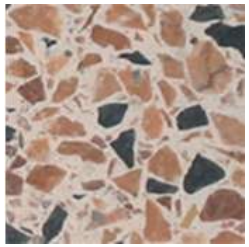
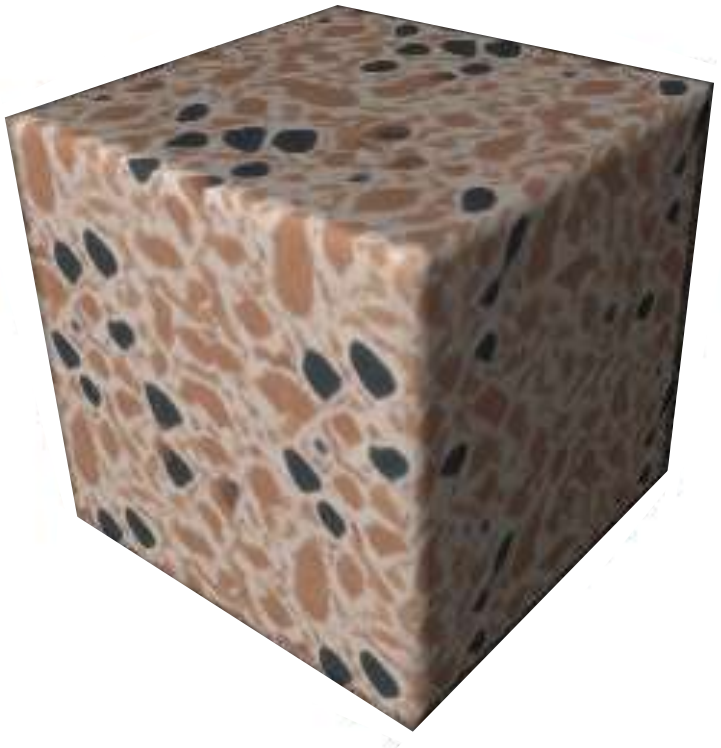
Lazy Solid Texture Synthesis

Yue Dong [1,2] Sylvain Lefebvre [3] Xin Tong [1] George Drettakis [3]

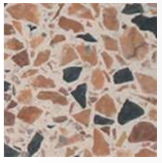
[1] Microsoft Research Asia
[2] Tsinghua University
[3] REVES / INRIA Sophia-Antipolis

Solid textures from 2D exemplars

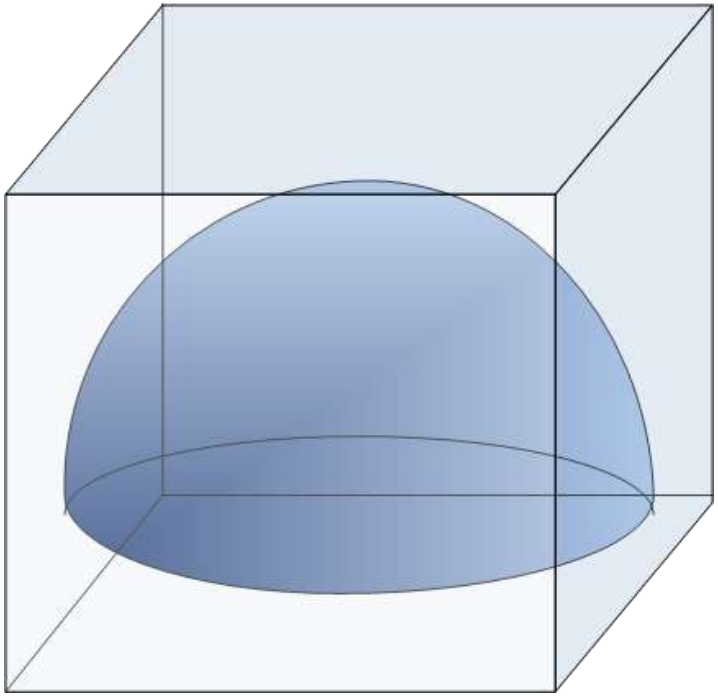




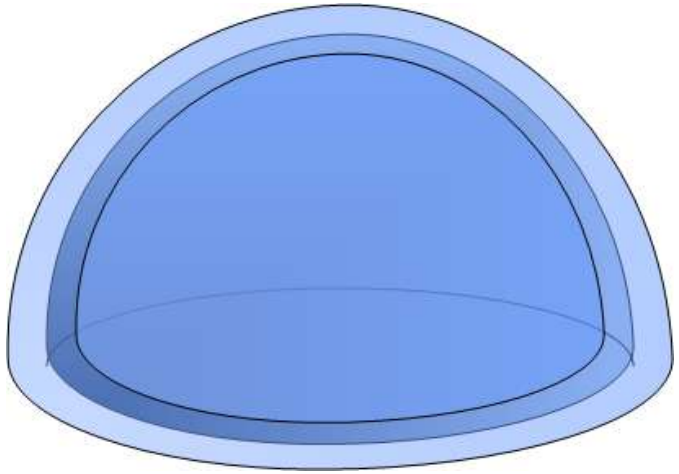
Solid textures for surfaces



Key motivation



Full volume

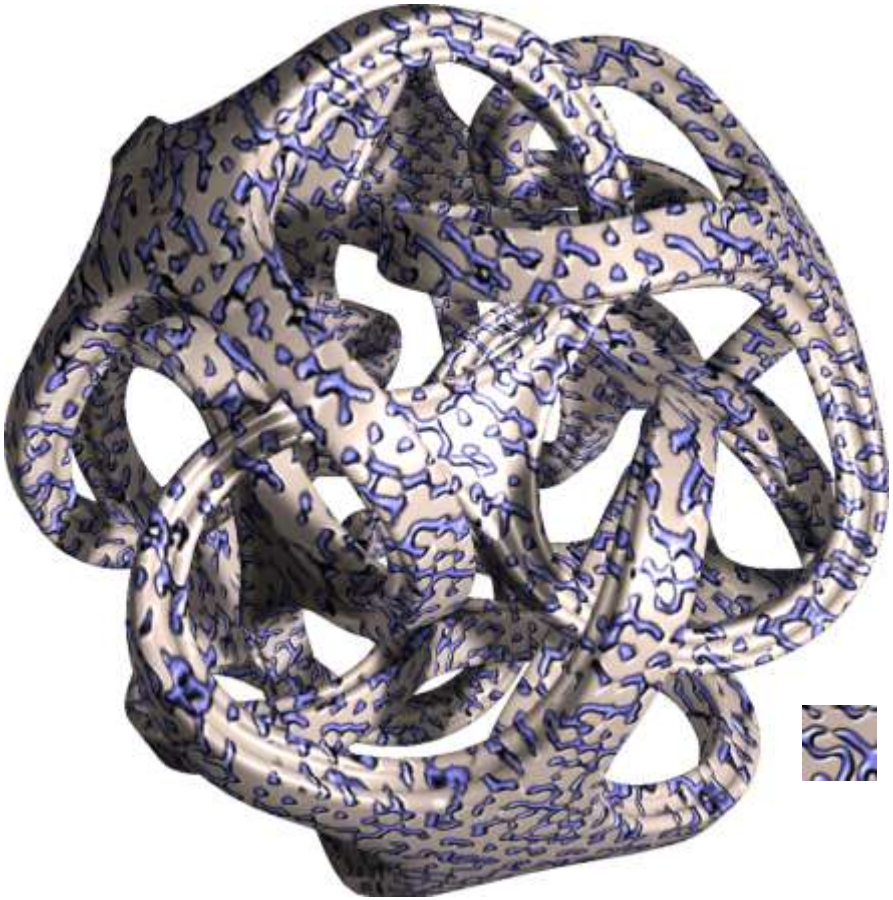


Only around the surface!

Advantages

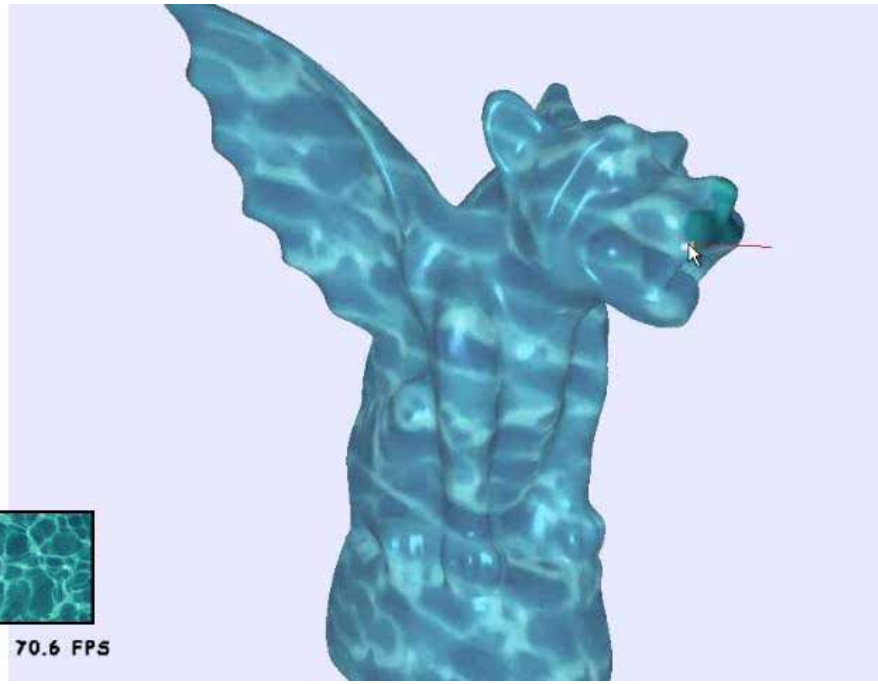
- Less memory, less processing
  - Depends on textured *area* only
- Equivalent to solid synthesis
  - Spatially deterministic

