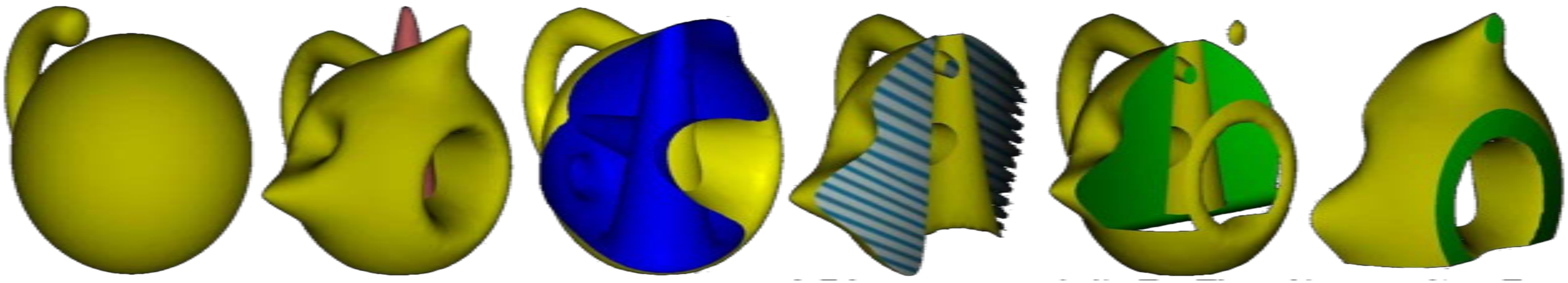


Direct Rendering of Boolean Combinations of Self-trimmed Surfaces

Jarek Rossignac, Georgia Tech

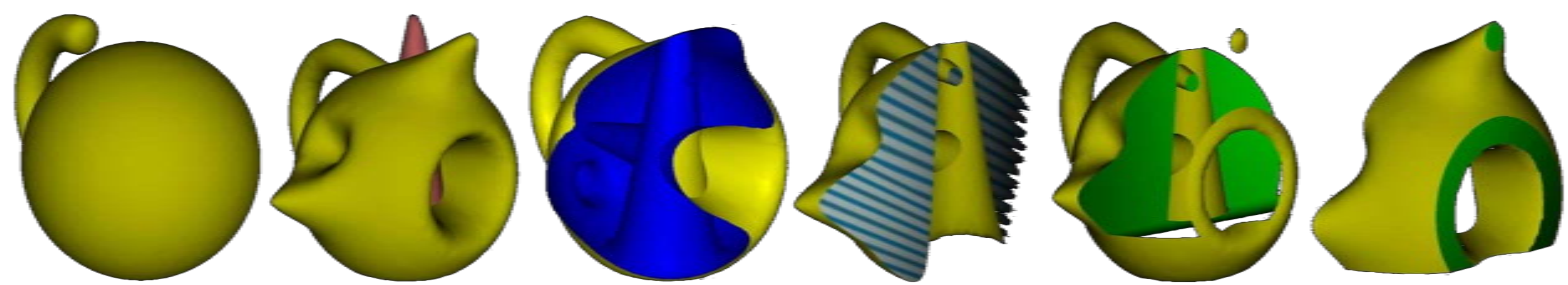
Ioannis Fudos, University of Ioannina

Andreas Vasilakis, University of Ioannina

Rendering (Boolean combinations of) self-trimmed surfaces

Objective: Provide a formal definition of the **interior** of the manifold solid defined by a **self-crossing surface.** Ensures that the definition produces intuitive results during simple local deformation (mimicking union, intersection, difference) and over more complex deformation.

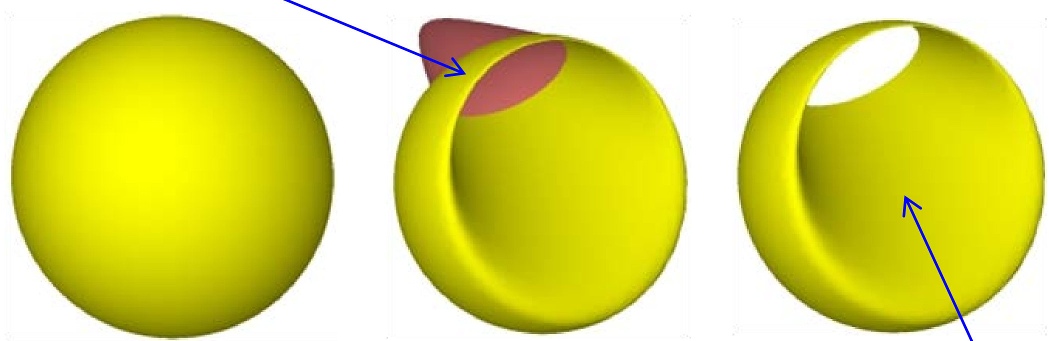
Applications: Preview of advanced (free-form) CAD operations where self-crossings extend Boolean. Animation of deforming objects.



Self-Trimmed Surface: Problem Formulation

A surface S is manifold when it is a compact and orientable 2-manifold without boundary

We say that S is a **Self-Crossing Surface** (**SCS**) when its immersion contains non-manifold intersection edges where S passes through itself.



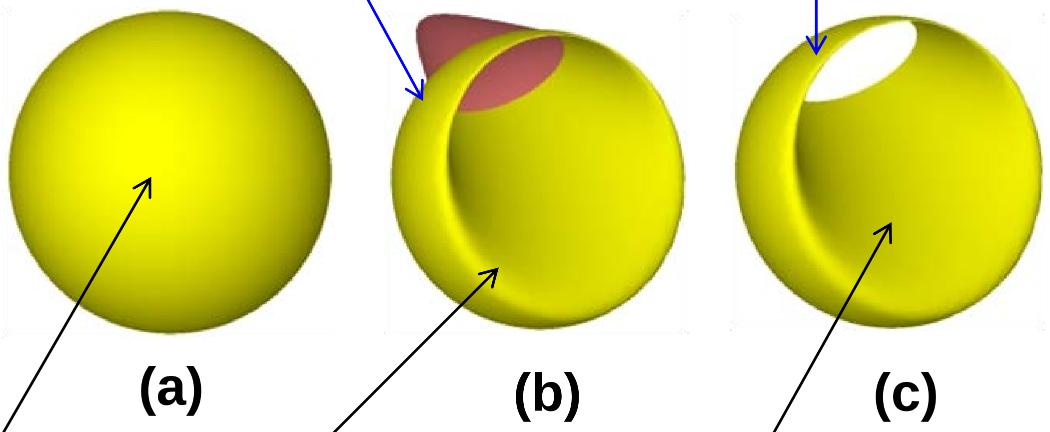
How to define the **interior**, I(S), of S?

We want to **render** the subset **T(S)** of S that is the **boundary** of **I(S)**

S is a **Self-Trimmed Surface** (**STS**) and T(S) is the **trim** of S

Self-trimmed surfaces: two problems

***Pb 1**: Given a self-crossing surface determine its trim*



***Pb 2**: Given S_0, an initial manifold boundary that is not self-crossing, and a continuous process D_t that deforms it to produce a time dependent surface S_t, determine its trim $T(S_t)$*

Pb 1 is captured by static rules that depend only on the self crossing surface

Pb 2 is captured by dynamic rules that depend on the deformation path

Advantages of STS rendering

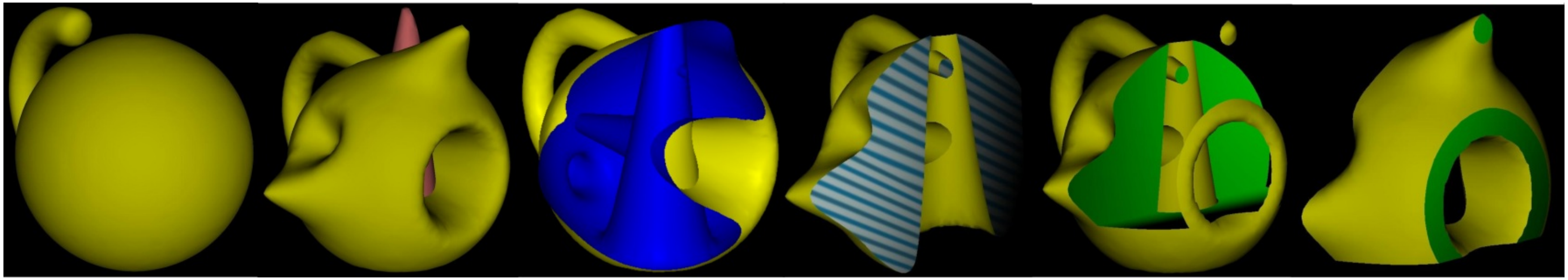
Eliminate the need for computing self-crossing curves

Provide realtime rendering of STS at variable LoD

Provide realtime inspection tools for STS (clipping, capping)

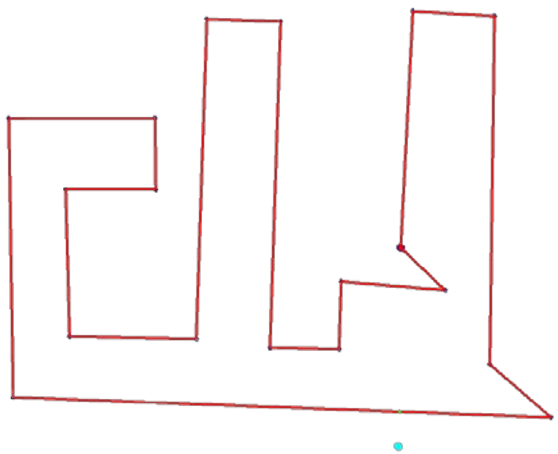
Treat STS as “valid” primitives in CAD, including CSG

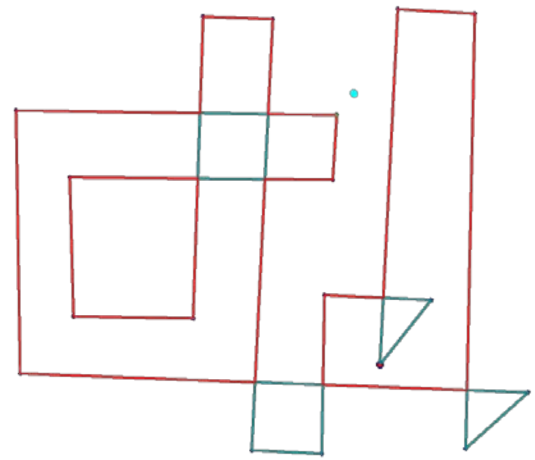
Support animations that involve Booleans and time dependent STS



Overview: Primitive deformation

Primitive deformation: each point is **crossed** only **once** by S





Alternating Boundary Rule (ABR):
The status (in-trim or not) **alternates at** each **self-crossing**

Two complementary solutions: Which one to select?

Relation to **winding number** (index)?

