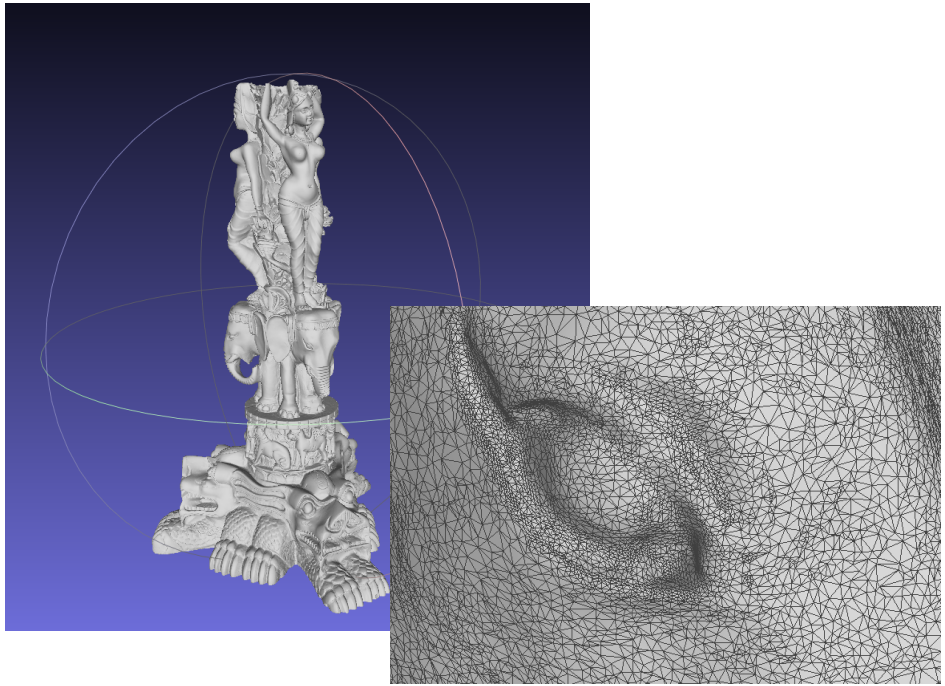
- The simplest representation
- The most general (can approximate any surface)
- The only one handled by GPU (fast rendering)

Cons:

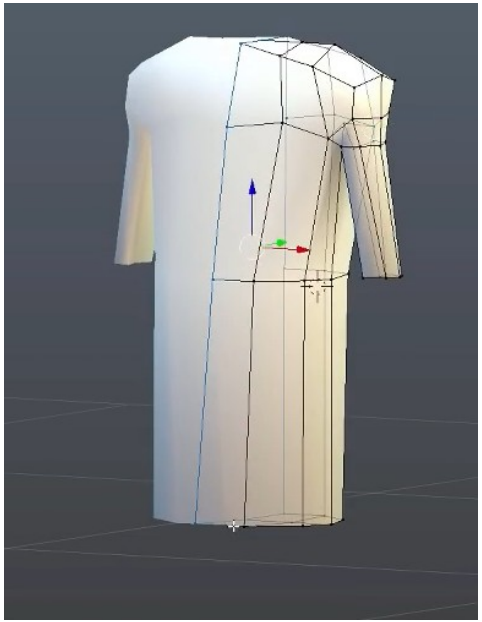
- The worst approximation for smooth surface (C^0 only)
  => High number of samples => ***complex modeling***

Note: meshes can be made of polygons. Artists generated meshes are mostly quadrangle meshes

Example of mesh modeling



Low res polygonal modeling



Explicit surface representation

I. Meshes

II. Parametric surfaces

III. Subdivision surfaces

IV. Point cloud

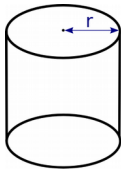
Parametric representation

Goal: to allow higher order elements

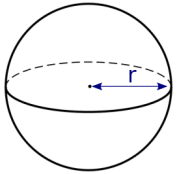
=> Smoother representation, less samples (easier manipulation)

We know the equation for basic primitives (cylinder, sphere, ellipsoids/quadrics, torus, cones, ...)

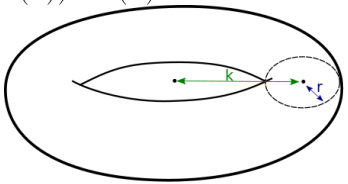
ex. $\begin{vmatrix} x(u,v) = r\ \cos(u) \\ y(u,v) = r\ \sin(u) \\ z(u,v) = v \end{vmatrix}$



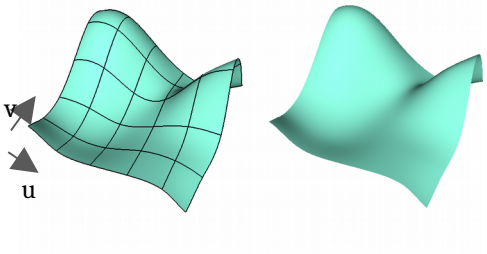
$\begin{vmatrix} x(u,v) = r\ \sin(u)\cos(v) \\ y(u,v) = r\ \sin(u)\sin(v) \\ z(u,v) = r\ \cos(u) \end{vmatrix}$

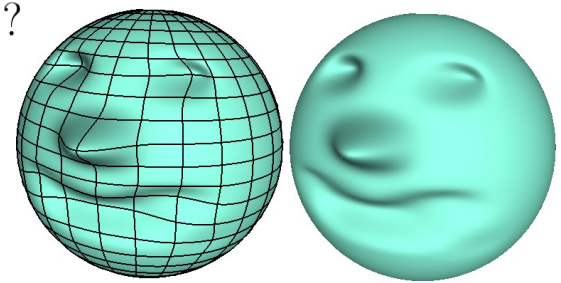


$\begin{vmatrix} x(u,v) = (k + r\cos(v))\ \cos(u) \\ y(u,v) = (k + r\cos(v))\ \sin(u) \\ z(u,v) = r\sin(v) \end{vmatrix}$