1024^3 (**3 GB**) 54 MB 17 sec

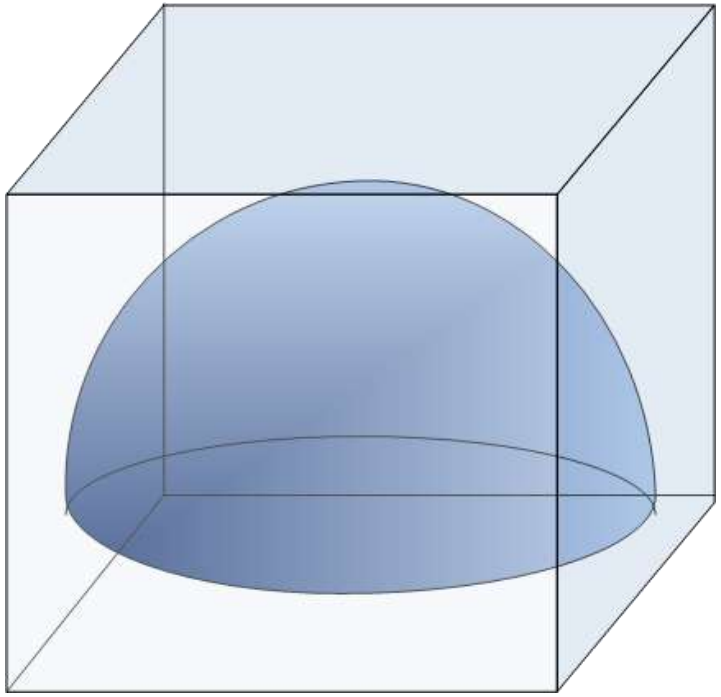


Advantages

- Less memory, less processing
  - Depends on textured *area* only
- Equivalent to solid synthesis
  - Spatially deterministic
- GPU implementation
  - On demand synthesis
  - No repetitions (**not** a tiling!!)



Previous work *Implicit volume*



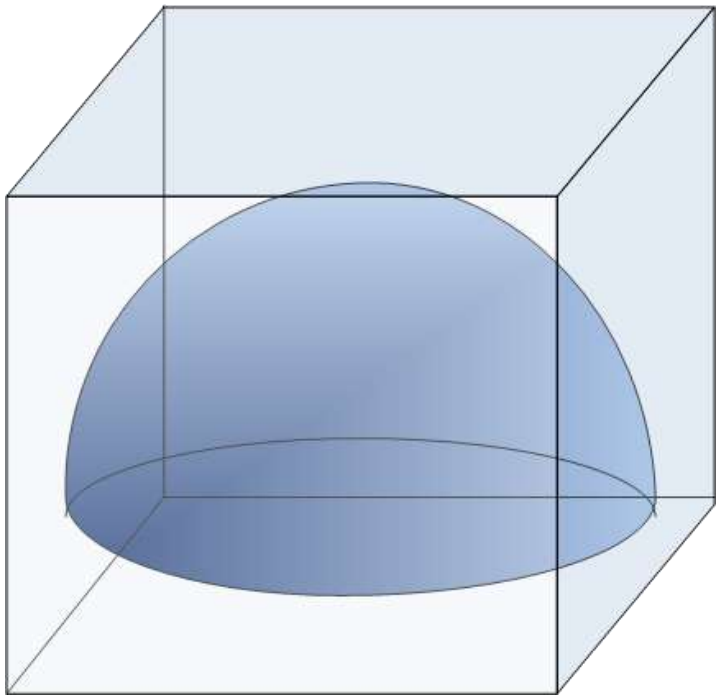
color = **f(**x,y,z**)**

- Procedural texturing
  - [EMP*94]
- Spectral analysis
  - [GD95] [GD96]

Low memory

Limited range of materials

Previous work *Explicit volume*



color = **grid**[x][y][z]

- Histogram matching
  - [HB95]
- Stereological techniques
  - [JDR04]
- Neighborhood matching
  - [Wei02], [QY07], [KFCO*07]

Good quality

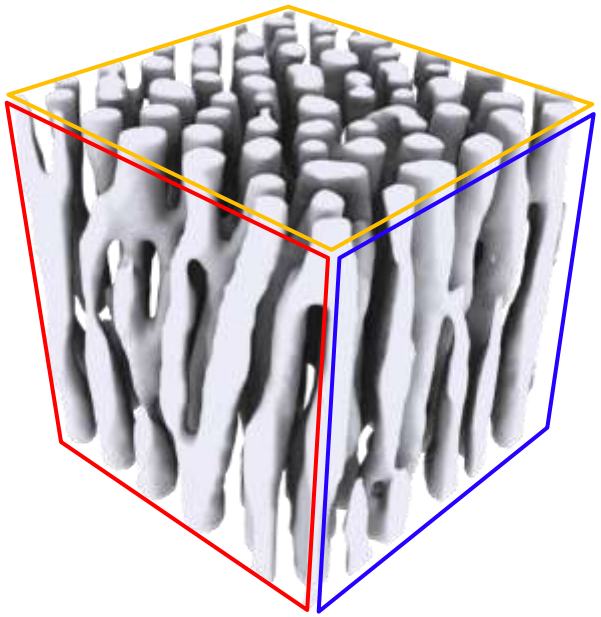
Larger range of materials

Huge, long to compute

Solid synthesis from 2D exemplars

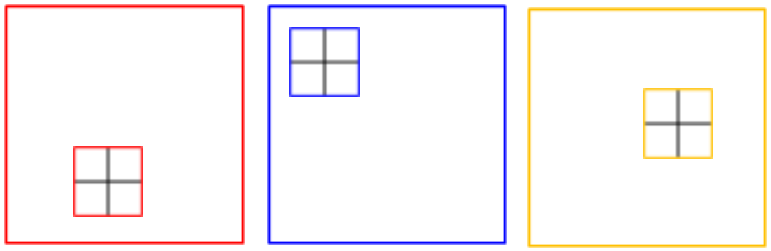
- Three 2D example images
  - Each define content along two major directions

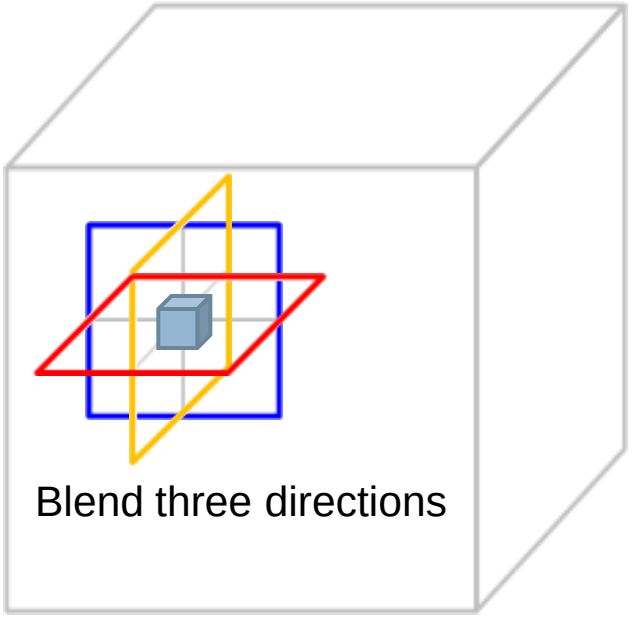
2D exemplars
(input)



Solid synthesis from 2D exemplars

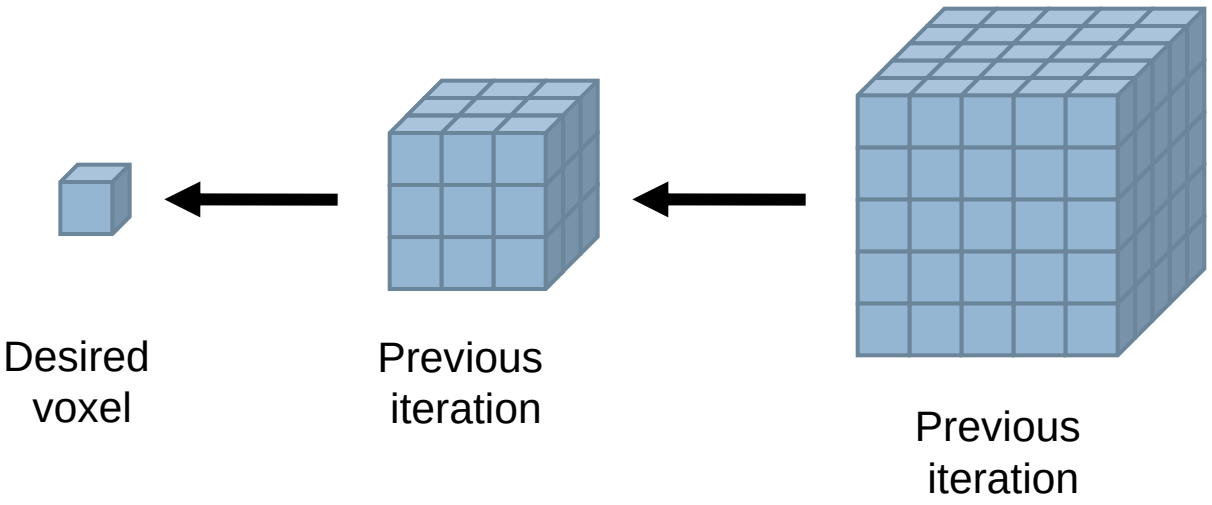
- At each voxel, three interleaved 2D problems
  - Try to match 2D neighborhoods in each direction
  - Express voxel color as a blend





How much work for one voxel?