I(S) = {P: w(P) % 4 == 1 or 2}

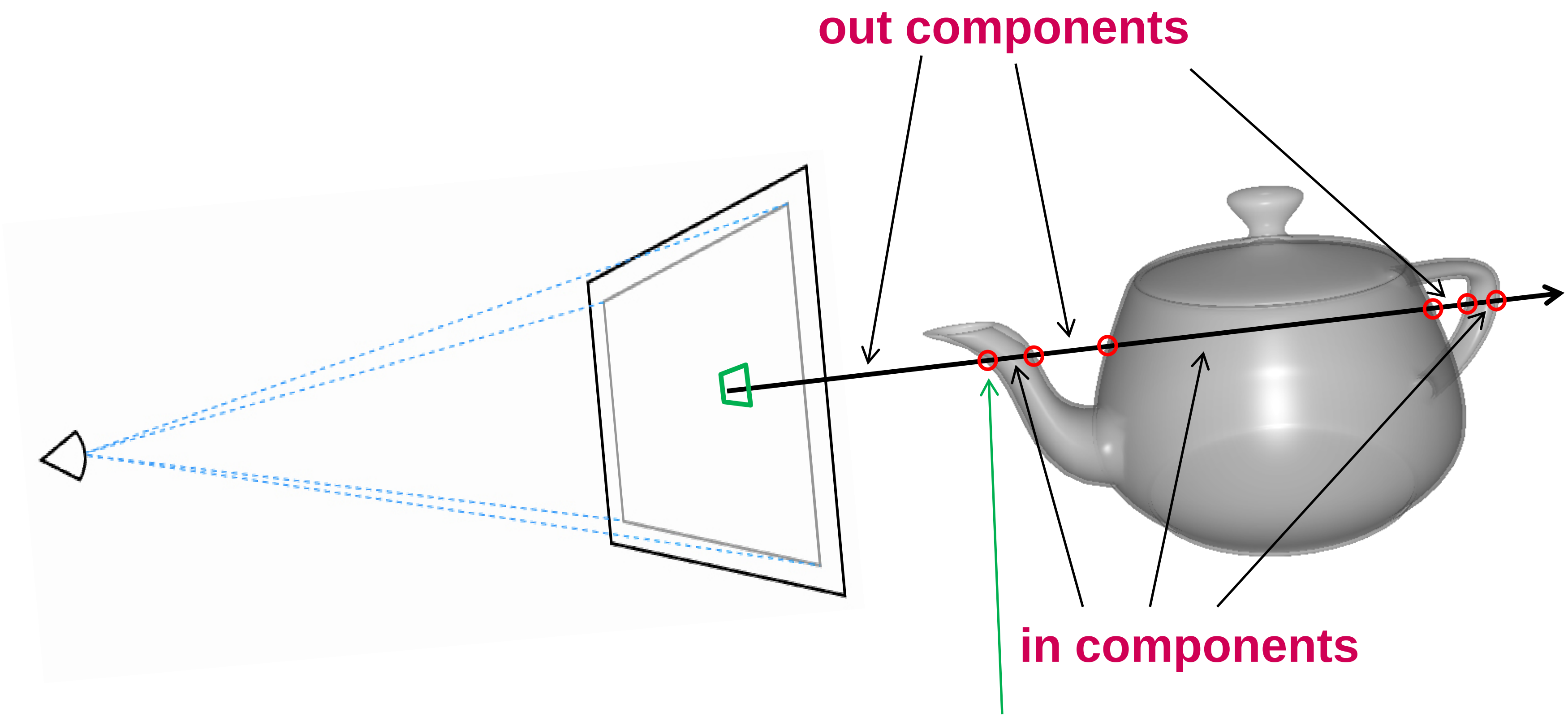
Overview: Complex, animated deformations

Decompose animation into series of simple deformations

At each frame, at each pixel:

For points just in front and just behind each surfel
compute winding number (before and after)
Conclude in-trim classification

Using fragments (surfels) from ray casting to determine and render interior/exterior and render the trim



Compute in and out components by using the fragment information, find the first fragment between an in and an out component and render it.

Background

A self crossing surface (SCS) S partitions the 3D space W into open components, one of which is infinite (denoted by C_0).

C_i: interior (in) or exterior (out)

I(S) is the closure of the union of C_i

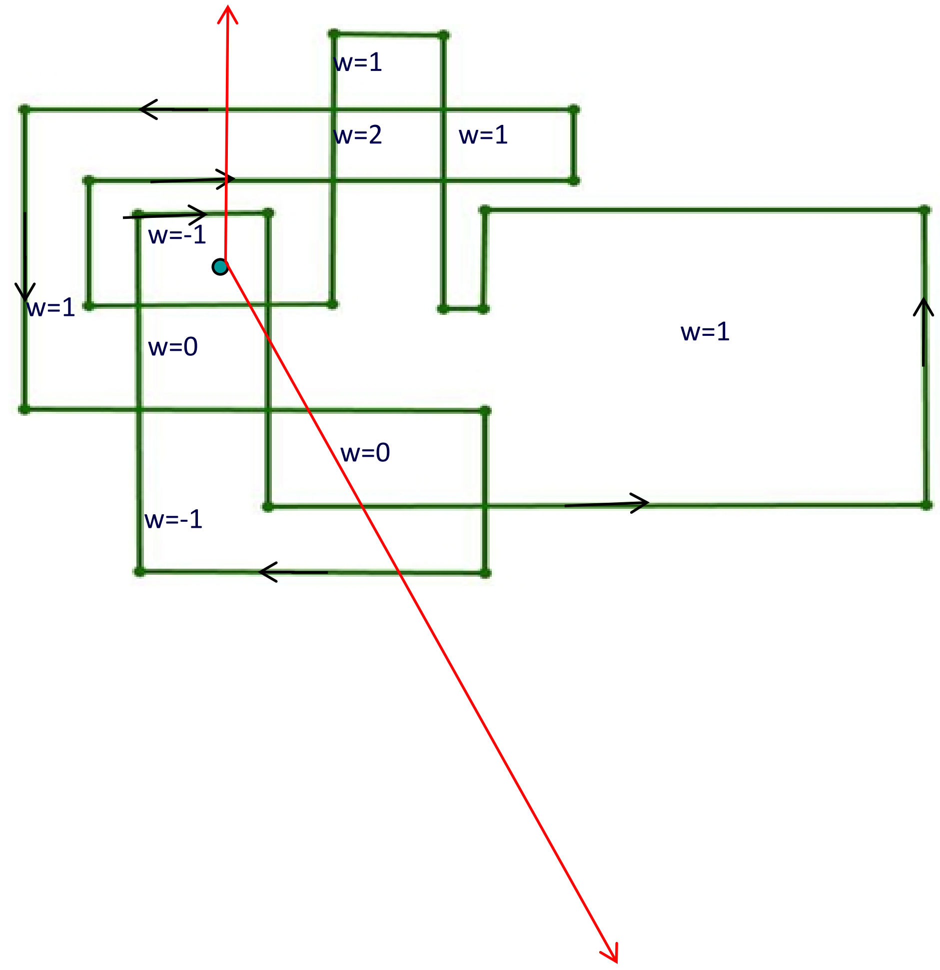
The trim T(S) is simply the boundary of I(S)

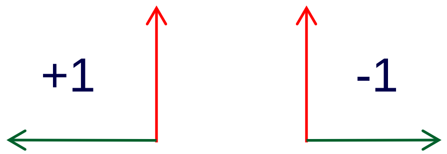
Boundary diminishing property: $T(S) \subseteq S$

Static rules for 2D: winding number or point index: given a self crossing closed loop C around a given point p, the index of p is the number of times that the curve travels counter clockwise around the point.

Winding number

The count of signed **crossings** of a **ray** to a point (the view point) outside.





Signed crossings

Independent of the choice of ray

Going 3D

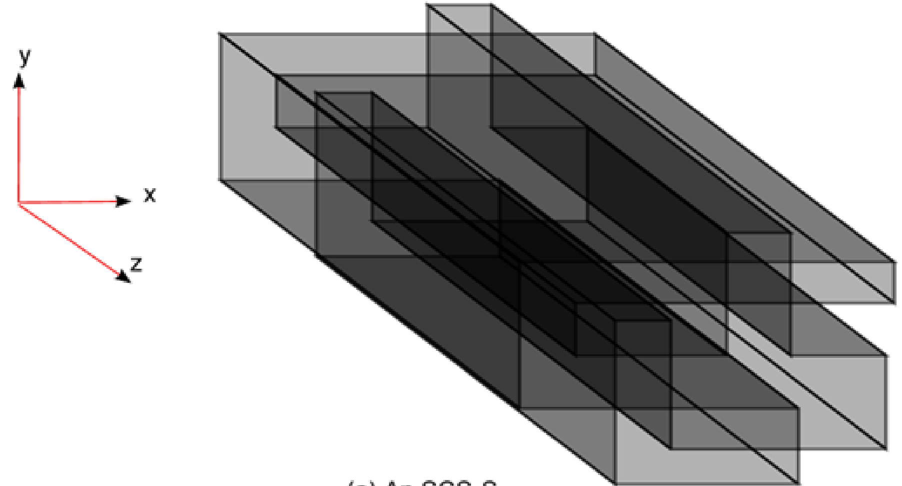
In 3D, the index w(p,S) of a point p with respect to an SCS S:

We assume that p is not on S. Consider any path P from p to infinity.

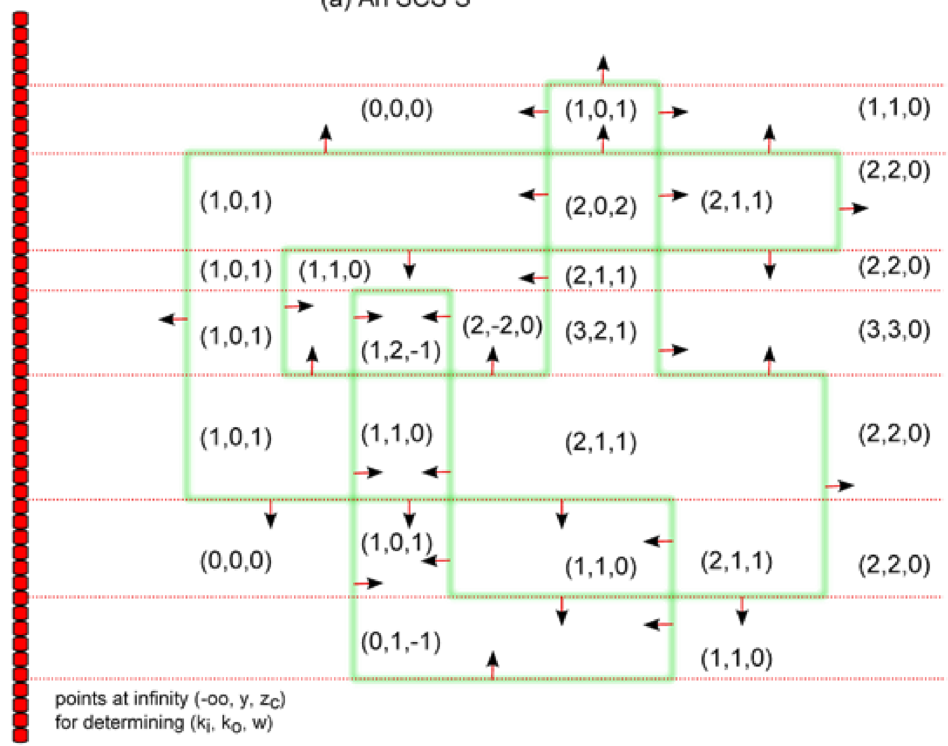
k_i # of times that P enters S (# of front facing normals)

K_o # of times that *P* **exits** *S* (# of back facing normals)

Then, $w(\boldsymbol{p},S) = k_i - k_o$.