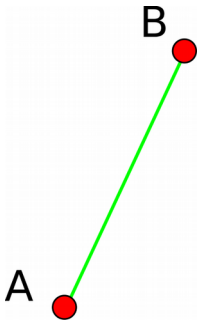
What about more general surfaces? $S(u,v) = ?$





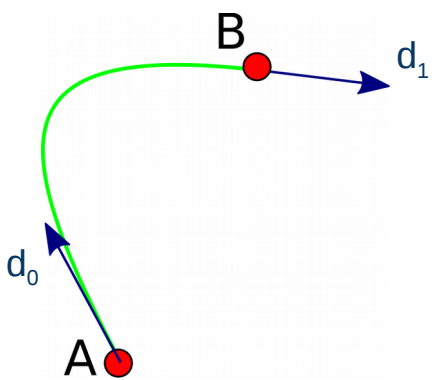
Parametric representation for curves

Straight segment



$$c(t) = A\,(1-t) + B\,t$$

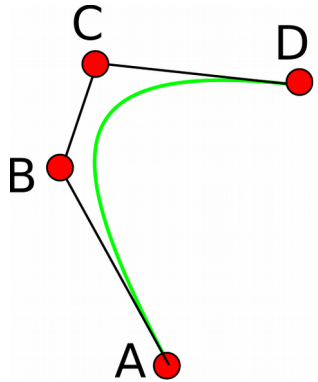
Cubic curve



[Hermite] *2 extremities+tangent*

$$c(t) = \alpha\,t^3 + \beta\,t^2 + \gamma\,t + \delta$$

$$\begin{cases} c(0) = A \\ c(1) = B \\ c'(0) = d_0 \\ c'(1) = d_1 \end{cases}$$



[Bezier] *Other representation using a control polygon (A,B,C,D)*

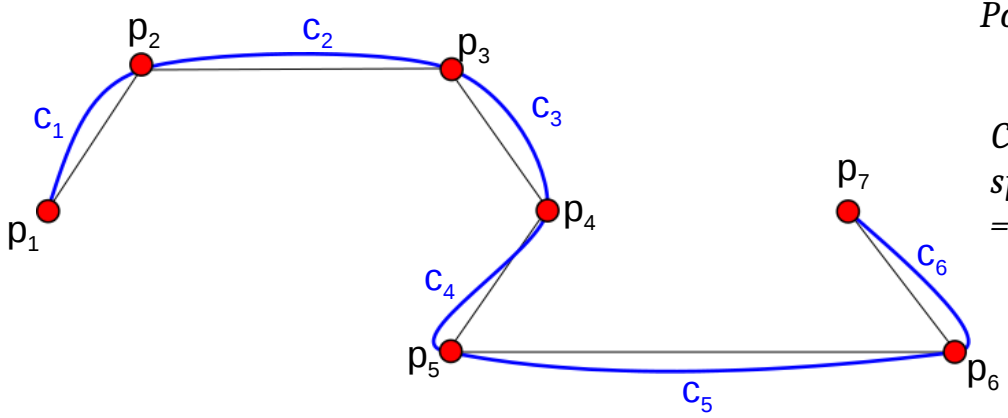
$$c(t) = (1-t)^3\,A + 3(1-t)^2 t\,B + 3(1-t)t^2\,C + t^3\,D$$

Note: Curve passes exactly by A, D. But only approximates B, C.

Parametric representation for curves

Create curves between multiples points

Union of cubics between segments



Rem: Settings all derivatives of Hermite curves is possible but tedious (in 3D!). Points are prefered for interactive purpose

Common parameterization: Catmull-Rom splines
=> compute automatically the derivatives

$$
\begin{cases}
C_i(0) = P_i \\
C_i(1) = P_{i+1} \\
C'_i(0) = k\,(P_{i+1} - P_{i-1}) \\
C'_i(1) = k\,(P_{i+2} - P_i)
\end{cases}
$$

k: tension parameter

Limitations:

- Overshoot beyond the sample points (outside the convex hull of the control polygon)
- Only globally C^1 (C^2 is prefered)

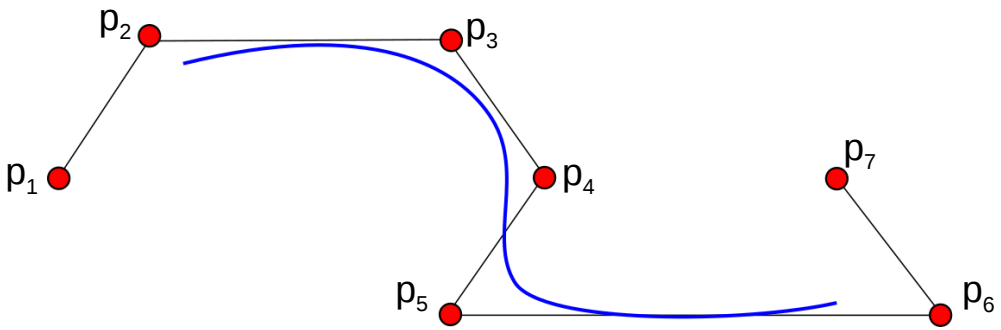
Parametric representation for curves

B-spline (basis spline) $$c(t) = \sum_i b_i(t)\, P_i$$

Basis functions: smooth polynomial, minimal support

Special case: **Cubic Uniform B-spline**

$$\begin{cases} b_0(t) = (1-t)^3/6 \\ b_1(t) = (3t^3 - 6t^2 + 4)/6 \\ b_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6 \\ b_3(t) = t^3/6 \end{cases}$$