- Global optimization [QY07], [KFCO*07]
  - ➔ All other voxels !
- Iterative neighborhood matching [Wei02]
  - ➔ More neighbors at every iteration
  - ➔ Quickly requires the full volume

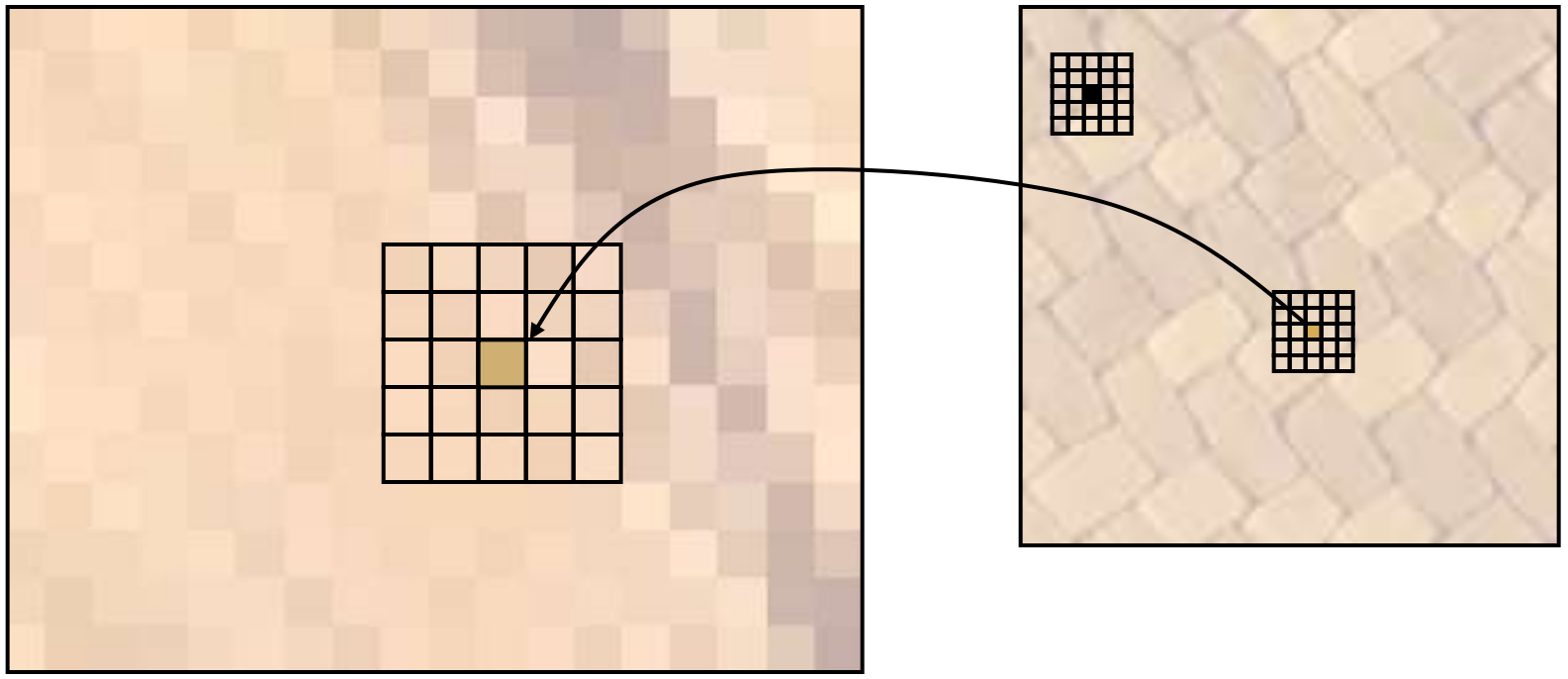


Contributions / overview

- 3D candidate pre-computation
  - Reduces dependency chain
- Parallel, spatially deterministic solid synthesis
  - Focus synthesis around surface
- Results

Why does it work better in 2D?

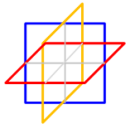
- Brief reminder



What happens in 3D?

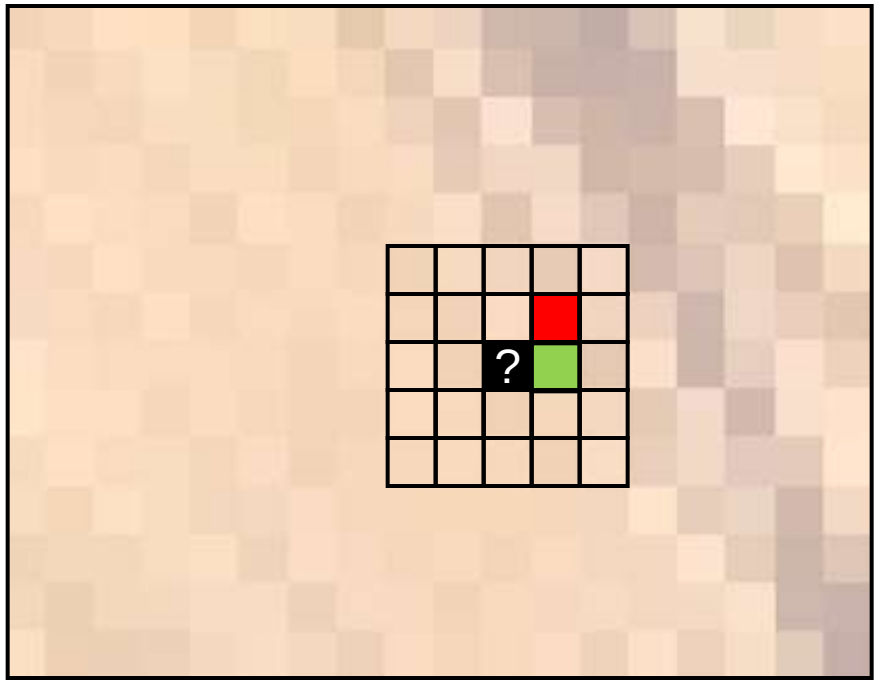
1. Colors always come from the exemplar

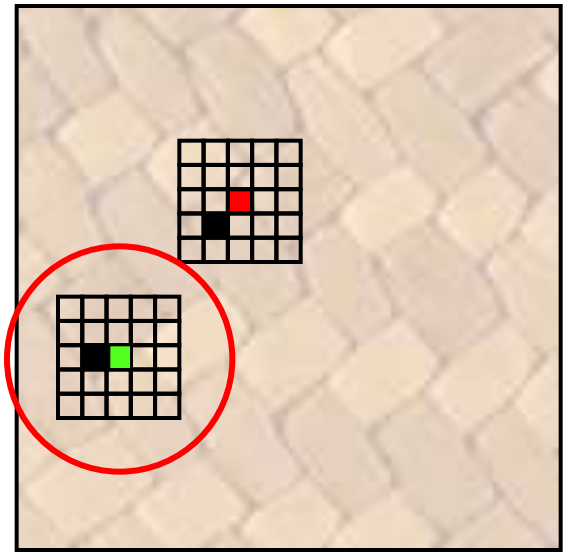
➔ In 3D, blending introduces wrong colors



Coherent candidates

- [Ashikhmin2001, Hertzmann2001, Tong *et al.* 2002]



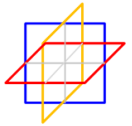


Exemplar

What happens in 3D?

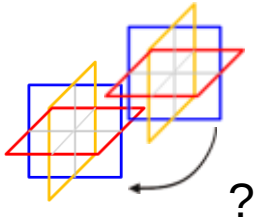
1. Colors always come from the exemplar

➔ In 3D, blending introduces wrong colors



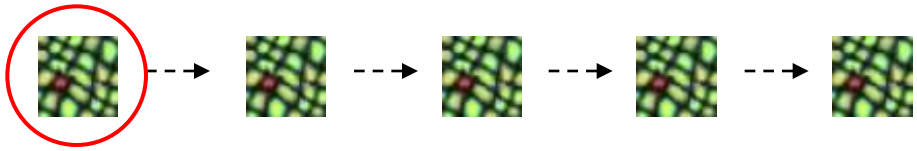
2. Candidates are likely to form patches

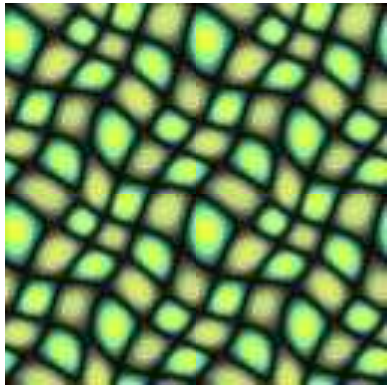
➔ In 3D, coherence in all directions is unlikely



Initialization

- Iterative neighborhood matching
  - Each iteration improves result
  - Good initialization is key to quality

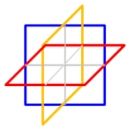




What happens in 3D?