(a) An SCS *S*

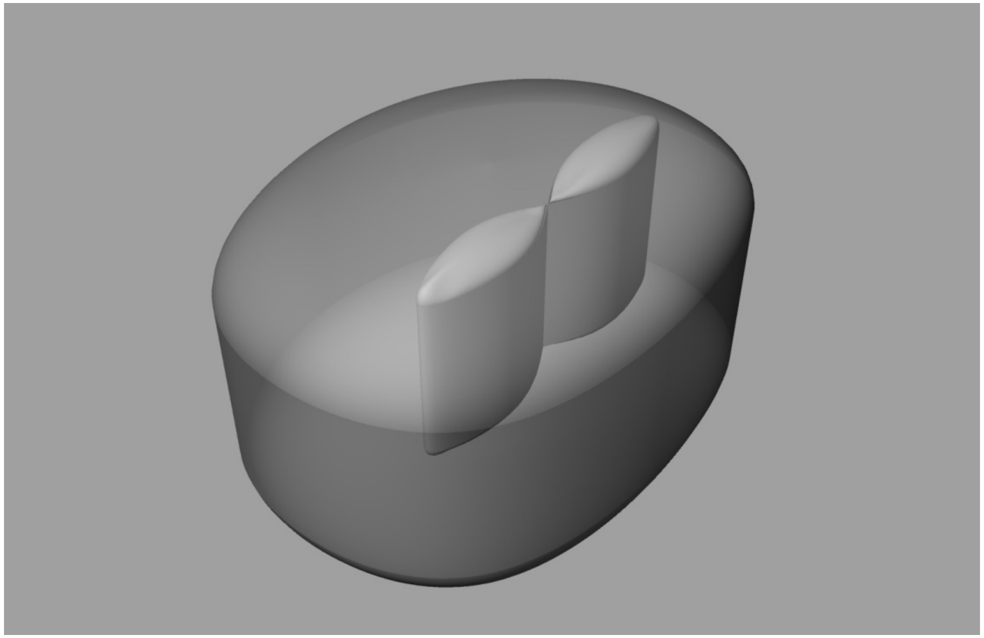


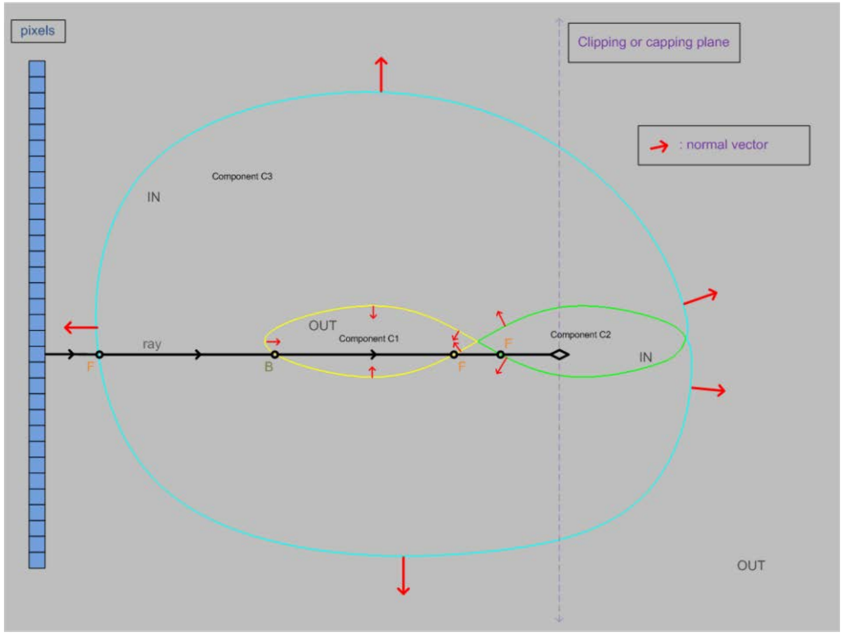
(b) A cross section of *S* on plane $z=z_c$

Going 3D

The point index is the signed generalization of the *overlap count*

Overlap count: the unsigned count of the number of surfels that correspond to a pixel (used for transparency effects)





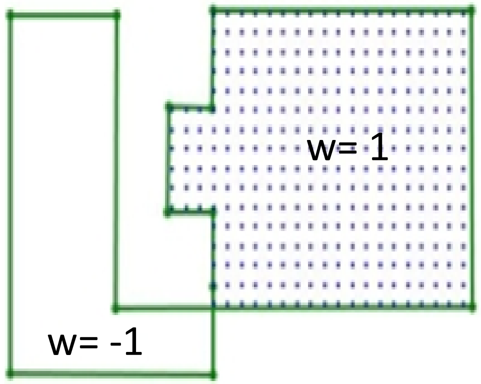
Static rules for in/out classification

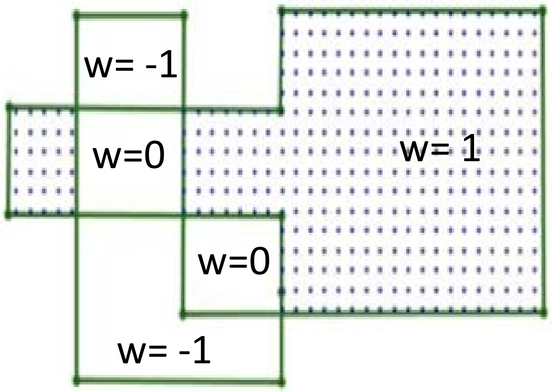
Here we present three static rules

1. Positive index rule
2. Parity or alternating component rule
3. Alternating border rule

Static Rules for in/out classification: positive index

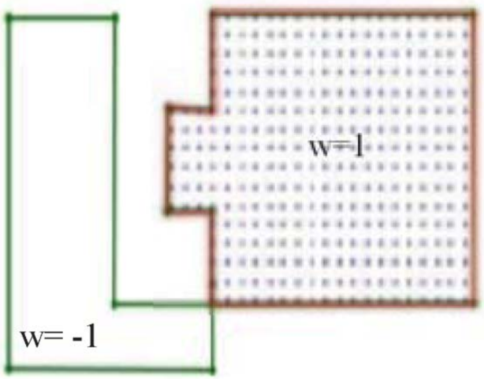
Positive Index Rule: interior are all components with positive index (depends on the orientation of the surface) [Heisserman 1992]

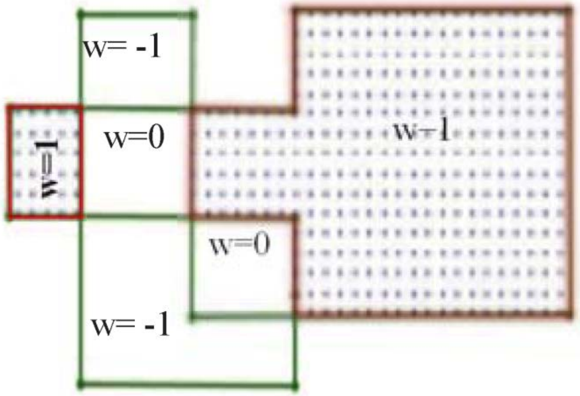




Static Rules for in/out classification: positive index

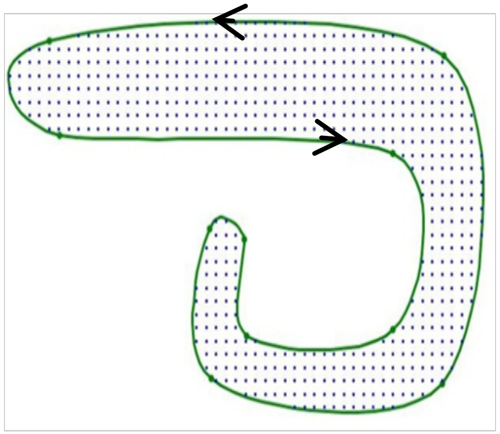
Positive Index Rule: interior are all components with positive index (depends on the orientation of the surface) [Heisserman 1992]

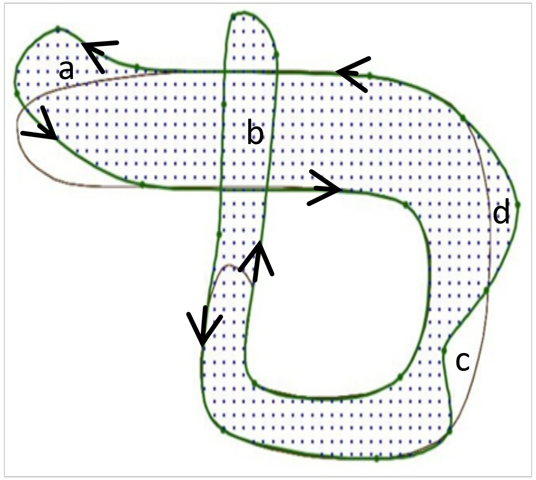




Static Rules for in/out classification: positive index

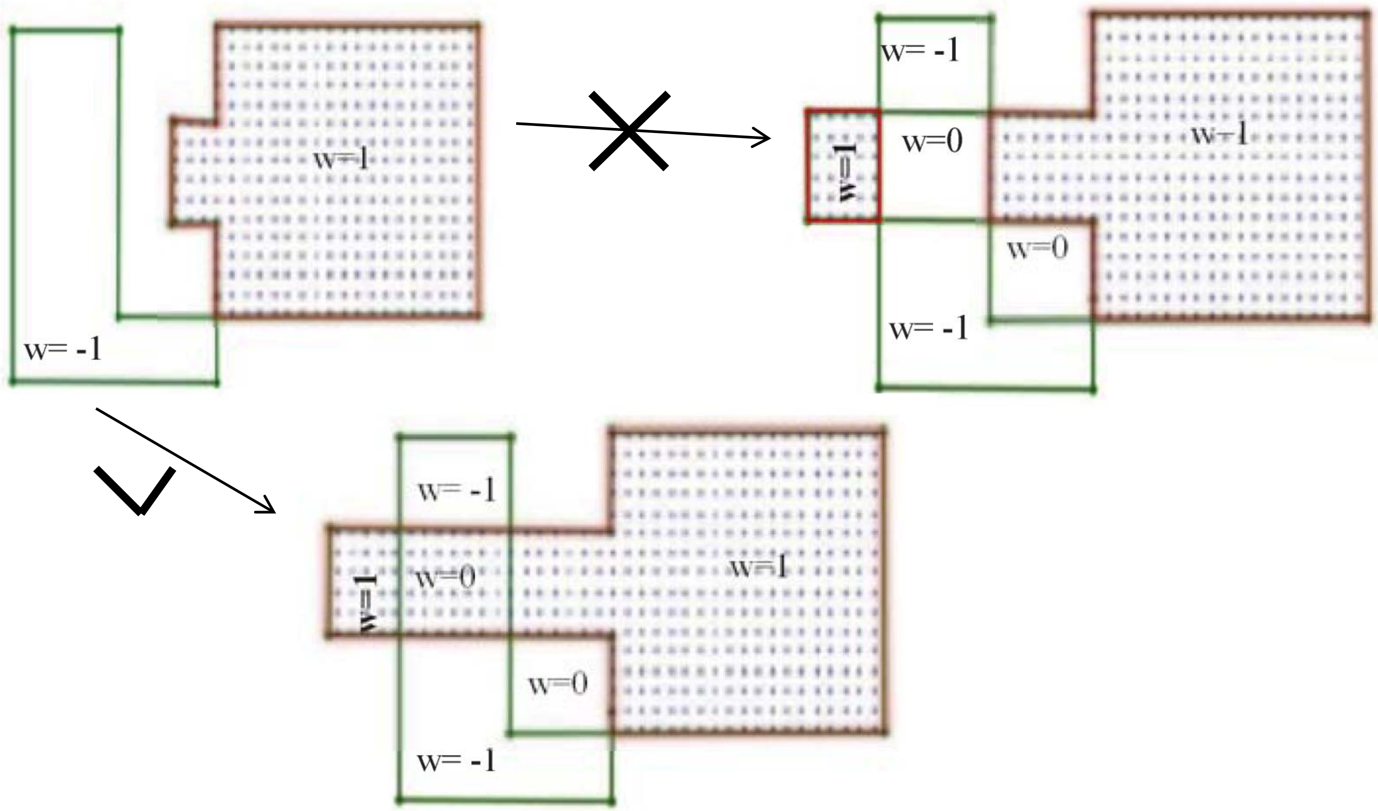
Top tip wagging, bottom tip extending, bumps all captured by the positive index rule.





Static Rules for in/out classification: positive index

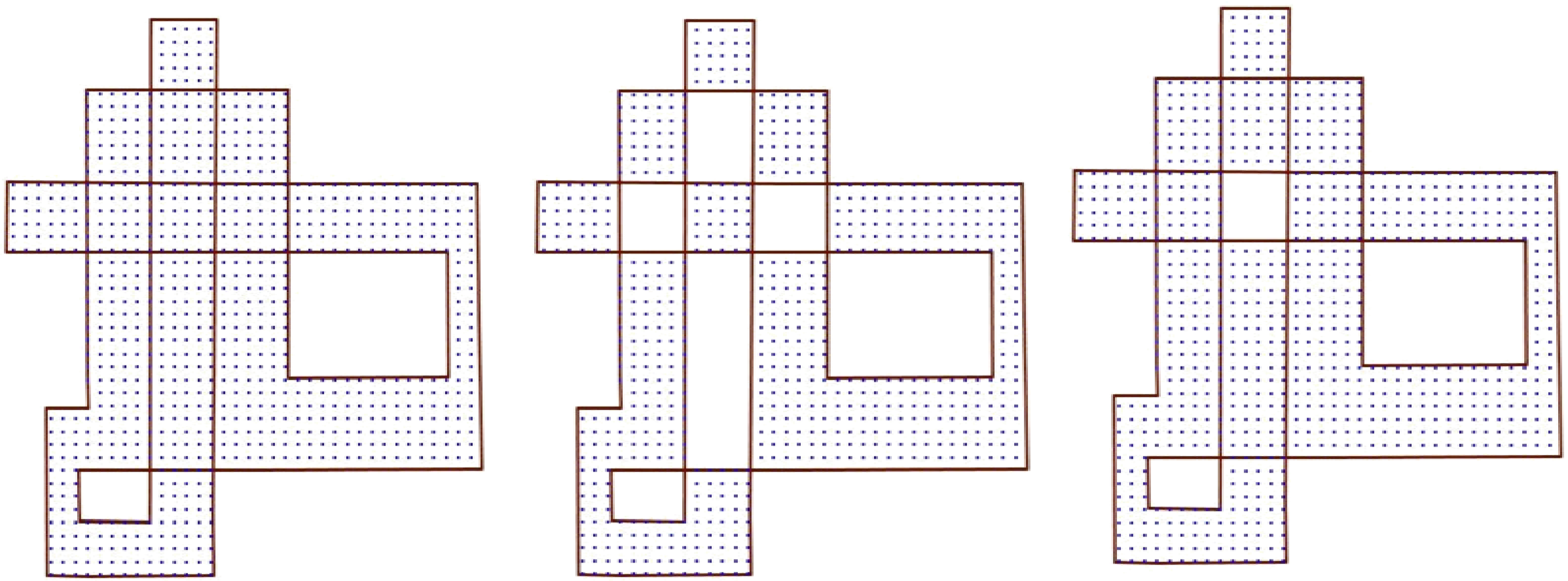
Cannot capture deformations. Complement is difficult to obtain.