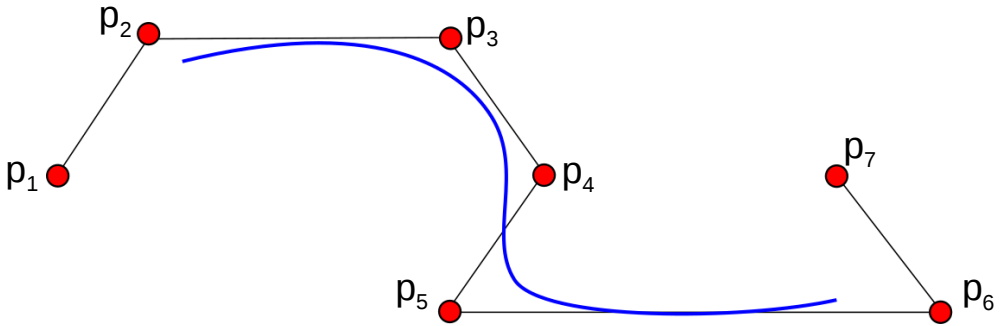


- Globally C^2
- Remains inside convex hull of control polygon
- Approximation only

Parametric representation for curves

Cubic uniform B-spline

$$\begin{cases} b_0(t) = (1-t)^3/6 \\ b_1(t) = (3t^3 - 6t^2 + 4)/6 \\ b_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6 \\ b_3(t) = t^3/6 \end{cases}$$



Matrix formulation

$$c(t) = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

***M**: fully defined by the basis function*

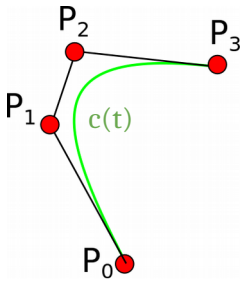
Parametric representation for **surface**

Curve -> Surface using *tensor product*

Curve

$$c(t) = \sum_i b_i(t)\, P_i$$

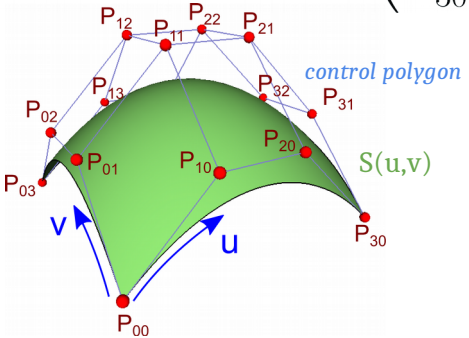
$$c(t) = (t^3\, t^2\, t\, 1) M\, (P_0\, P_1\, P_2\, P_3)^t$$



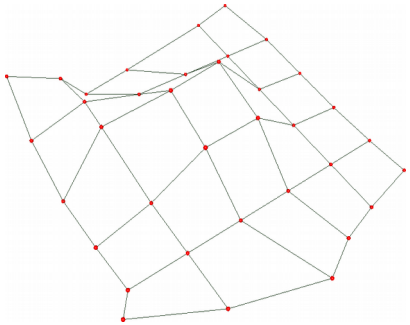
Surface

$$S(u,v) = \sum_i \sum_j b_i(u)\, b_j(v)\, P_{ij}$$

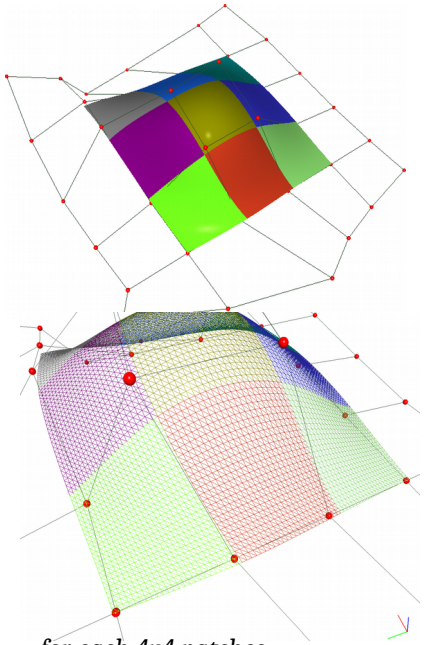
$$S(u,v) = (u^3\, u^2\, u\, 1) M \begin{pmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{pmatrix} M^t (v^3\, v^2\, v\, 1)^t$$



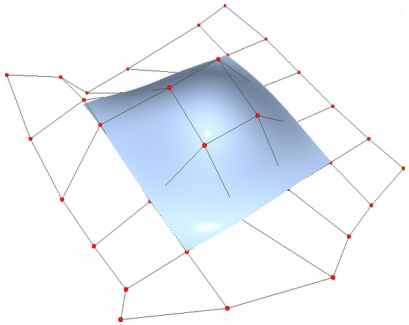
Practical generation of parametric surface

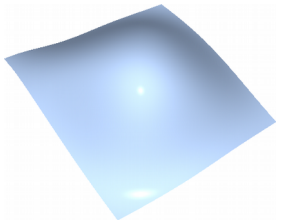


set 3D coordinates of
control polygon
(manual/function based step)



for each 4x4 patches
compute Nu x Nv sample points S(u,v)
build a quadrangular grid mesh





resulting smooth surface

Parametric surfaces: NURBS

NURBS (Non Uniform Rational BSpline)

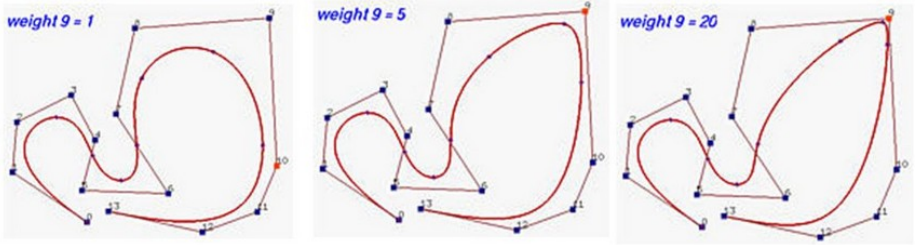
- Varying “parametric distance” between points
- Allow to represent exactly circles (2D), spheres/cylinders (3D)