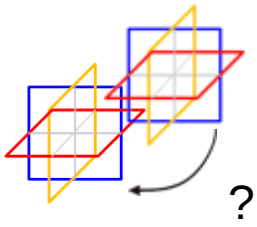
1. Colors always come from the exemplar

➔ In 3D, blending introduces wrong colors



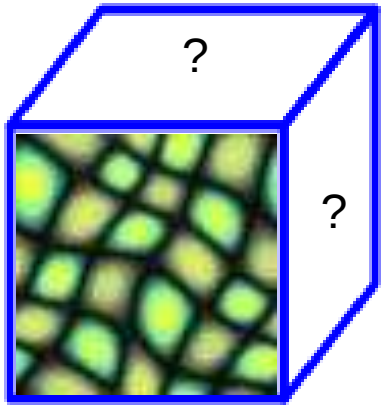
2. Candidates are likely to form patches

➔ In 3D, coherence in all directions is unlikely

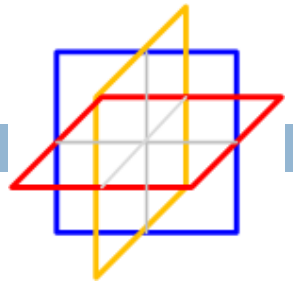


3. Tiling provides a good initialization

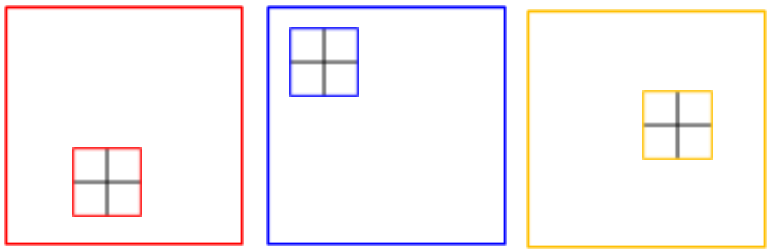
➔ In 3D, no obvious initialization from 2D

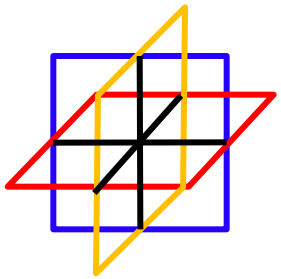


3D candidates



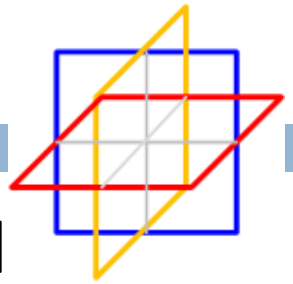
- Definition:
  - Triple of 2D coordinates



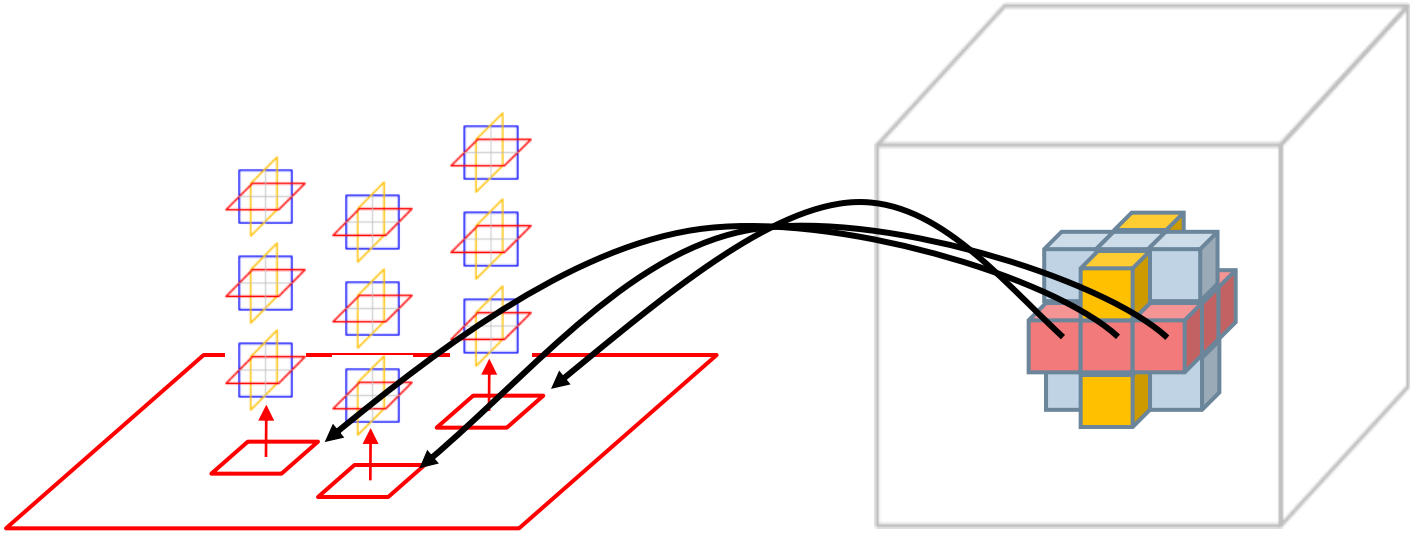


- Pre-computed to:
  - Enforce color consistency
  - Encourage coherence in all directions

3D candidates

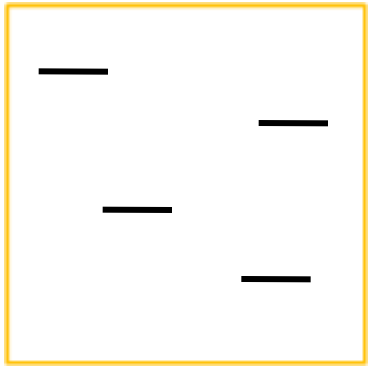


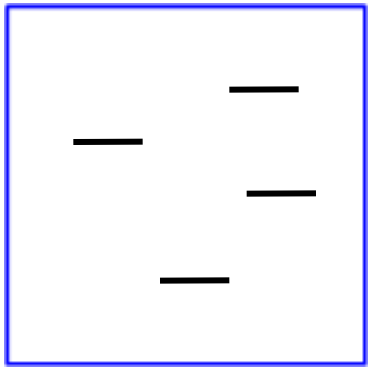
- Pre-computed triples in **each** exemplar **pixel**
  - Candidates using pixel neighborhood
- During synthesis
  - Each voxel stores a 3D candidate

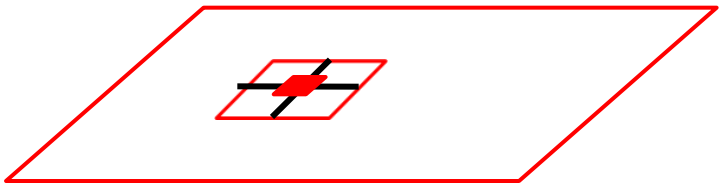


Choose best in:  ...