Static Rules for in/out classification: parity and alternating border rules

Parity or alternating component rule: p is interior iff w(p) is odd. Does not trim anything, derives highly non-manifold objects. (b) Behave well in surface orientation modification.

Alternating border rule: p is interior iff $\left\lceil \frac{w(p)}{2} \right\rceil$ is odd (c).



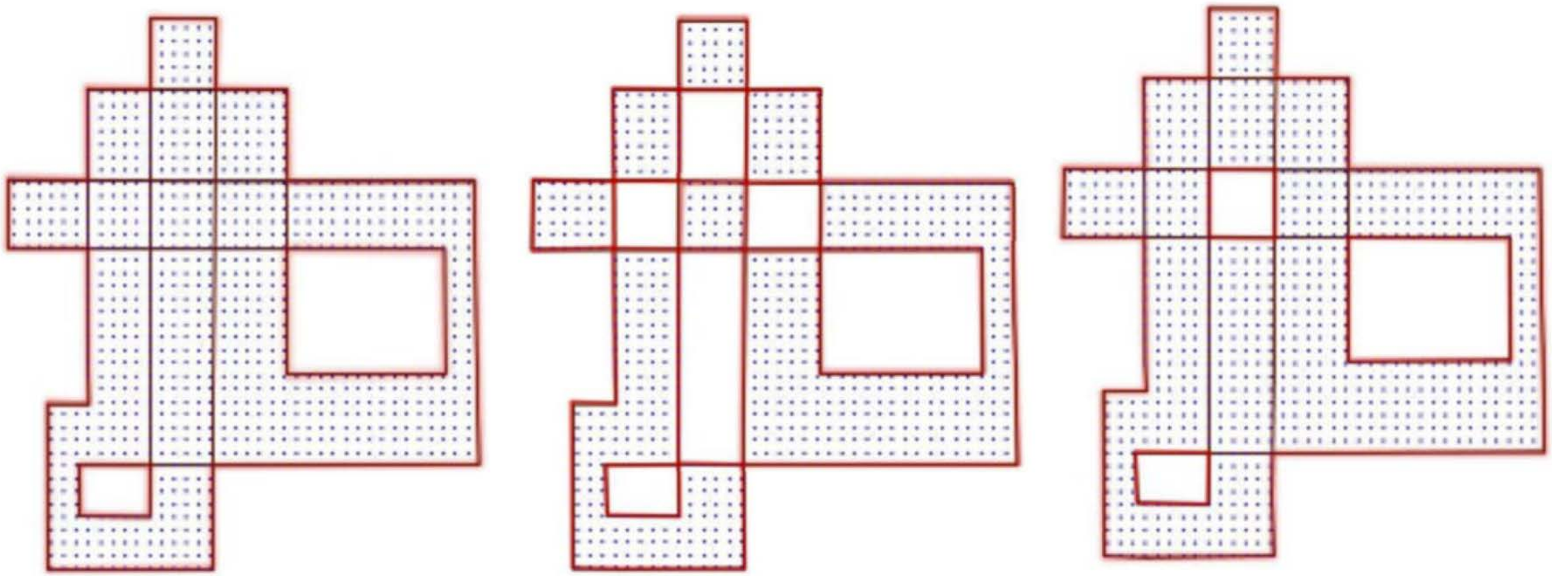
(a) (b) (c)

Determining the trim: parity and alternating border rules

For the parity rule, no trimming occurs (b). STS == SCS

Theorem (alternating border rule): adjacent faces have opposite classification with respect to the trim.

Lemma: The alternating border rule does not produce non manifold solids with simple self crossings (c)



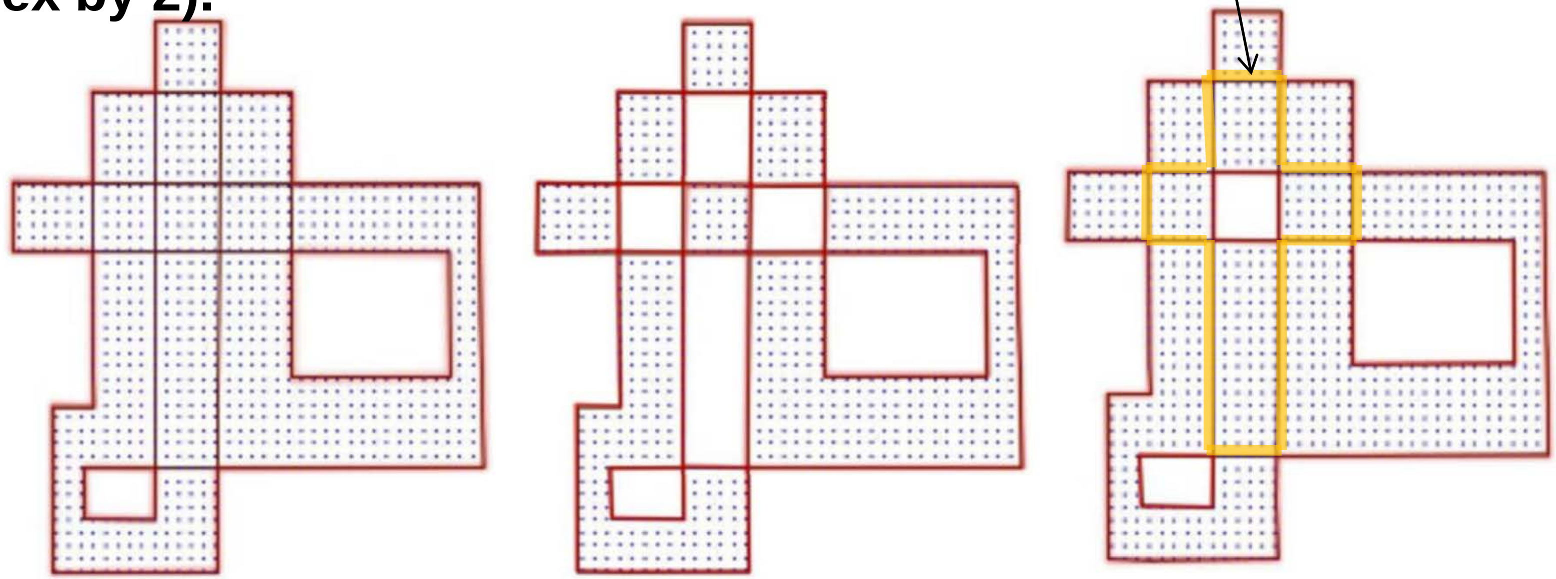
(a) (b) (c)

Static Rules for in/out classification: alternating border rule

Alternating border rules yields maximal genus non-manifold objects.

Changing surface orientation => complement of the STS (see orange lines in (c))

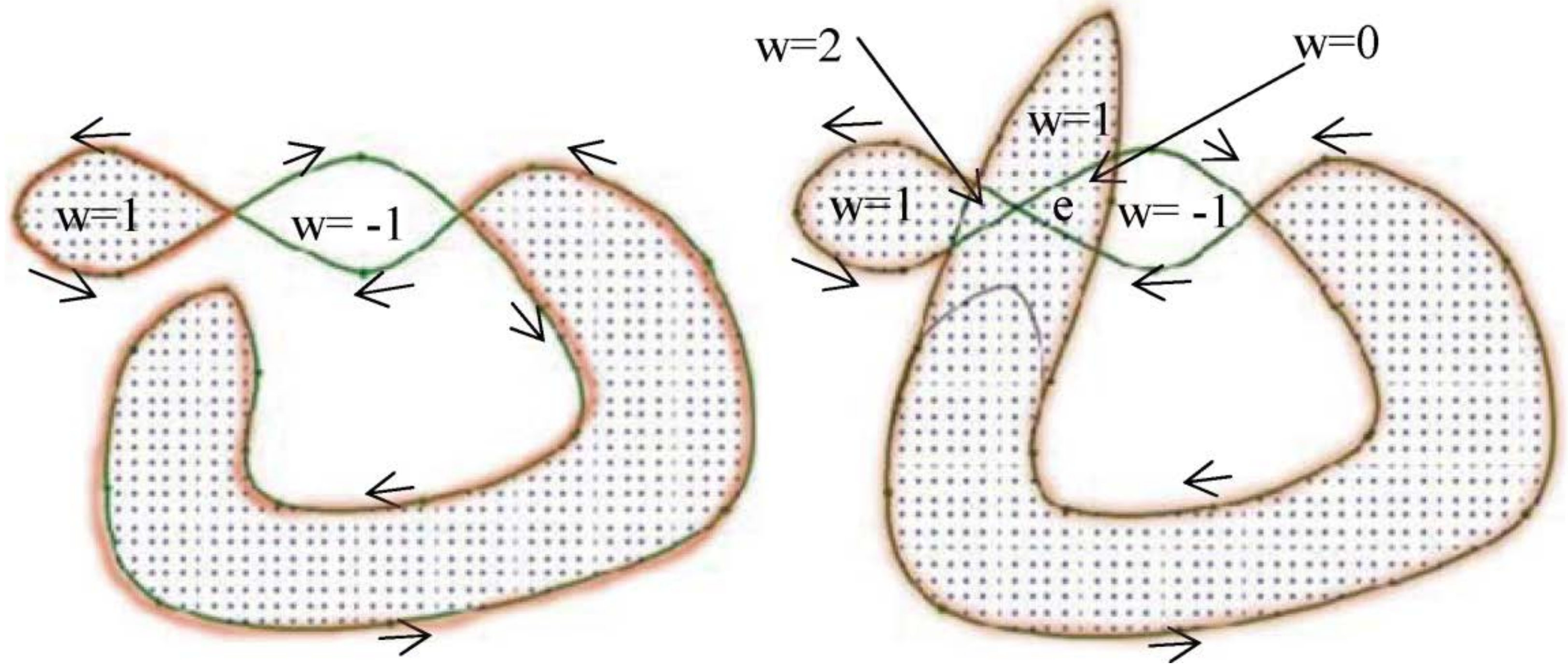
To obtain the complement add two surrounding boxes (increase index by 2).