=> NURBS = Base of freeform CAD modeling surfaces

$$S(u,v) = \sum_i \sum_j R_{i,j}(u,v) P_{ij}$$

$$R_{i,j}(u,v) = \frac{\omega_{ij}\, b_i(u)\, b_j(v)}{\sum_{m,n} \omega_{m,n}\, b_m(u)\, b_n(v)}$$

rational part



Parametric surfaces

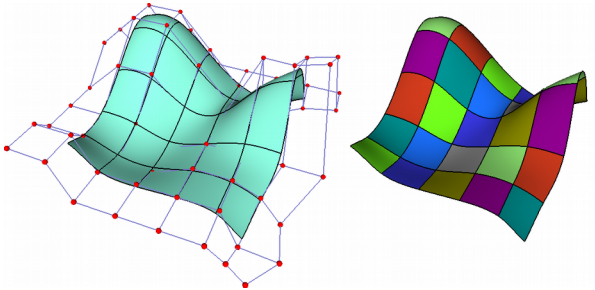
Pros:

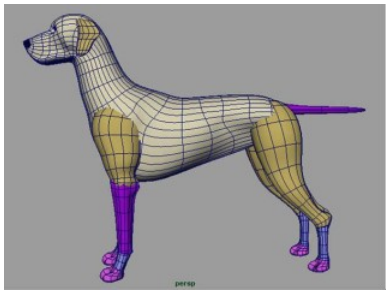
- Smooth (C2 for approximation, C1 for interpolation), local control.
- Derivative computation (curvature)

Highly used in CAD manufacturing

Cons:

- 4x4 grid based patch topology





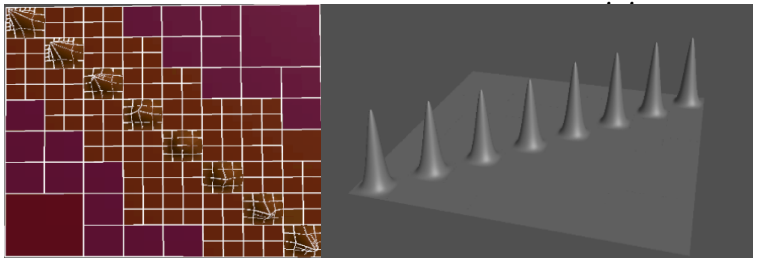
How to locally refine? add a branch ?

Several shapes will not smoothly

Extensions

Hierarchical spline
[Forsey, Bartels, SIGGRAPH 88]

T-splines
[Sederberg, Zheng, SIGGRAPH 03]



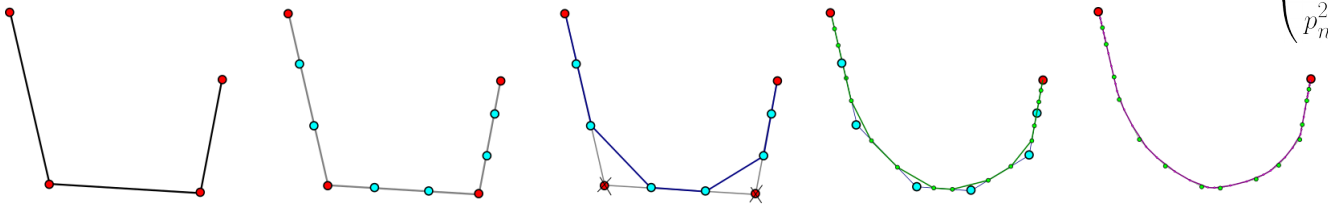
Explicit surface representation

I. Meshes
II. Parametric surfaces
III. Subdivision surfaces
IV. Point cloud

Subdivision surfaces

Idea - Consider arbitrary mesh as a control polygon. Compute a smooth surface interpolating/approximating it.
Objectives - Smooth, local control, arbitrary topology

Subdivision for curves



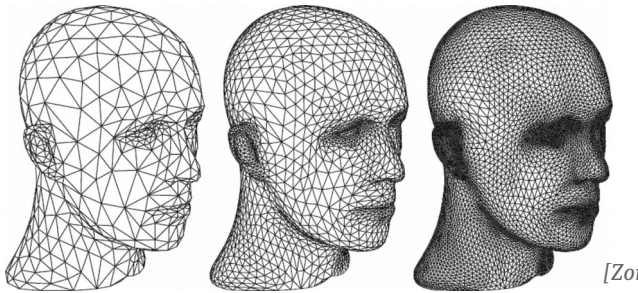
$$\begin{pmatrix} p_{n+1}^{2k} \\ p_{n+1}^{2k+1} \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} p_n^k \\ p_n^{k+1} \end{pmatrix}$$

...

Converge toward a C^2 curve
(Corner cutting)

Subdivision for surfaces

2D subdivision mask



[Zorin, Schroeder, SIGGRAPH Course Notes 99]

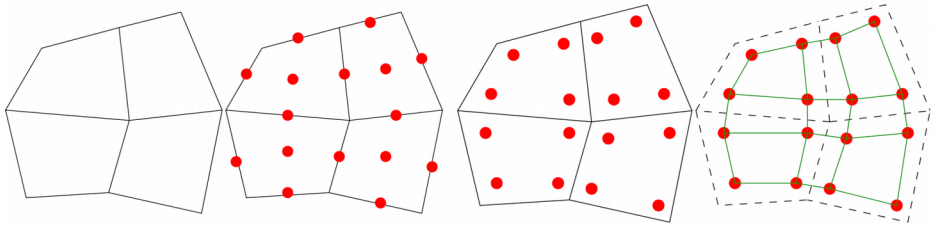
Example of subdivision surface: Doo-Sabin