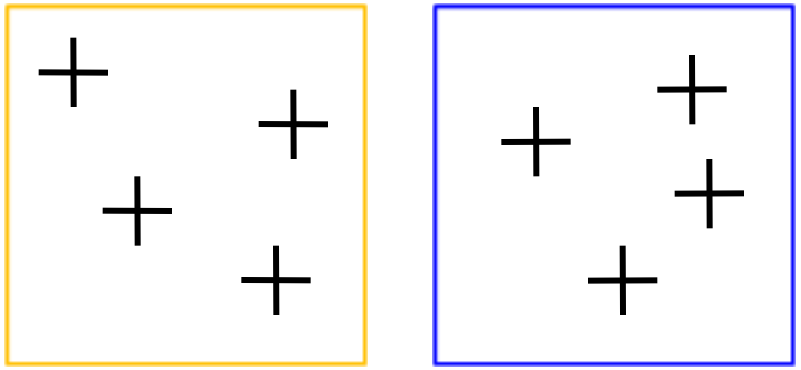
1. Color Consistency

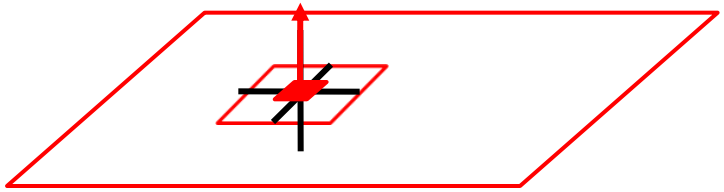




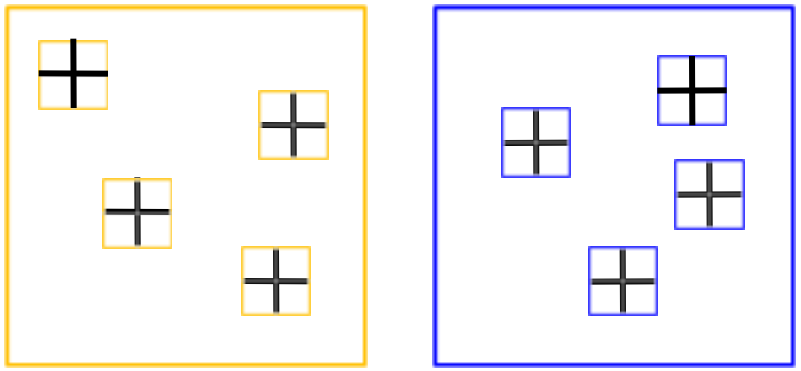


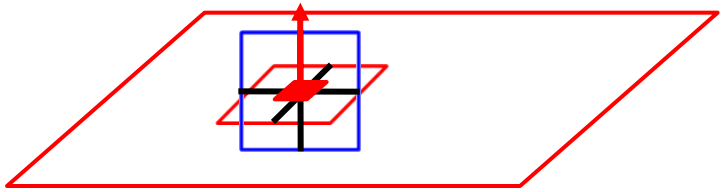
1. Color Consistency



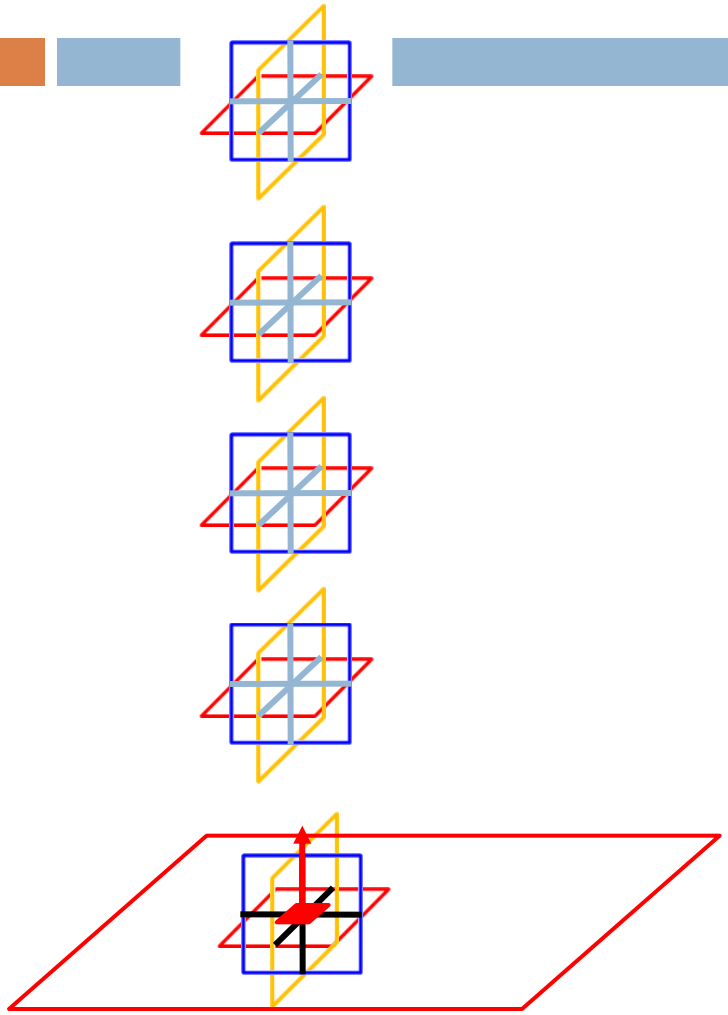


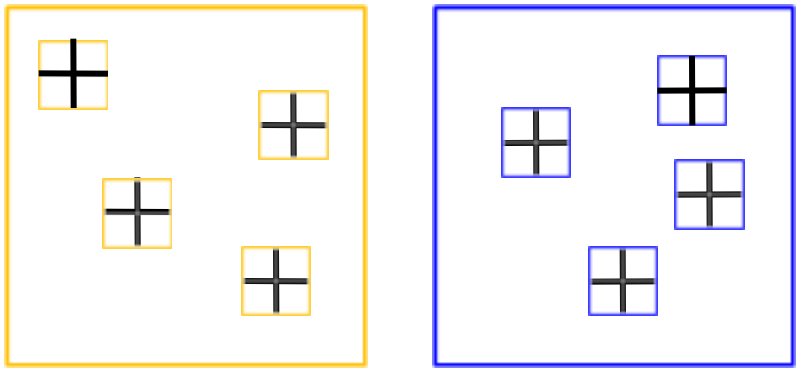
1. Color Consistency



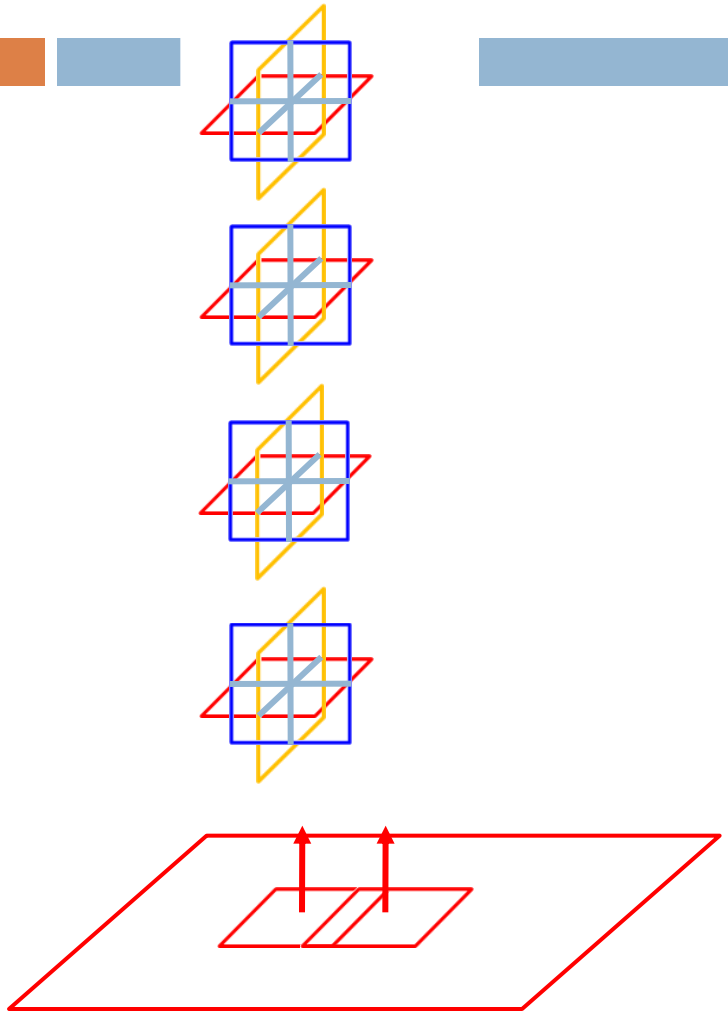


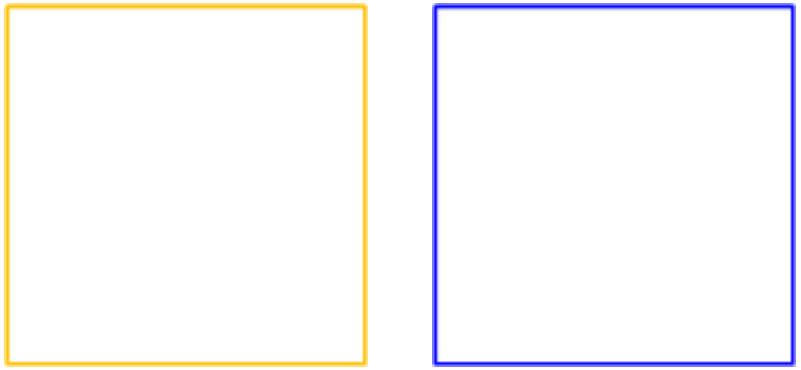
1. Color Consistency



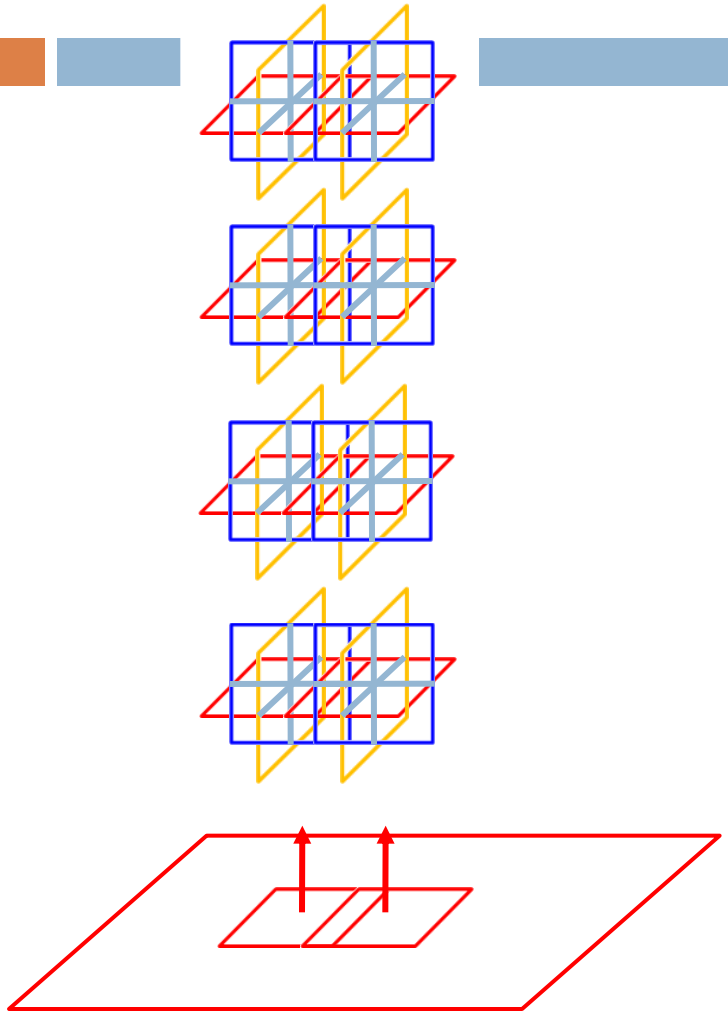


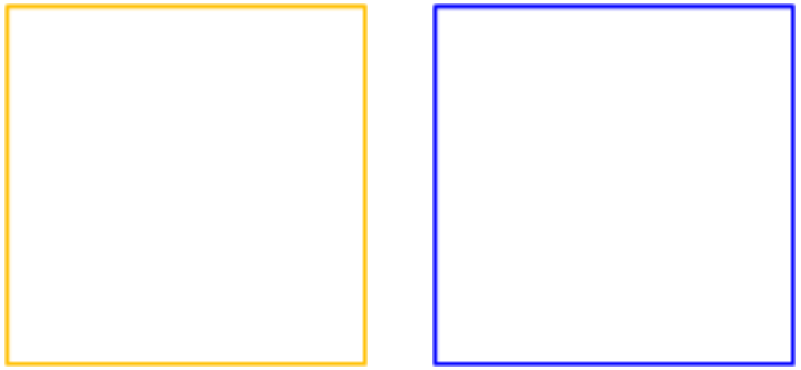
2. Coherent candidate selection



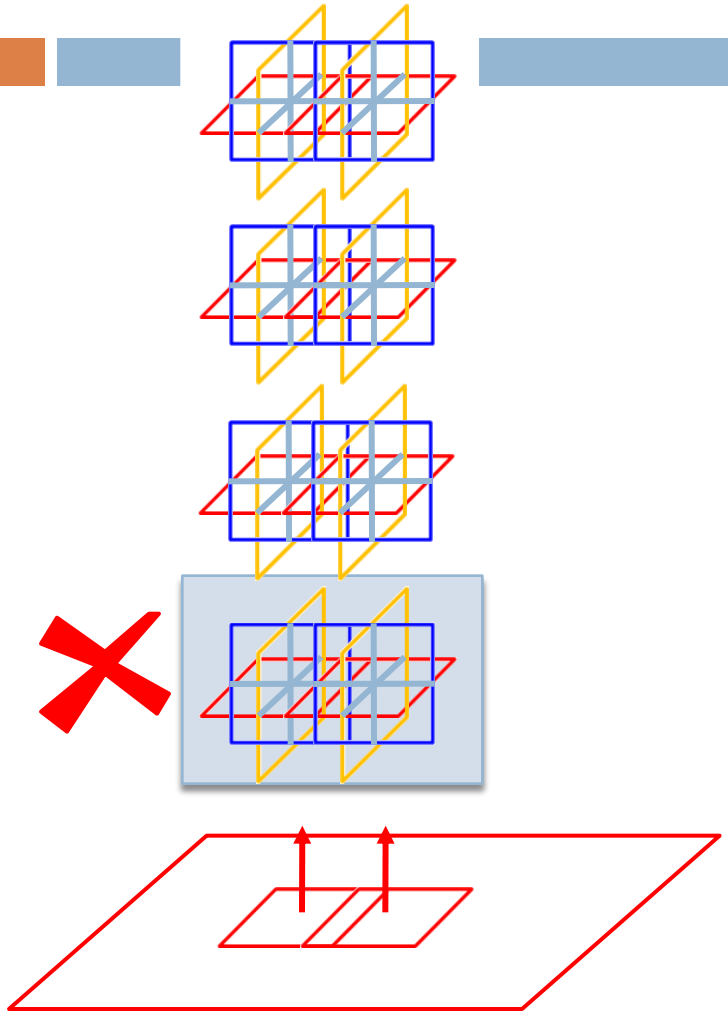