(a) (b) (c)

Static Rules for in/out classification

Static rules do not capture well deformations …



Dynamic Rules for in/out classification

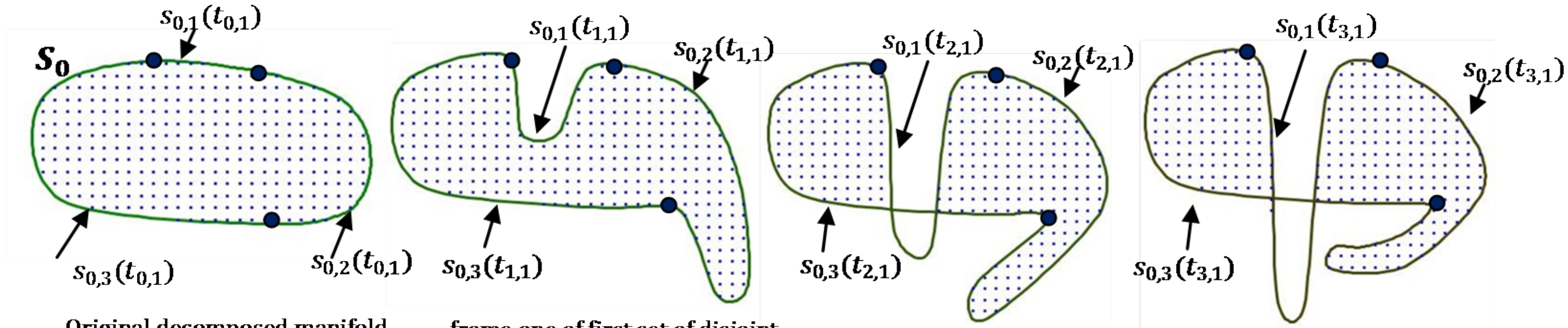
Extension-normal confluence property: When deforming a surface towards the normal the points crossed either become interior or they are not affected.

Complement symmetry property: If we apply the same deformations on the complement we obtain the complement of the result.

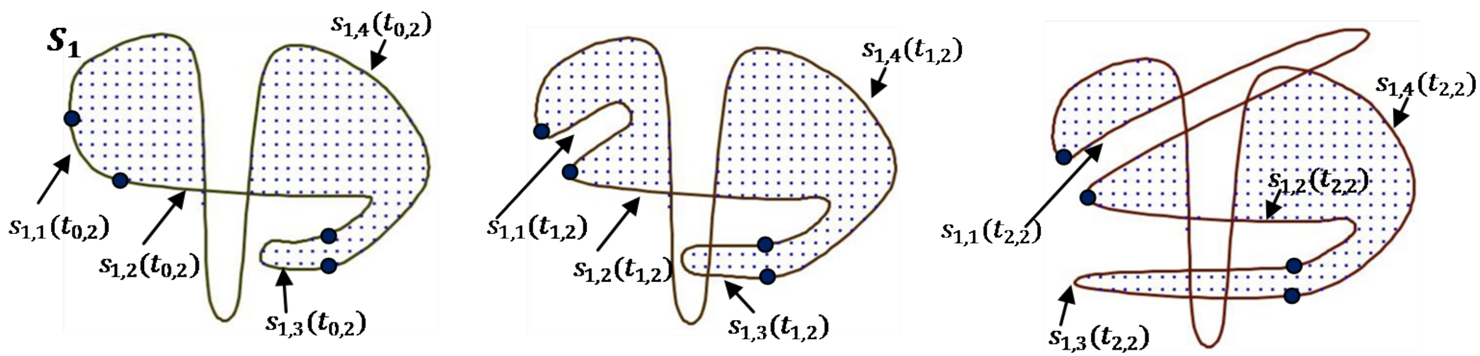
Component homogeneity property: Each component contains only interior or only exterior points. This is a very important property since otherwise the border of the interior parts may not be a subset of the deformed initial surface. This is equivalent to the aforementioned ***boundary diminishing*** property.

Modeling complex deformations

We can apply and render sequences of primitive disjoint deformations. We apply the first set of primitive disjoint deformations: the deformations in this set are non intersecting and non self intersecting. Then we apply the second set. The first frame of each set is identical to the last frame of the previous set and is the reference frame for the current frame.



Original decomposed manifold, reference frame for f_1 — frame one of first set of disjoint concurrent deformations f_1 — frame two of f_1 — frame three of f_1



Decomposed manifold after applying f_1, reference frame for f_2 — frame one of second set of disjoint concurrent deformations f_2 — frame two of f_2

Modeling Deformation for Dynamic Rules

In the rest, we use as previous frame the reference frame, i.e. a frame after which we have applied only one set of disjoint concurrent deformations.

This means that the border will cross a point p only once in each set of disjoint concurrent deformations.

Dynamic Rules: Constructive rule

$$i(p,S)= \begin{cases} 1, \text{ iff } p \text{ is interior point} \\ 0, \text{ iff } p \text{ is exterior point} \\ \text{undefined, if } p \text{ is on the boundary T(S)} \end{cases}$$

For the initial classification use any static rule, e.g.: $i(p,S_0) = \left\lfloor \frac{w(p,S_0)}{2} \right\rfloor mod\, 2$

For the classification of surface S'= S_t based on the previous frame S= S_{t-1}: A point belongs to the newly created volume iff

$$(w(p,S) \neq w(p,S'))$$

Thus

$$ip(p,S') = i(p,S)\, op\, (w(p,S) \neq w(p,S'))$$

A op B can be: logical OR (union – additive operation), logical AND (intersection), logical difference (A ∧ ¬B)

Dynamic Rules: Constructive Rule

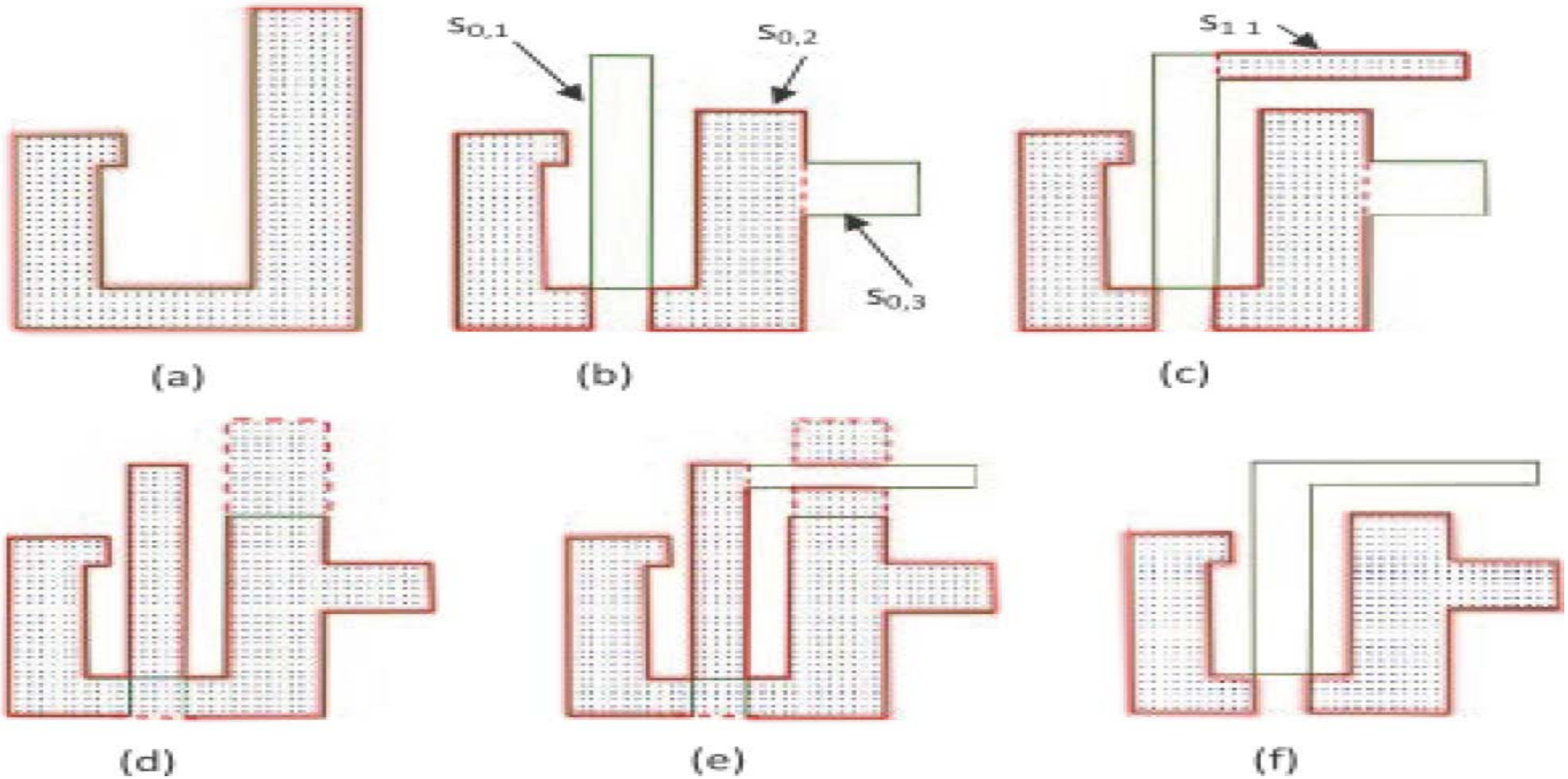
Extension-normal confluence property: No

Complement symmetry property: No

Component homogeneity property: No

Dynamic Rules: Constructive Rule

Although this rule captures design intent and has a constructive nature, it does not preserve component homogeneity.
This may yield highly non intuitive and formally ill-defined results, where the boundary B may not be part of the initial surface S. To address this problem, we use only the confluent deformation rule.



Dynamic Rules: Confluent Deformation Rule

For the initial classification of surface S_0 any static rule will do, e.g.:

$$i(p, S_0) = \left\lceil \frac{w(p, S_0)}{2} \right\rceil mod 2$$

For the classification of surface

$S' = S_t$ based on the previous instance $S = S_{t-1}$:

if $(w(p,S) == w(p,S'))$ then $i(p,S') = i(p,S)$ else

This means that it is 1 if the point index is increased and 0 if the point index is decreased.

$$i(p, S') = \left\lceil \frac{w\left(p, S'\right) - w(p, S)}{2} \right\rceil mod 2$$

***Extension-normal confluence property:* Yes**

Complement symmetry property: Yes

Component homogeneity property: Yes if we enforce a restriction on the type of acceptable deformations

Dynamic Rules: Homogeneous Confluent Deformation Rule

A part s_{out} of surface *S* that is not part of the border *B* cannot be deformed towards the normal if s_{out} is between two exterior components. Likewise a part s_{in} of a surface *S* that is not part of border *B* cannot be deformed in a direction opposite to its normal if s_{in} is between two interior components.

Implementation wise this is a challenging rule to enforce. To enforce the deformation restriction we need to detect trimmed parts of the surface, i.e. parts that do not belong to the border and the interior/exterior classification of the adjacent components.