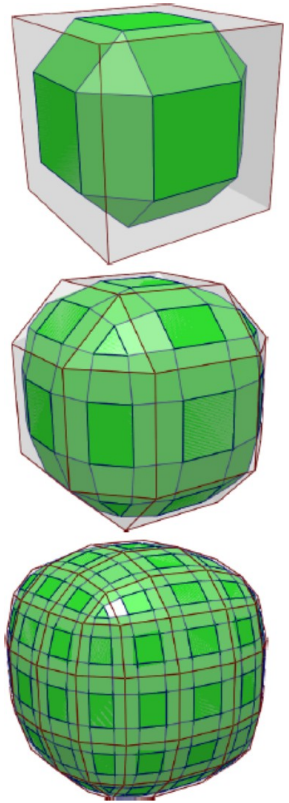
Given a face $(\mathbf{p}_i)_{i=[\![0,N-1]\!]}$

Compute middle vertex $m_i = (\mathbf{p}_i + \mathbf{p}_{i+1})/2$

Compute the barycenter of the face $\mathbf{b} = \frac{1}{N}\sum_{i=0}^{N} \mathbf{p}_i$

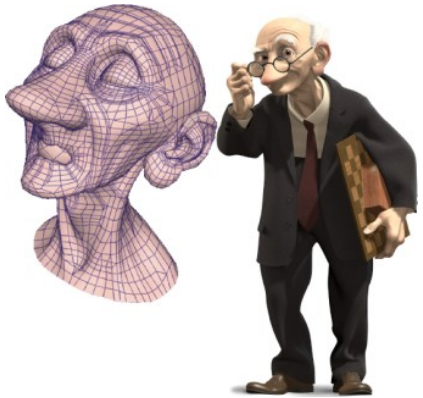
The new vertices are $\mathbf{n}_i = (\mathbf{p}_i + \mathbf{m}_i + \mathbf{m}_{i-1} + \mathbf{b})/4$



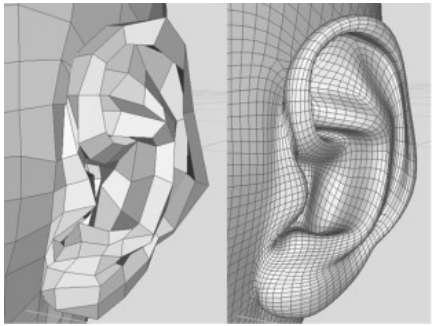


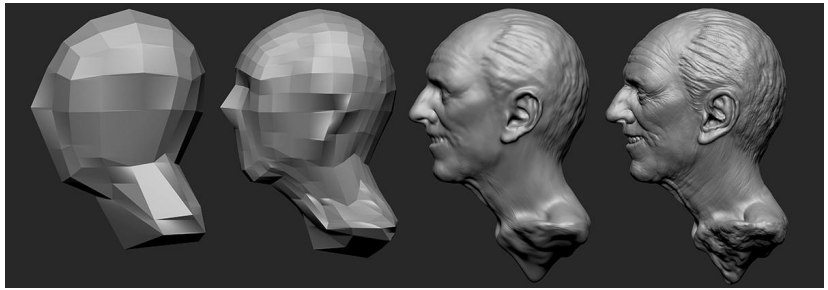
Subdivision surfaces applications

- Low resolution modeling.
- Iterative subdivision + addition of details (multiresolution)



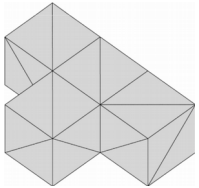
[Geri's Game, Pixar]

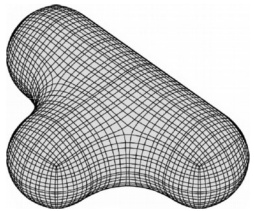
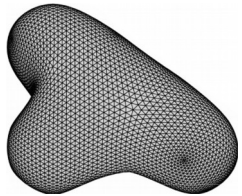
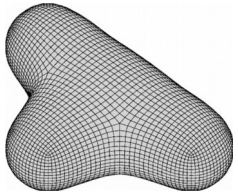
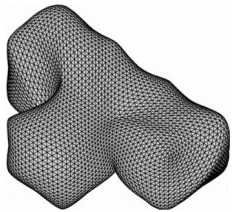




[Modeling with ZBrush]

Subdivision surfaces schemes



Doo-Sabin	**Loop**	**Catmull-Clark**	**Butterfly**
Dual C^2 approximation	Triangles C^2 approximation	Quadrangles C^2 approximation	Triangles C^1 interpolation
			

see http://graphics.stanford.edu/courses/cs468-10-fall

Several more schemes: Kobbelt, mid-edge, biquartic, etc

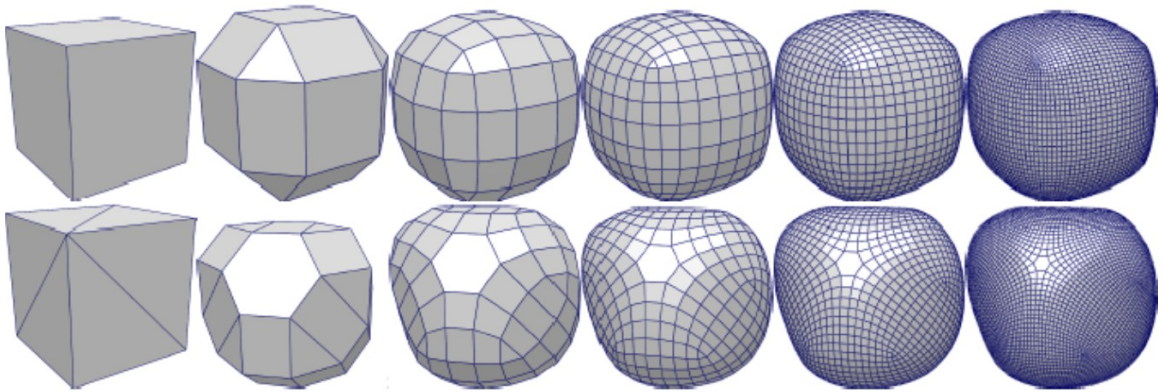
Subdivision surfaces: analysis

Pros

- Arbitrary input mesh topology
- Guaranteed convergence to C^1 / C^2 (excepted singular points).