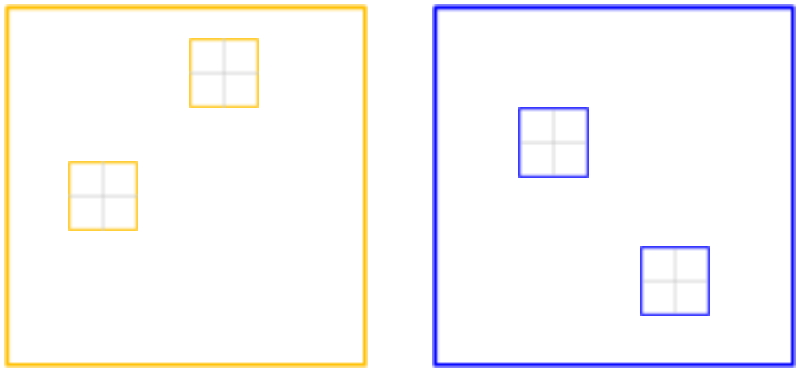
2. Coherent candidate selection



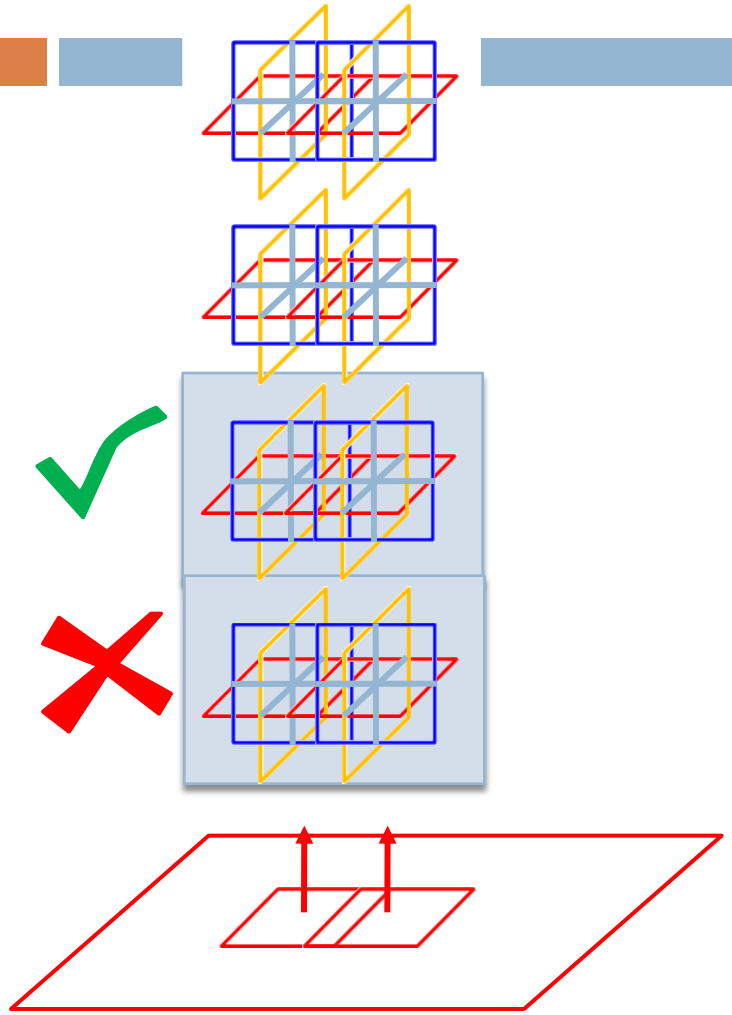


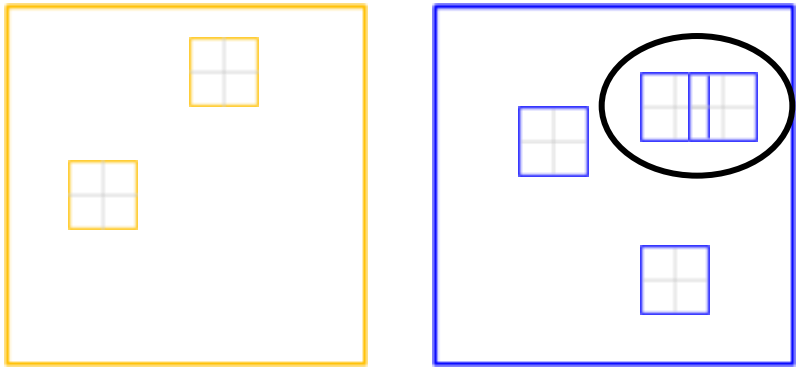
2. Coherent candidate selection



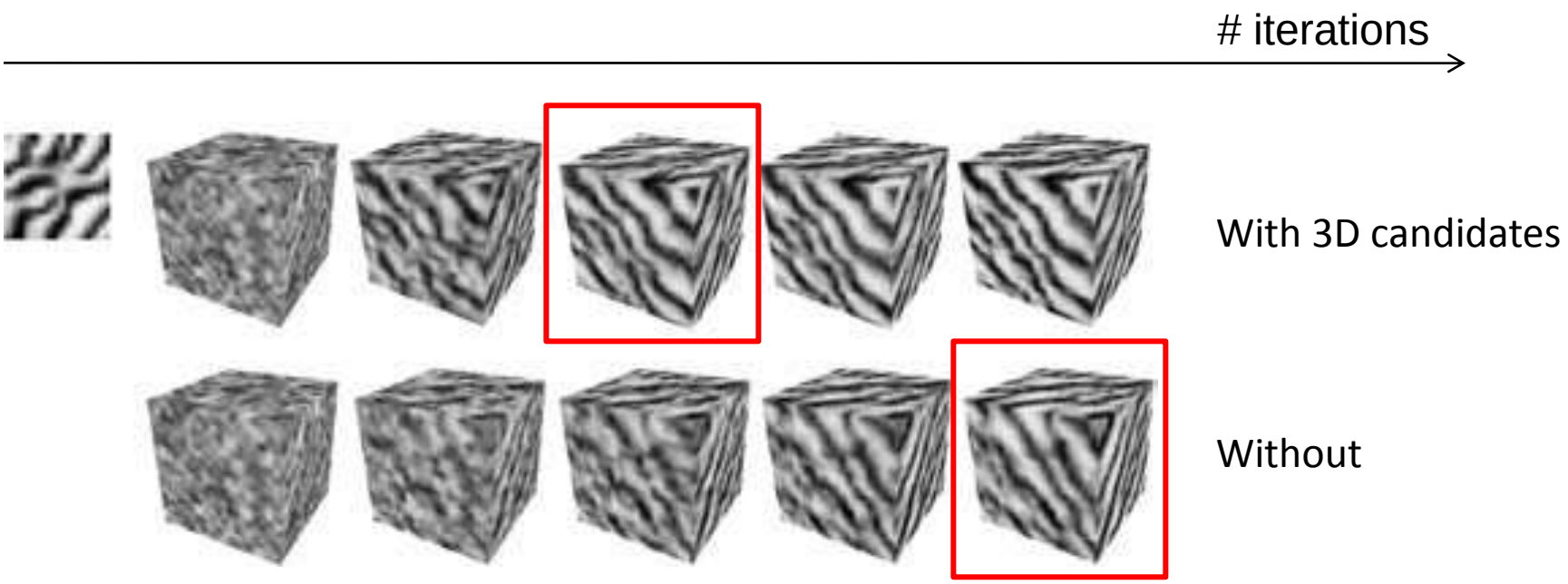


2. Coherent candidate selection





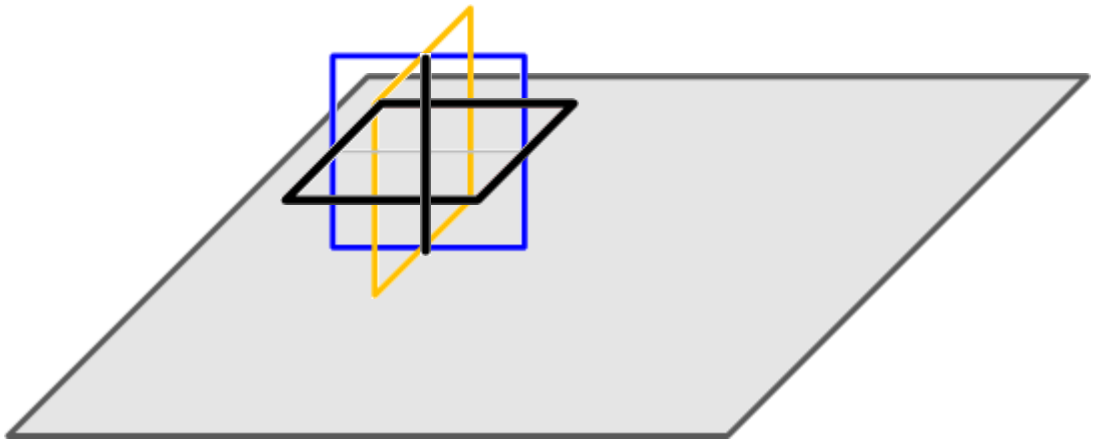
Fewer iterations



- 1 voxel dependency chain: 77K to 6K voxels

Candidate Slabs for initialization

- Thicken the exemplar using candidates
  - Used for initialization of the synthesis process
  - **Not** used during synthesis

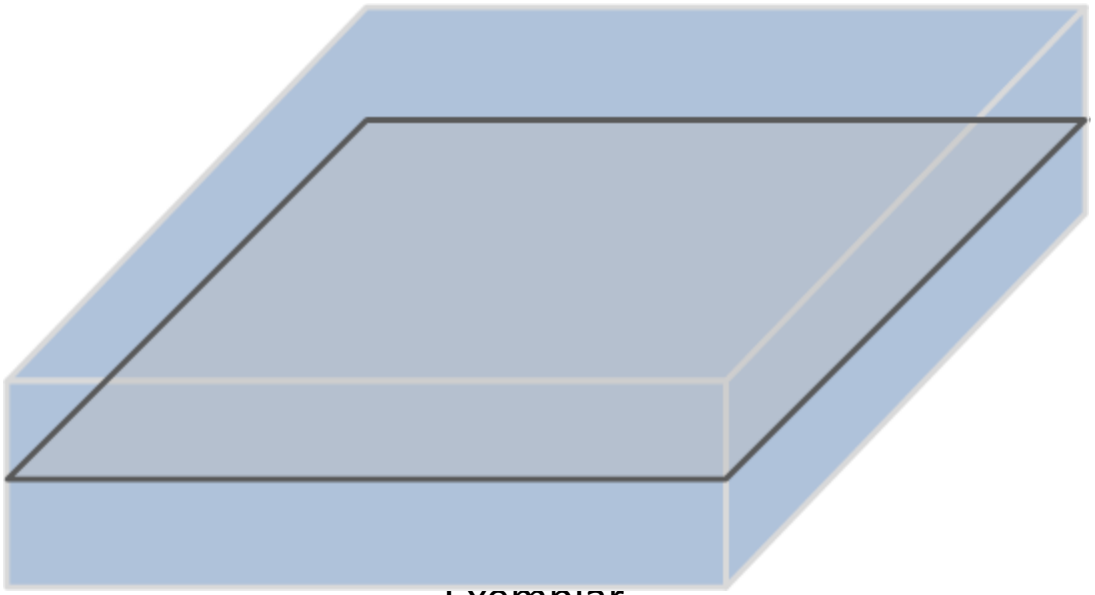


Exemplar

Candidate slab