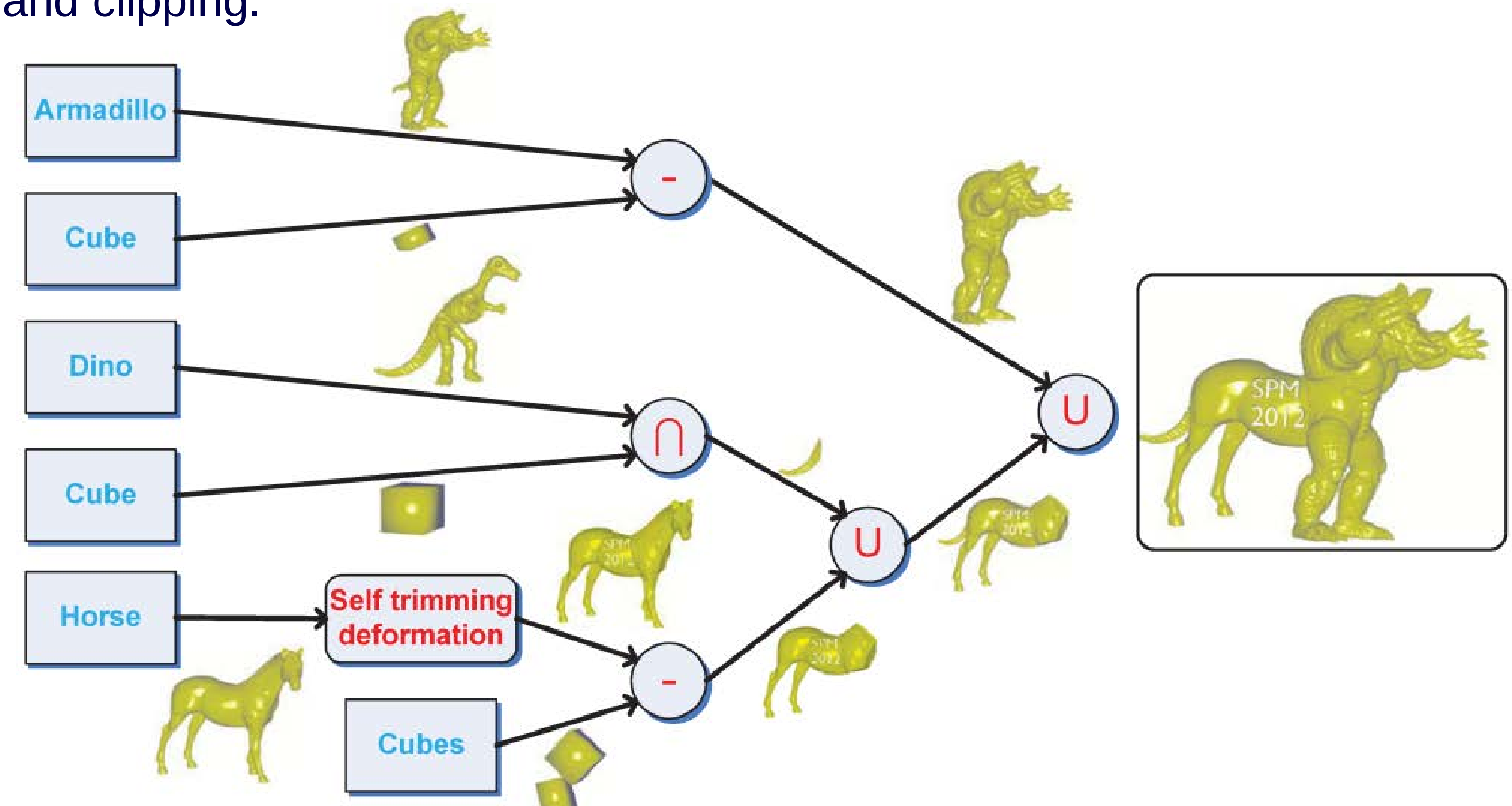
For the purposes of user interaction it is more intuitive to be more restrictive and prohibit deformations on all non-border (trimmed) surface parts.

Applications: CSG, LOD, capping

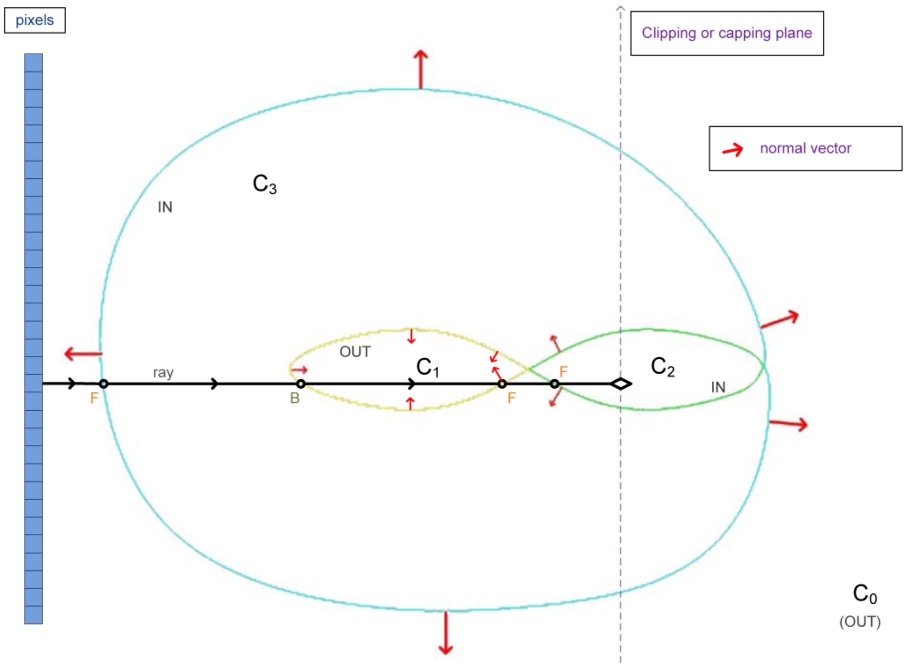
Boolean operations of solids obtained by self crossing surfaces

Change LOD at any point of the rendering process (consider this as deformation).

Capping and clipping.



Rendering Algorithm for Static Rules



Multipass Rendering

- **Depth Peeling Front**
- **Depth Peeling Back**
- **Point Index and Classification**

Depth Peeling Front

depthPeelingFront:

while (!lockF) {
  peel the next front facing fragment f: depth(f) > depthF;
  depthF= depth(f);
  if depthF <= D_p then O_f ++;
  else lockF=true;
}

classification and point index:

indexP= O_f-O_b;

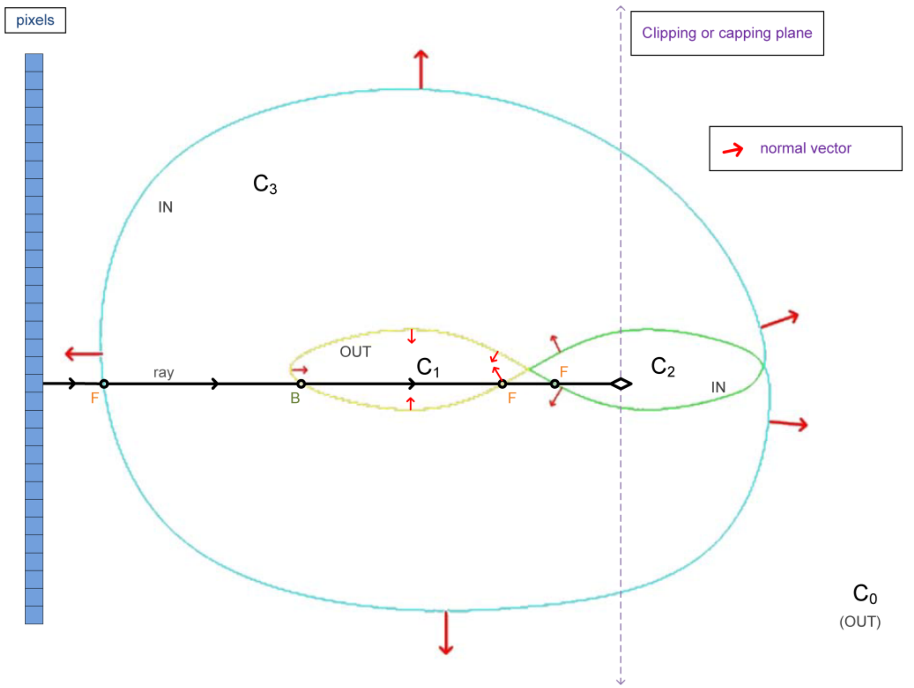
characterizationP= rule(O_f-O_b);

Rendering Algorithm for Static Rules

Two pass Rendering

Point Index Computation

Classification



Point index computation

pointIndexComputation:

ADD_blend indexP on
    for each fragment f {
        indexP:=0;
        if (depth(f) <= D_p) indexP= (F is frontFacing) ? 1 : -1;
        }

classification

characterizationP= rule(indexP);

Other Alternatives to Peeling

One pass rendering

Use FreePipe or Linked lists (2011, 2010) techniques.

FreePipe:

state of the art API and hardware,

pre allocation of large amount of memory

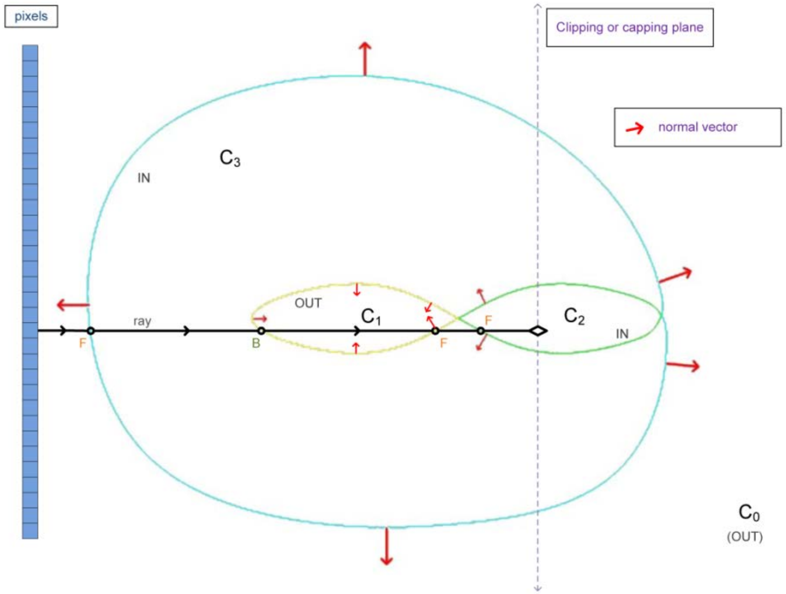
Linked lists:

state of the art API and hardware,

low memory requirements,

slowdown when creating linked lists due to memory write conflicts.

Rendering Algorithm for Static Rules



Trimming

Calculate indexP for clipping plane

Peel the first "visible" fragment

(part of the trim)

Color the corresponding pixel

depthPeeling:

while (!color) {
peel the fragment f with the smallest depth: depth(f) > currentDepth;
currentDepth= depth(F);
w_b= w_a = currentIndex;
if (f is frontFacing) w_a++; else w_a--;
currentIndex= w_a;
if (rule(w_a)!=rule(w_b)) color=color(f);
}

finalColor:

use color *for rendering the pixel;*

Rendering Algorithm for Dynamic Rules

- **Index Computation and Classification for Clipping Plane Point**
  - **Same indexP computation**
  - **Slightly different classification based also on reference frame**
    - **index and classification**

Trimming is much more complicated, we need all previous frame info (i.e. point index, classification, z-value) for all fragments.

Instead we maintain in the scene the reference (previous) frame fragments as well

We also maintain a bit vector with the in/out classifications for all regions of the reference frame per pixel (i.e. for 64 layers we just need a 64 bit vector per pixel).

Color coding to denote whether a fragment is only in the current frame, only in the reference frame, or in both. Stored in the alpha-value of RGBA and is then automatically conveyed to the fragments

Rendering Algorithm for Dynamic Rules

pointIndexComputation:

```
ADD_blend [cIndex, rIndex] on
          for each fragment f {
                    [cIndex, rIndex]=[0, 0];
                    if (depth(f)<depth(C)) {
                              w_f=(f is frontFacing) ? 1 : -1;
                              case color.α(f) {
                                        0:        [cIndex(f), rIndex(f)]=[0, w_f];
                                        ½:        [cIndex(f), rIndex(f)]=[ w_f, w_f];
                                        1:        [cIndex(f), rIndex(f)]=[ w_f, 0];
                              }
                    }
          }
return [cIndex, rIndex];
```

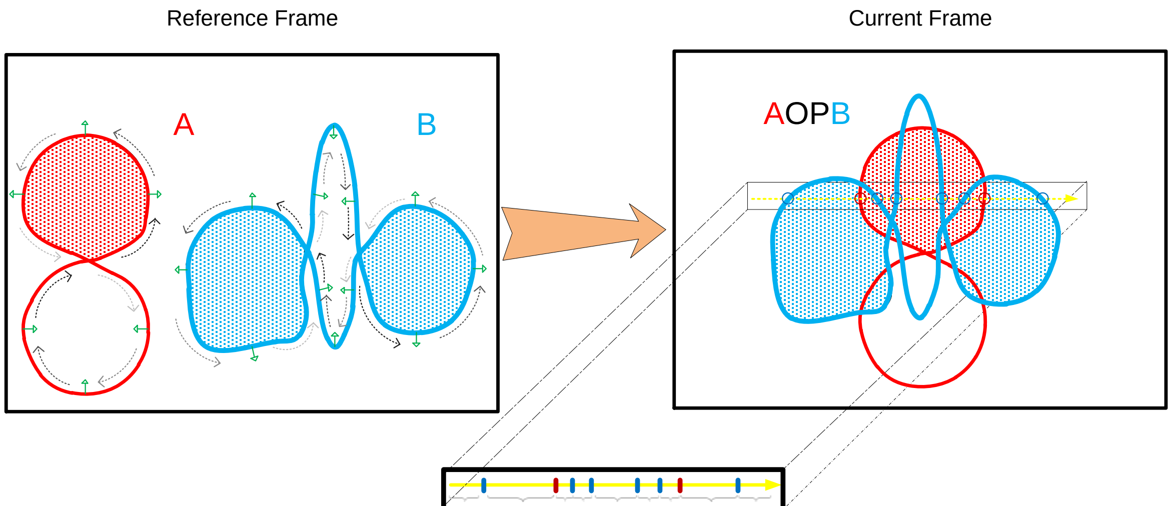
Rendering Algorithm for Dynamic Rules

depthPeeling:

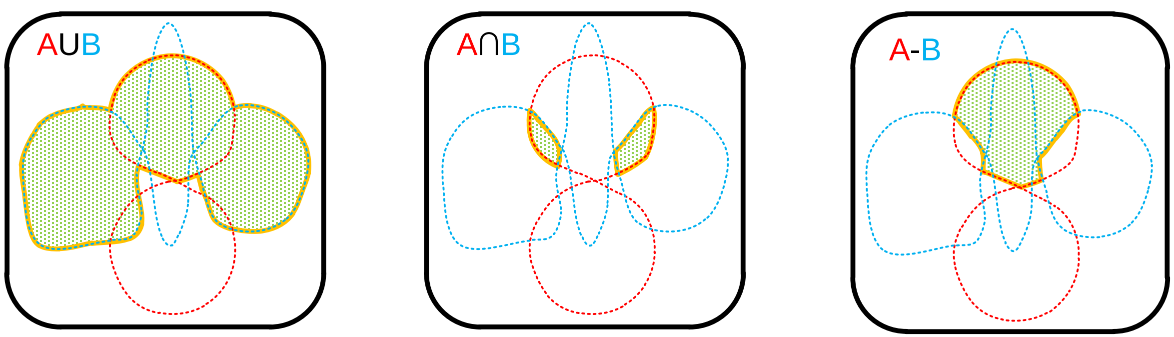
```
while (!color) {
        peel the fragment f with the smallest depth: depth(f) > currentDepth;
        currentDepth= depth(f);
        case color.α(f) {
                0:      referencePointCharacterization= RFBC[++referenceFragmentNumber];
                        if (f is frontFacing) referenceIndex++; else referenceIndex --;
                ½:      referencePointCharacterization= RFBC[++referenceFragmentNumber];
                        if (f is frontFacing) referenceIndex++; else referenceIndex --;
                        if (f is frontFacing) currentIndex ++; else currentIndex --;
                        I_b= currentPointCharacterization;
                        I_a=dynamic_rule(currentIndex, referenceIndex,
                                    referencePointCharacterization);
                        currentPointCharacterization= I_a;
                        if (I_a != I_b) color=color(f);
                1:      if (f is frontFacing) currentIndex++; else currentIndex --;
                        I_b= currentPointCharacterization;
                        I_a=dynamic_rule(currentIndex, referenceIndex,
                                    referencePointCharacterization);
                        currentPointCharacterization= I_a;
                        if (I_a != I_b) color=color(f);
        }
}
```

Rendering CSG results

Use a variation of the rendering algorithm for the trim of dynamic rules

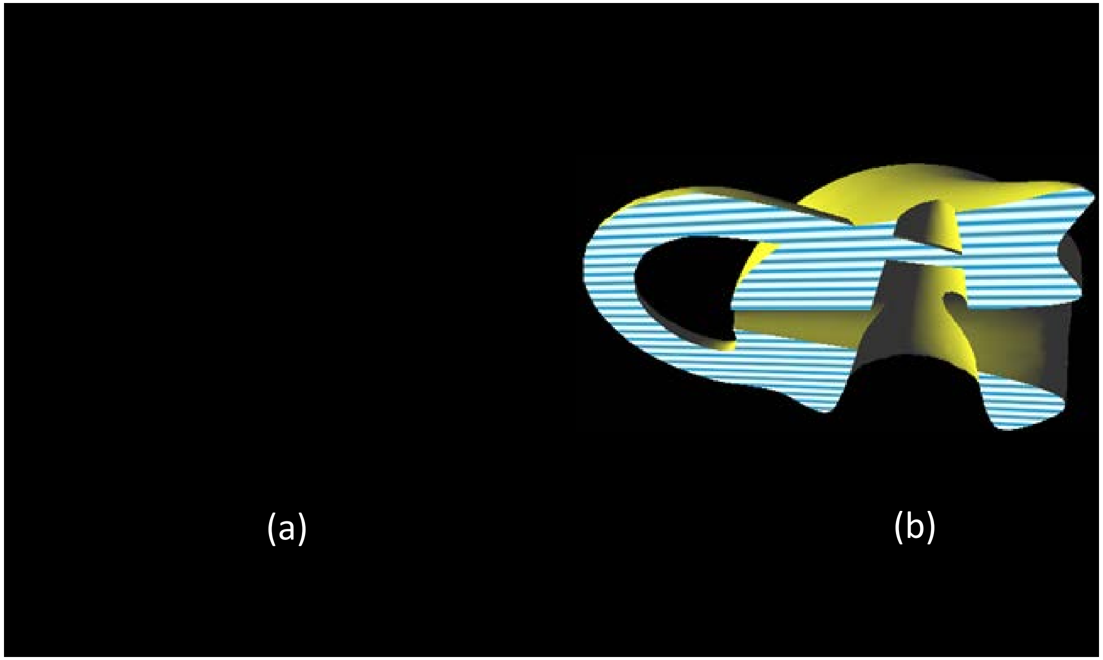


A	0	0	1	1	1	1	1	0	0
B	0	1	1	0	0	0	1	1	0
AUB	0	1	1	1	1	1	1	1	0
A∩B	0	0	1	0	0	0	1	0	0
A-B	0	0	0	1	1	1	0	0	0



Capping and LOD

Create a halfspace using the capping plane and subtract this from the object.

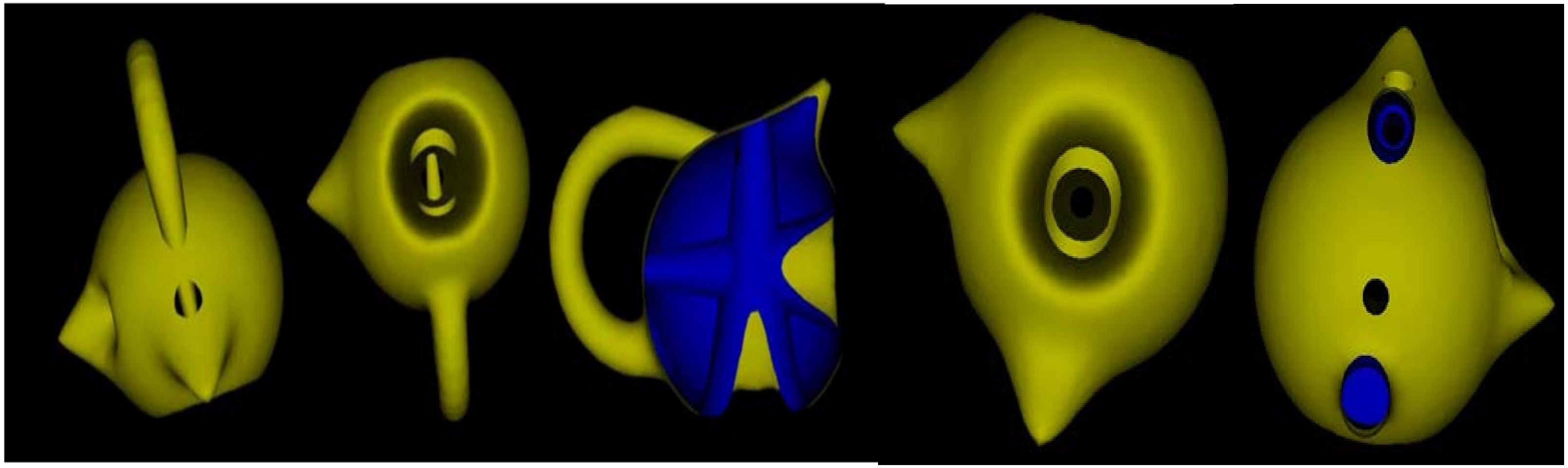


Treat change of LOD as deformation. Alternatively LOD can change in the initial surface S_0 and propagated to the rest of the sequence.

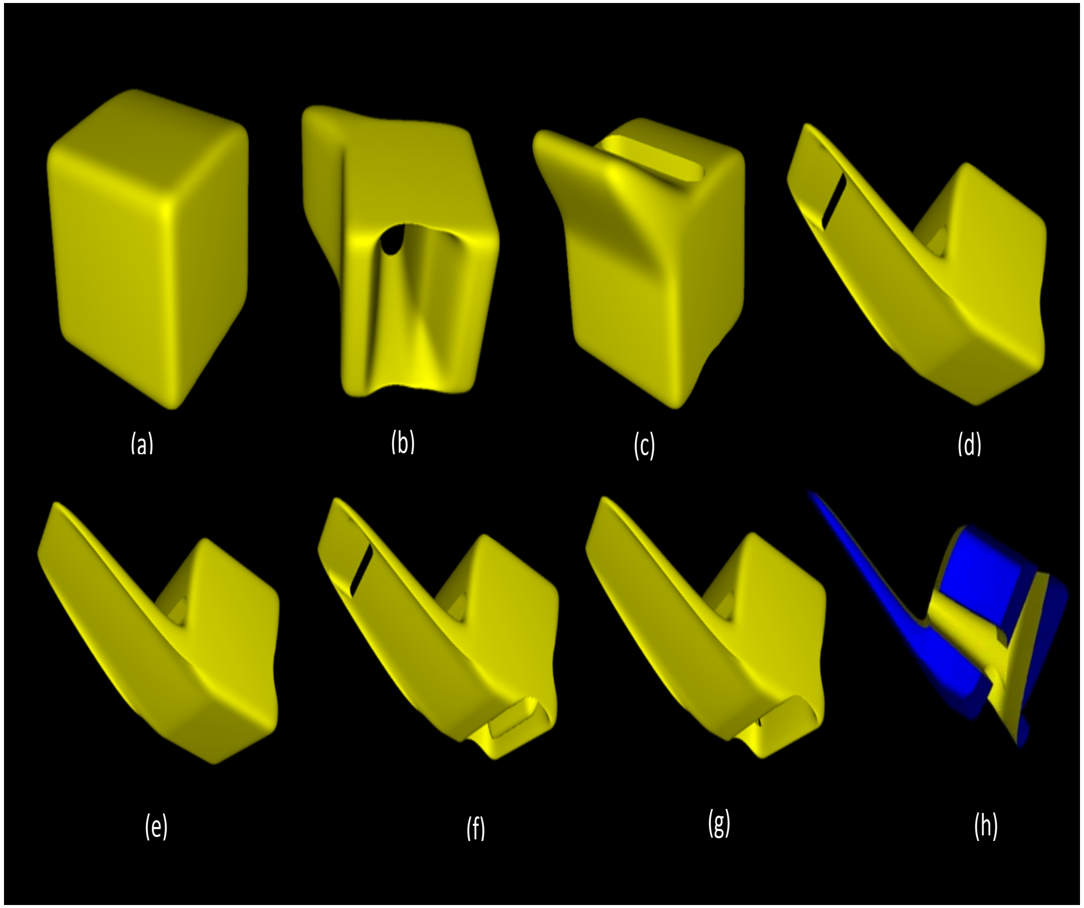
Results

Local influence deformations in conjunction with Laplacian smoothing OR control point movement of NURB surfaces combined with mesh subdivision.