Cons

- Hard to predict final shape (depends on scheme, connectivity)
- No direct access to differential parameters

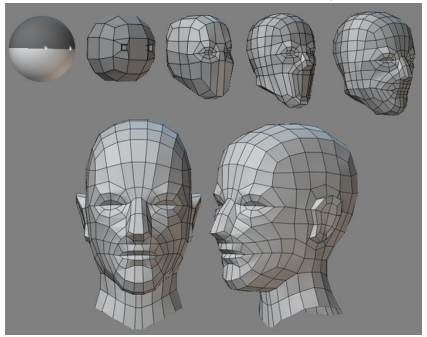


Wrap-up: Common manual modeling approaches

Mesh based modeling

1- Manual mesh modeling (coarse to fine): extrusion, deformation
"Low polygon"

2- Subdivision surface (smoothing+details)

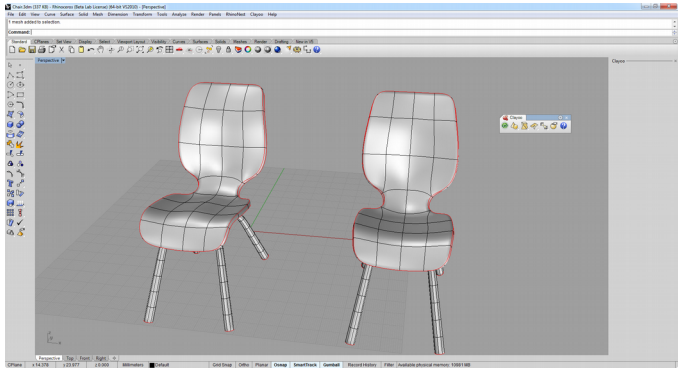


CG: digital artists

Parametric surface modeling

1- Choose a basis function (smoothness, locality, weights, etc).

2- Manually define the control polygon



CAD modeling

End of the 1st class on the 13/09

Softwares, tools, libraries for meshes & surfaces

Main 3D modelers

General mesh modeling

Commercials
- Maya (Autodesk)
- 3DS Max (Autodesk)
- Cinema 4D (Maxon)
- Modo (The Foundry)

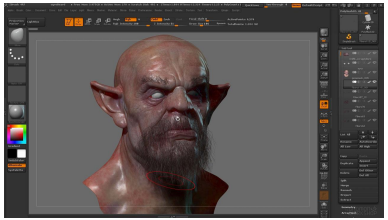
Free software
- Blender

Specialized modeling

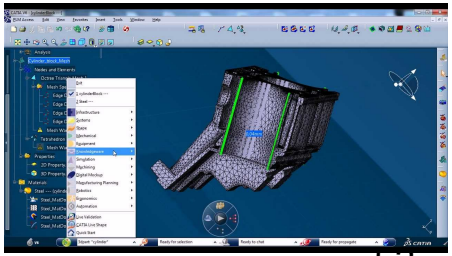
Sculpting details over meshes
- ZBrush (Pixologic)
- 3DCoat (Pilgway)

Design
- Sketchup

...



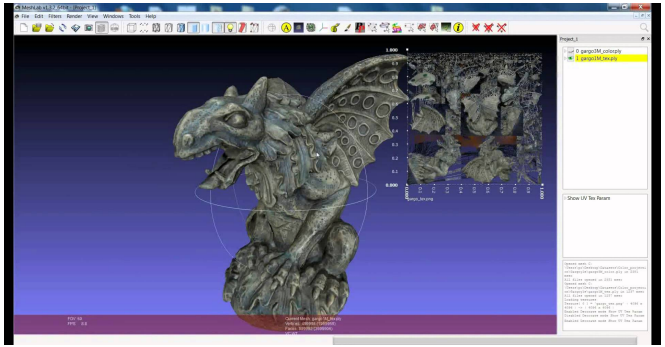
Parametric modelers
- Catia (Dassault Systèmes)
- AutoCAD (Autodesk)
- Rhino 3D (McNeel)



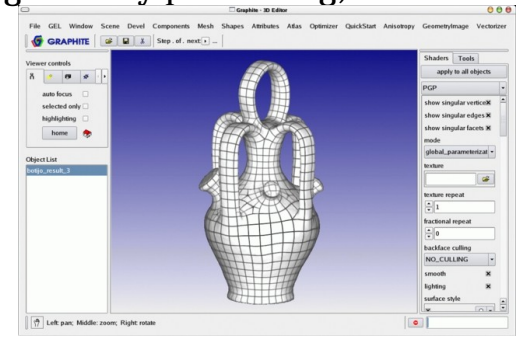
Softwares, **tools**, libraries for meshes & surfaces

Helping 3D tools (free open-source softwares)

- **Meshlab** (mesh viewer & processing)



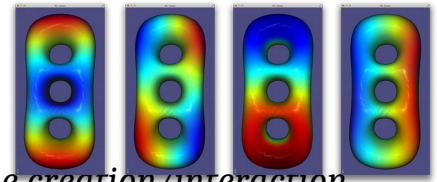
- **Graphite** (mesh viewer based on GEOGRAM & processing, API for geometry processing)



geomview (basic off viewer), wings 3D (subdivision modeler)

Softwares, tools, **libraries** for meshes & surfaces

CGAL: Advanced mesh data structure & processing (only for used C++ programmers)
OpenMesh: Mesh data structure (simpler than CGAL)
LibIGL: Library for geometry processing

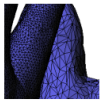


Assimp: library for mesh loading
Three.js: javascript framework for 3D scene creation/interaction