Candidate Slabs for initialization

- Thicken the exemplar using candidates
  - Used for initialization of the synthesis process
  - **Not** used during synthesis



Exemplar

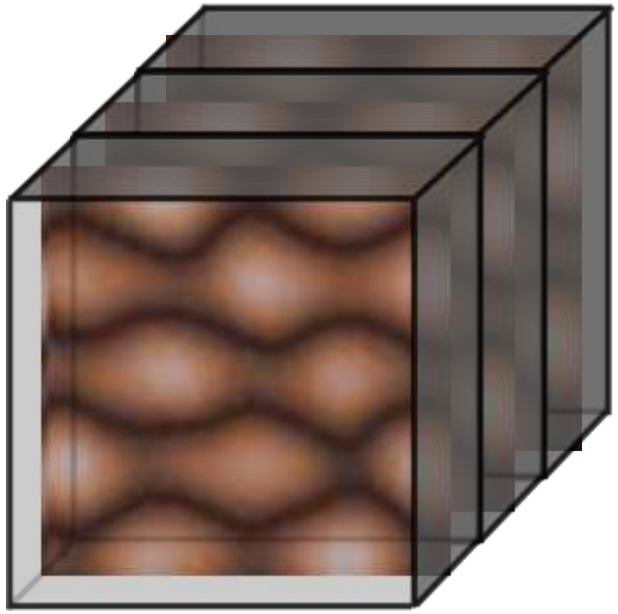
Candidate slab

Candidate Slabs for initialization

- Thicken the exemplar using candidates
  - Used for initialization of the synthesis process
  - **Not** used during synthesis

Initialization

- Tiled candidate slab

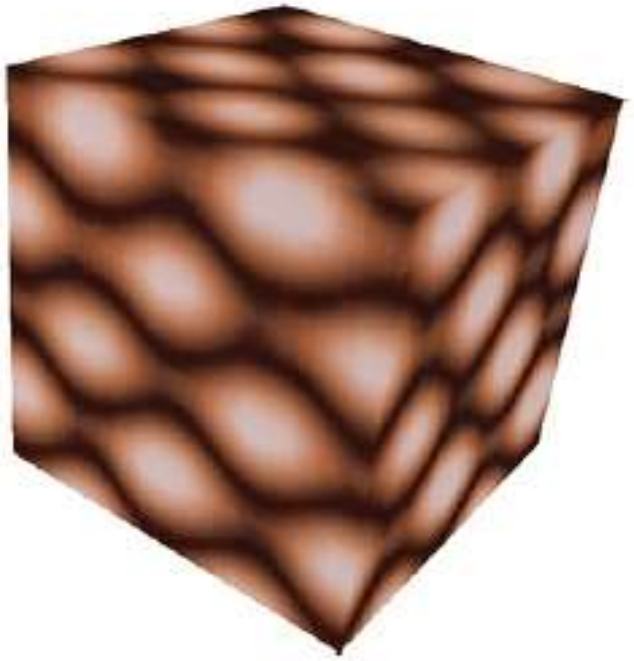


Synthesis result using
our initialization

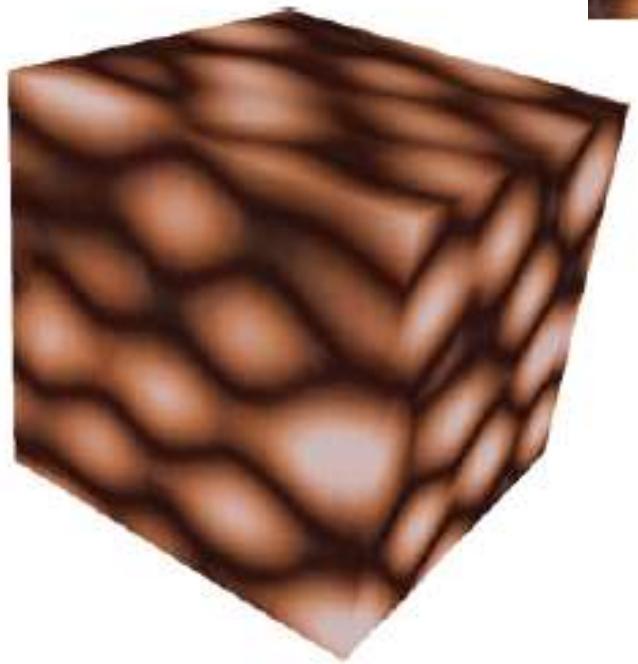
Synthesis result using
random initialization