All experiments were carried out on a commodity desktop with Intel Core i7-870@2.93GHz, 4GB DDR3 memory and NVIDIA GeForce GTX 480 graphics hardware using Opengl and GLSL

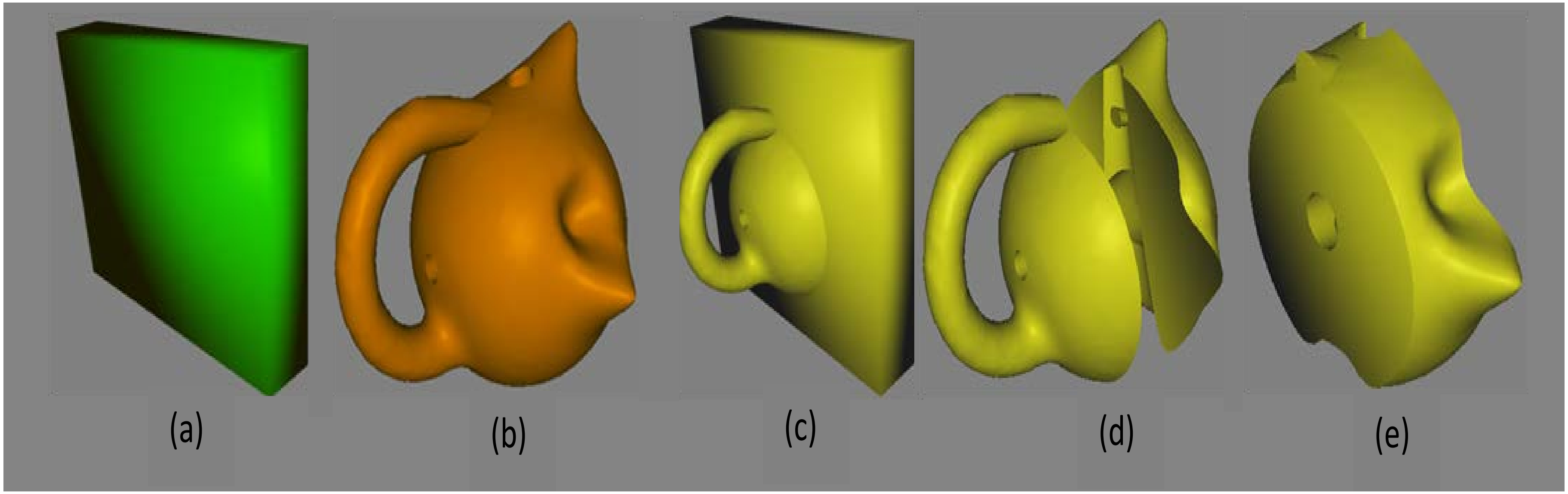


Results



(a) The original NURB surface. (b and c) After applying a set of disjoint deformations. (d) Rendering the result of one more deformation with the static rule, where the upper extrusion is deformed so as to cross an extended hole (observe the unintuitive hole in the upper part) and (e) the same using the dynamic rule (no hole is present in the upper part). (f) Rendering the trim with the static rule after one more hole is created at the bottom, observe the unintuitive bump at the bottom and (g) the same with the dynamic rule (no bump is present). (h) g with clipping.

Results



. *(a) Object A. (b) Object B. (c) Rendering A U B , (d) B – A and (e) A ∩ B*

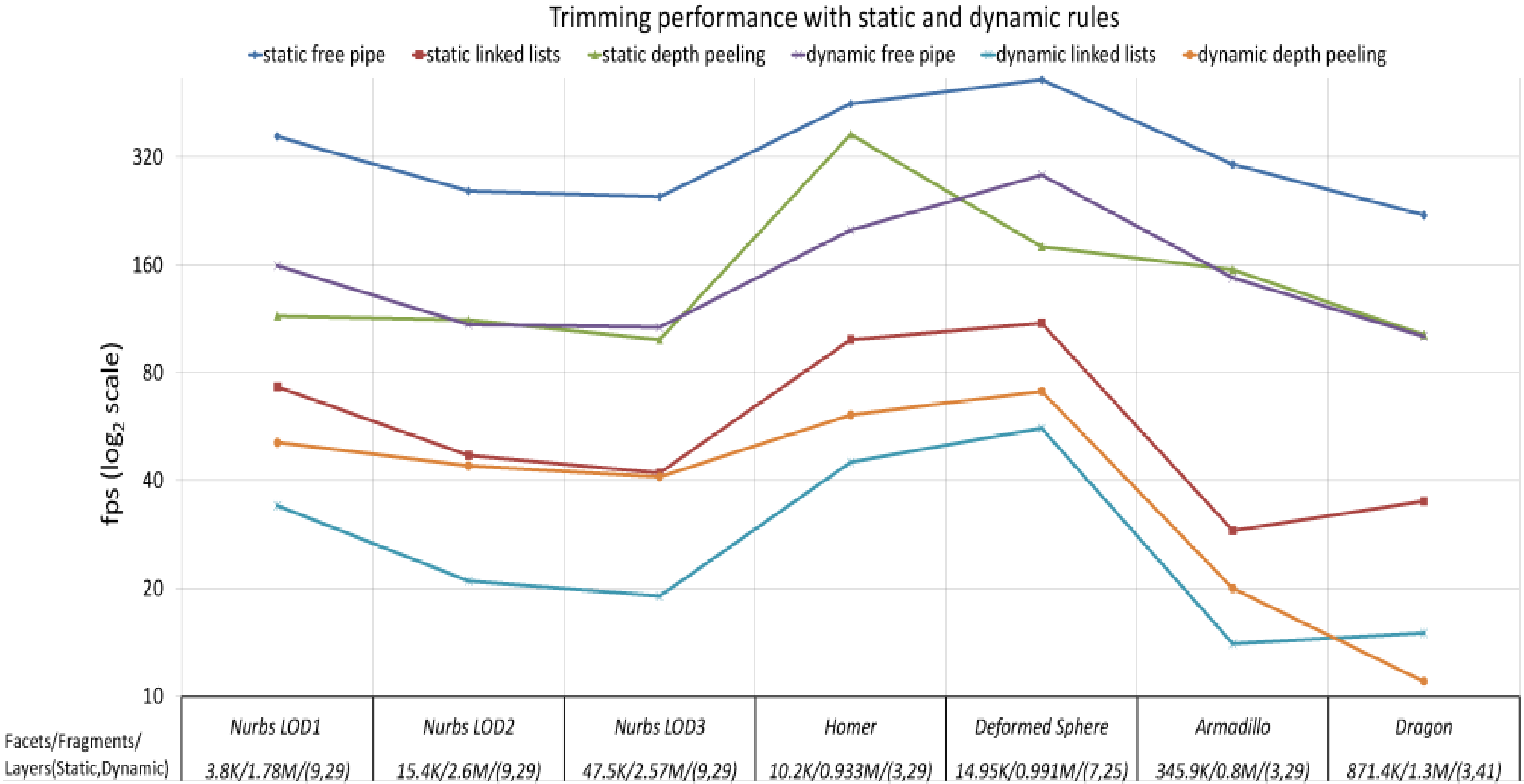
Results

Resolution 1024x768				Static Rules																							Dynamic			Standard	
Model				F2B							Dual							2 passes			FreePipe			Linked Lists			2 passes			without	
Name	Size			Best		Average		Worst			Best		Average		Worst			All			All			All			All			All	
	Vertices	Faces	Fragments	Fps	Passes	Fps	Passes	Fps	Passes	MB	Fps	Passes	Fps	Passes	Fps	Passes	MB	Fps	Passes	MB	Fps	Passes	MB	Fps	Passes	MB	Fps	Passes	MB	Fps	Passes
Homer	5103	10202	0.991M	510	2	212	7	121	16	21,75	490	1	203	5	113	9	31,5	940	2	7,5	575	1	90	111	1	20,34	895	2	9,75	1355	1
Sphere	7478	14952	0.897M	500		263	6	180	10		465		240	4	162	6		1055			620		54	123		19,26	990			1518	
Deformed sphere	7478	14952	0.933M	515		219	7	135	14		485		194	5	125	8		965			608		78	115		19,67	905			1465	
Deformed Nurbs	23807	47557	2.57M	330		149	7	88	14		300		120	5	76	8		520			312		78	47		38,4	494			824	
Armadillo	172974	345944	0.8M	213		51	9	31	16		245		73	6	51	9		273			350		90	30		18,2	265			461	
Dragon	437645	871414	1.3M	121		23	12	12	22		162		38	7	22	12		198			258		126	38		23,83	196			286	

FPS, number of passes and memory needed using the static rule without trimming.

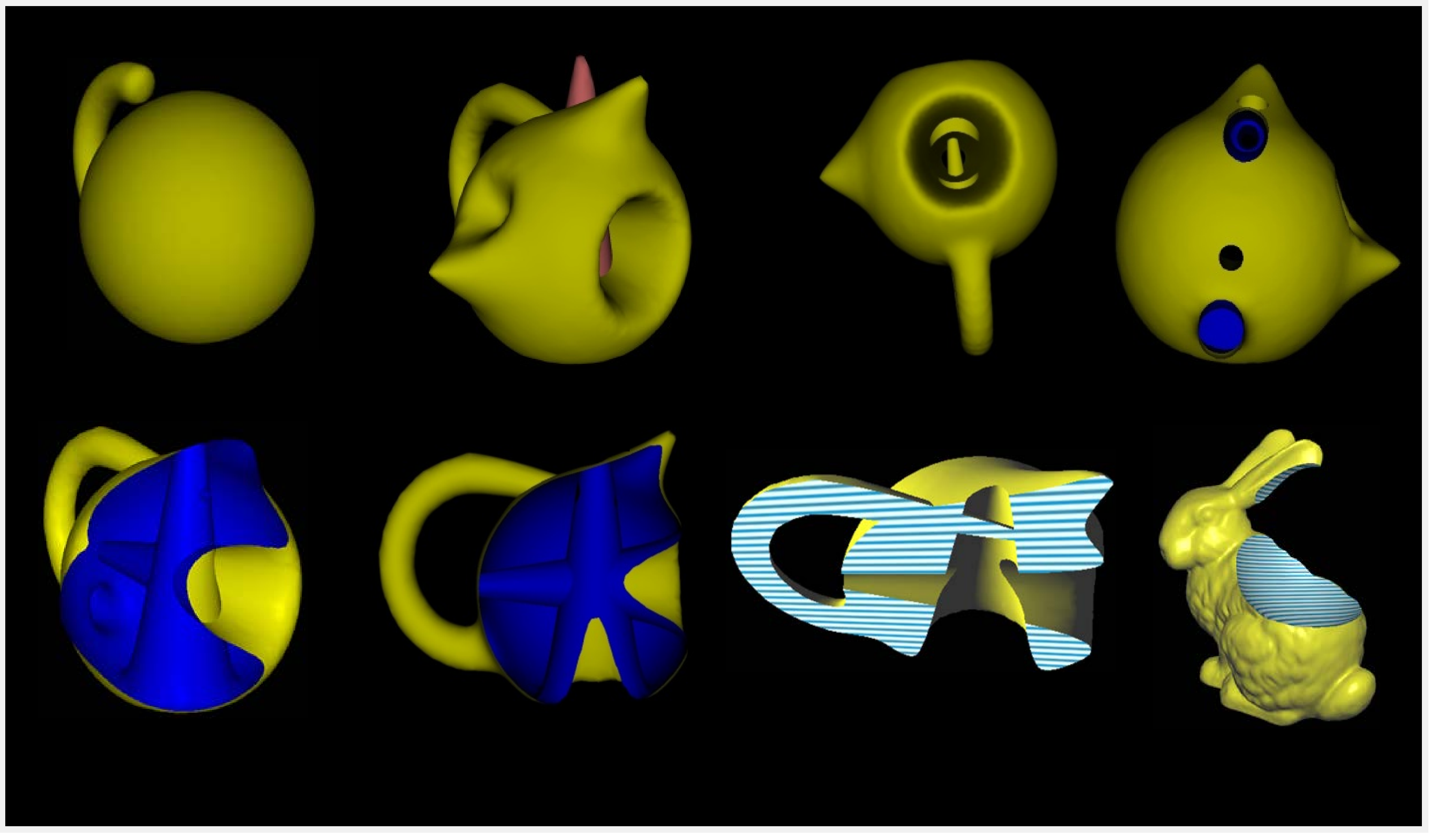
For dynamic rules without trimming there is a small overhead for all techniques in terms of FPS, here we present results of the 2 passes technique that attains the best results. The rightmost column corresponds to standard rendering without in/out classification.

Results



for rendering a trimmed SCS using the dynamic and the static rule. With multipass depth peeling, freepipe and linked lists.

Results



Video: deforming jug

Video: deforming NURBS

Thank you!

More at:

http://www.cs.uoi.gr/~fudos/trimming.html

Questions ?