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Triangulation_3.h>

#include <iostream>
#include <fstream>
#include <cassert>
#include <list>
#include <vector>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

typedef CGAL::Triangulation_3<K>       Triangulation;

typedef Triangulation::Cell_handle    Cell_handle;
typedef Triangulation::Vertex_handle  Vertex_handle;
typedef Triangulation::Locate_type    Locate_type;
typedef Triangulation::Point          Point;

int main()
{
  // construction from a list of points :
  std::list<Point> L;
  L.push_front(Point(0,0,0));
  L.push_front(Point(1,0,0));
  L.push_front(Point(0,1,0));

  Triangulation T(L.begin(), L.end());

  Triangulation::size_type n = T.number_of_vertices();

  // insertion from a vector :
  std::vector<Point> V(3);
  V[0] = Point(0,0,1);
  V[1] = Point(1,1,1);
  V[2] = Point(2,2,2);

  n = n + T.insert(V.begin(), V.end());

  assert( n == 6 );   // 6 points have been inserted
  assert( T.is_valid() ); // checking validity of T

  Locate_type lt;
  int li, lj ;
  Point p(0,0,0);
  Cell_handle c = T.locate(p, lt, li, lj);
  // p is the vertex of c of index li :
  assert( lt == Triangulation::VERTEX );
  assert( c->vertex(li)->point() == p );

  Vertex_handle v = c->vertex( (li+1)&3 );
  // v is another vertex of c
  Cell_handle nc = c->neighbor(li);
  // nc = neighbor of c opposite to the vertex associated with p
  // nc must have vertex v :
  int nli ;
  assert( nc->has_vertex( v, nli ) );
  // nli is the index of v in nc

  std::ofstream oFileT("output",std::ios::out);
  // writing file output;
  oFileT << T;
```

```
function initWater() {

	var materialColor = 0x0040C0;

	var geometry = new THREE.PlaneBufferGeometry( BOUNDS, BOUNDS, WIDTH - 1, WIDTH -1 );

	// material: make a ShaderMaterial clone of MeshPhongMaterial, with customized vertex shader
	var material = new THREE.ShaderMaterial( {
		uniforms: THREE.UniformsUtils.merge( [
			THREE.ShaderLib[ 'phong' ].uniforms,
			{
				heightmap: { value: null }
			}
		] ),
		vertexShader: document.getElementById( 'waterVertexShader' ).textContent,
		fragmentShader: THREE.ShaderChunk[ 'meshphong_frag' ]

	} );

	material.lights = true;

	// Material attributes from MeshPhongMaterial
	material.color = new THREE.Color( materialColor );
	material.specular = new THREE.Color( 0x111111 );
	material.shininess = 50;

	// Sets the uniforms with the material values
	material.uniforms.diffuse.value = material.color;
	material.uniforms.specular.value = material.specular;
	material.uniforms.shininess.value = Math.max( material.shininess, 1e-4 );
	material.uniforms.opacity.value = material.opacity;

	// Defines
	material.defines.WIDTH = WIDTH.toFixed( 1 );
	material.defines.BOUNDS = BOUNDS.toFixed( 1 );
```

Triangulated Surface Mesh Simplification

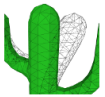


Fernando Cacciola

This package provides an algorithm to simplify a triangulated surface mesh by edge collapsing. It is an implementation of the Turk/Lindstrom *memoryless* surface mesh simplification algorithm.

User Manual Reference Manual

Triangulated Surface Mesh Deformation

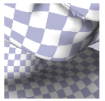


Sébastien Loriot, Olga Sorkine-Hornung, Yin Xu and Ilker O. Yaz

This package offers surface mesh deformation algorithms which provide new positions to the vertices of a surface mesh under positional constraints of some of its vertices, without requiring any additional structure other than the surface mesh itself.

User Manual Reference Manual

Triangulated Surface Mesh Parameterization



Laurent Saboret, Pierre Alliez and Bruno Lévy

Parameterizing a surface amounts to finding a one-to-one mapping from a suitable domain to the surface. In this package, we focus on triangulated surfaces that are homeomorphic to a disk and on piecewise linear mappings into a planar domain. This package implements several surface mesh parameterization methods, such as least squares conformal maps, discrete conformal map, discrete authalic parameterization, Floater mean value coordinates or Tutte barycentric mapping.

User Manual Reference Manual

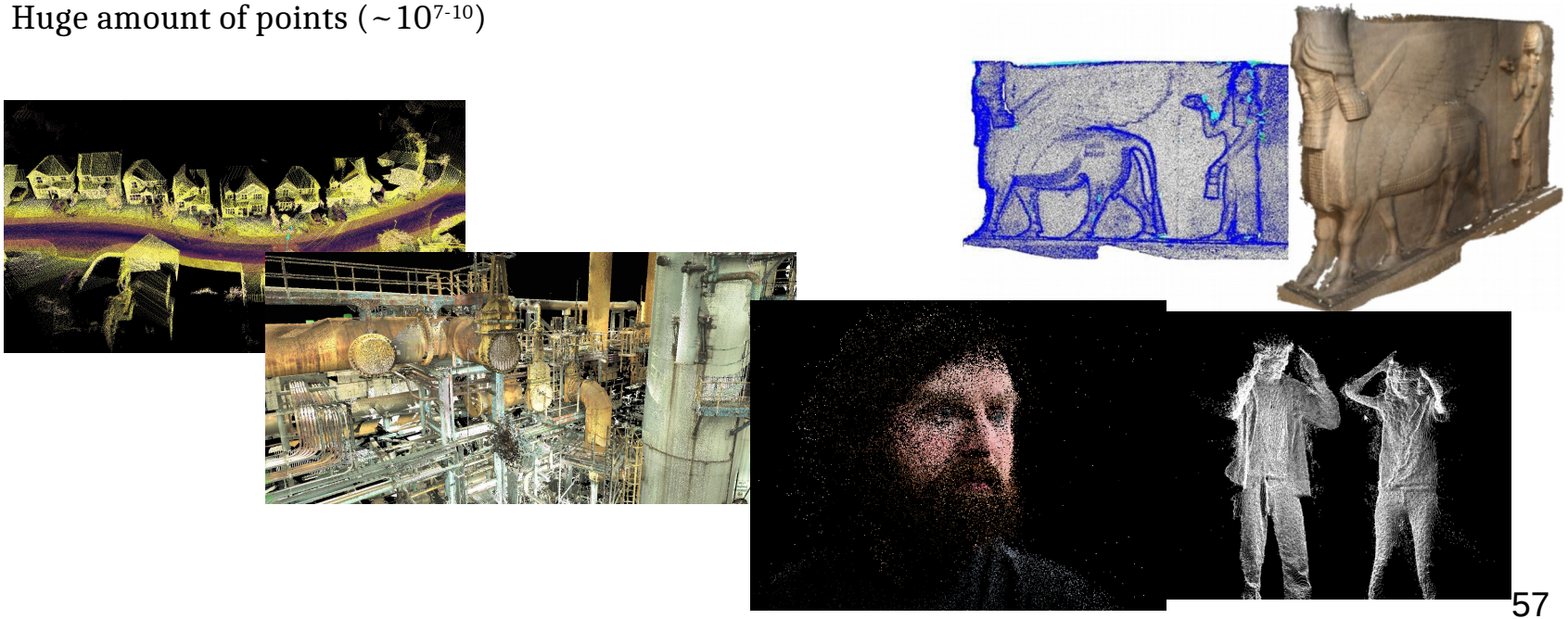
Explicit surface representation

I. Meshes
II. Parametric surfaces
III. Subdivision surfaces
IV. Point cloud

Point cloud

3D Scanner is developing: acquire raw 3D point cloud

- No explicit connectivity
- Huge amount of points ($\sim 10^{7\text{-}10}$)



Point cloud manipulation

2 Choices:

1- Reconstruct a mesh: problem of noise/size of data

2- Work directly with the points: assume a dense sampling

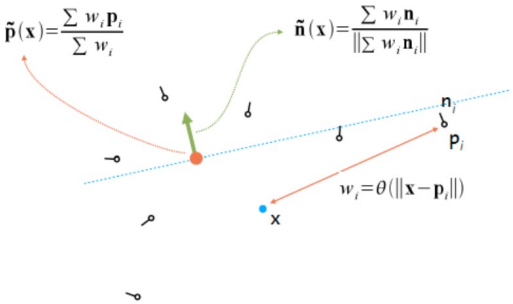
Local analysis
(normals, curvature)

ex. Compute local covariance matrix

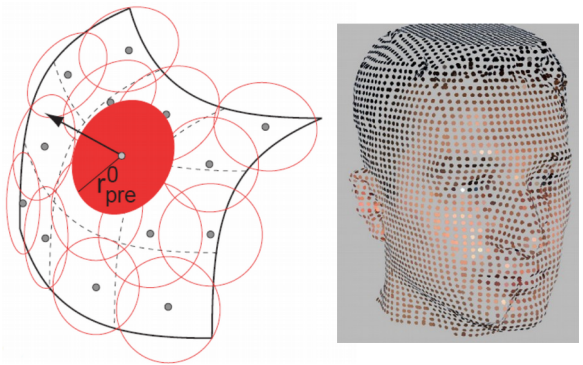
$$\sum_i (p_i - \overline{p})(p_i - \overline{p})^T$$

Normal oriented along eigenvector associated to the smallest eigenvalue

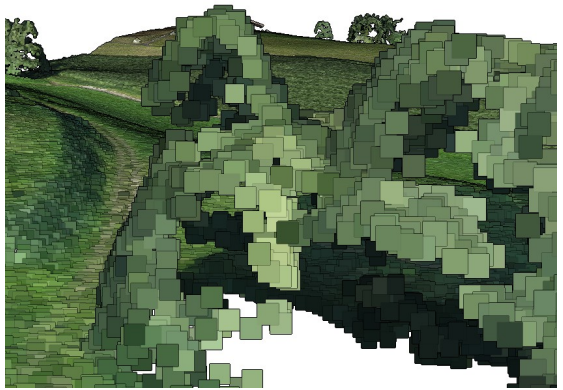
Interaction through space deformation
(Moving Least Squares)



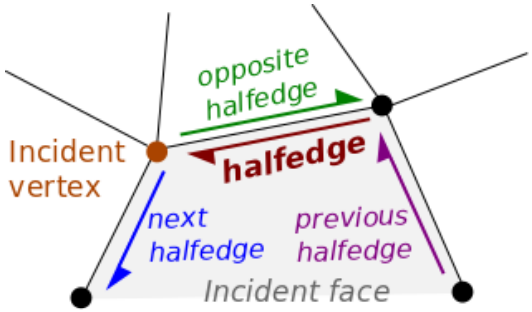
Rendering
(surface splatting)

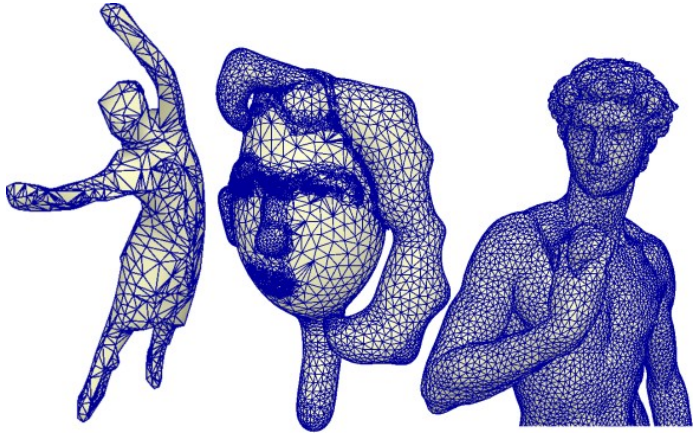


Point cloud example



Next week:
Mesh representations





And after: Volumes