Initialization

- Tiled candidate slab

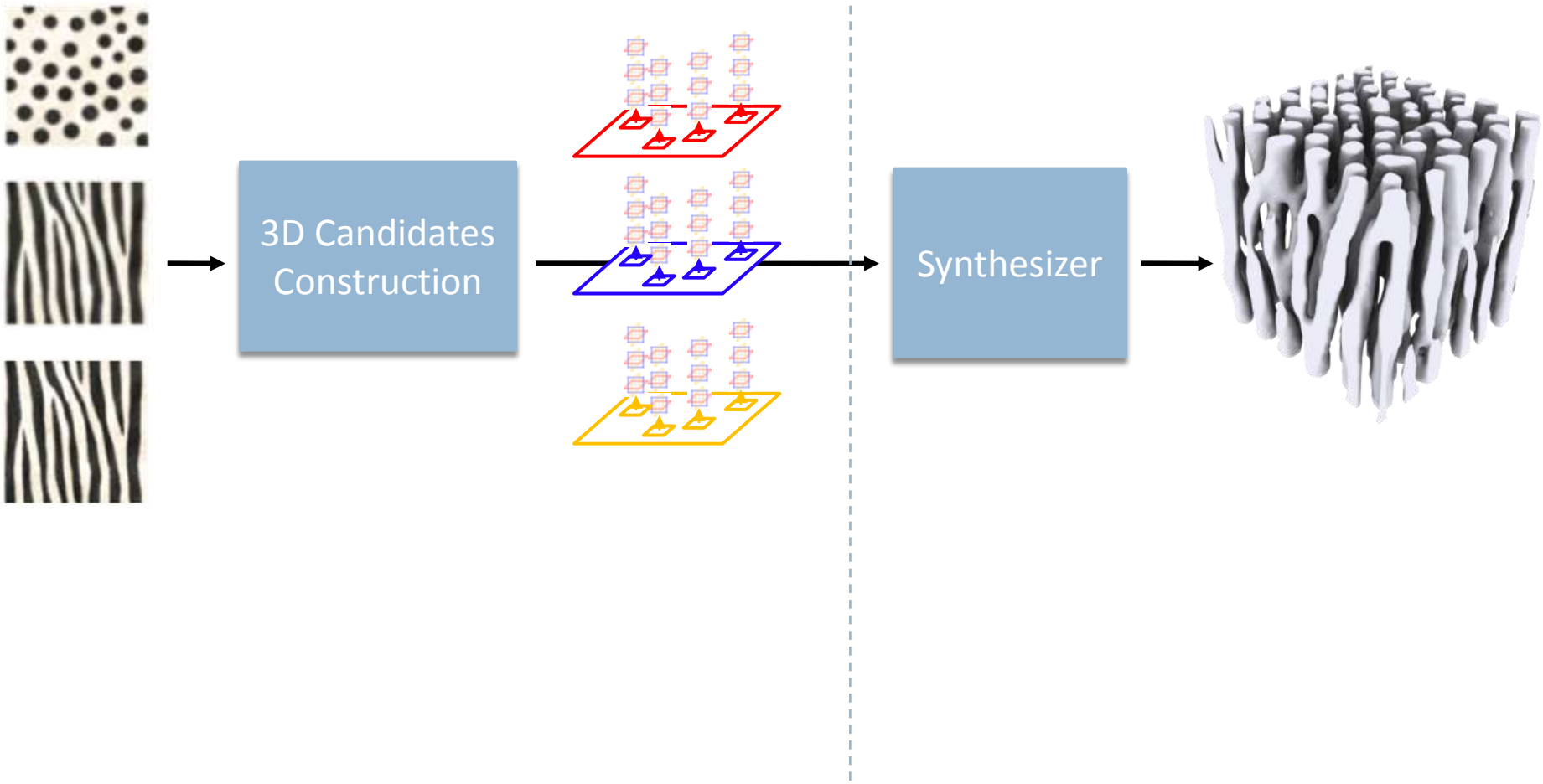


Synthesis result using
our initialization



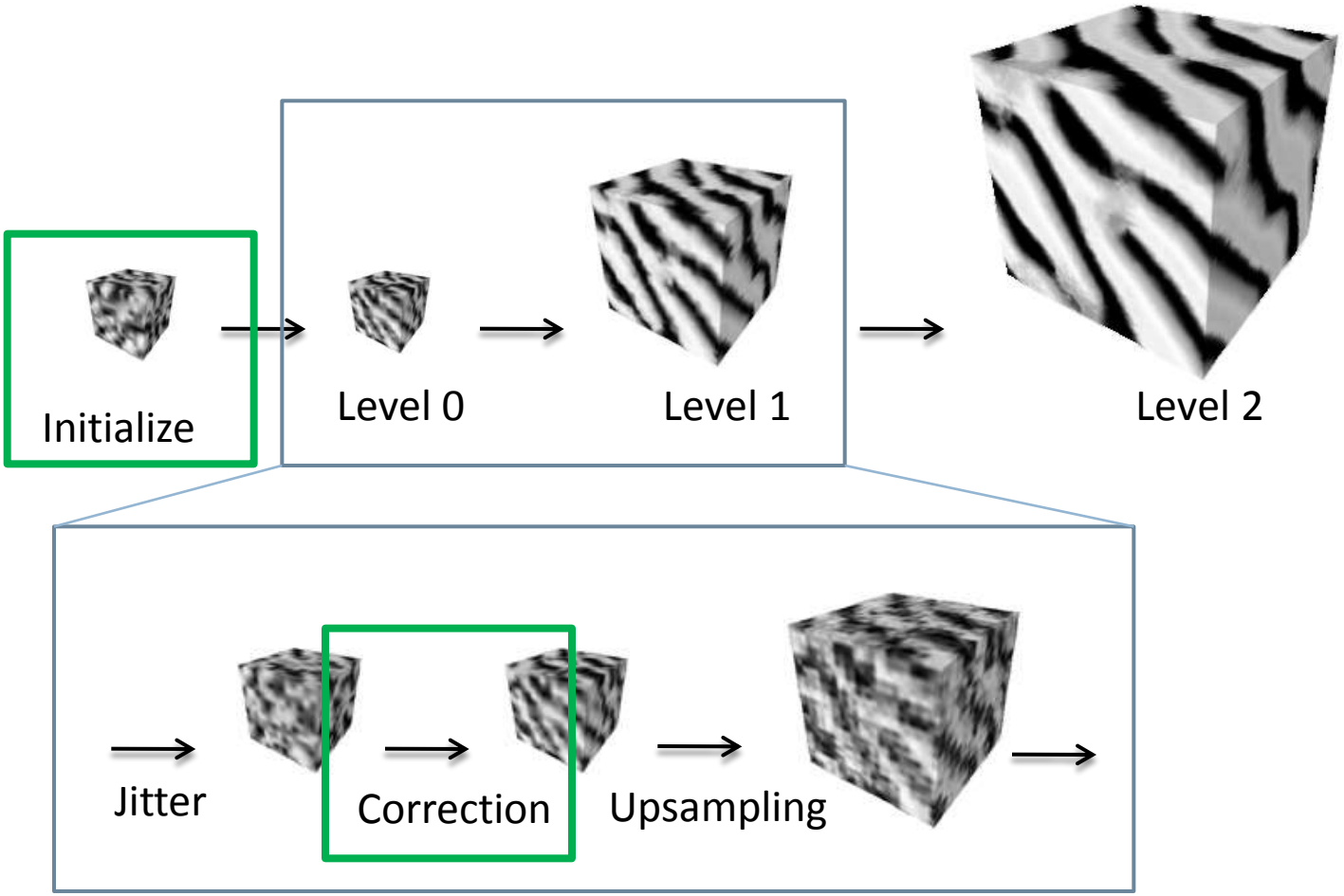
Synthesis result using
random initialization

Synthesis Pipeline



Synthesis Overview

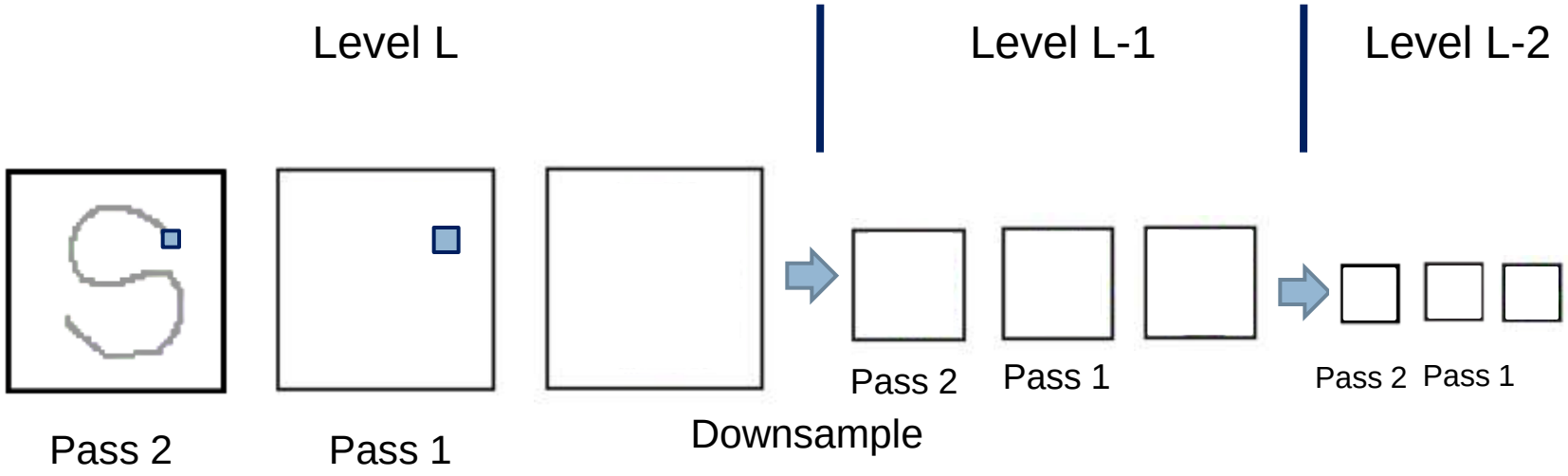
- Based on 2D parallel synthesis [LH2005]



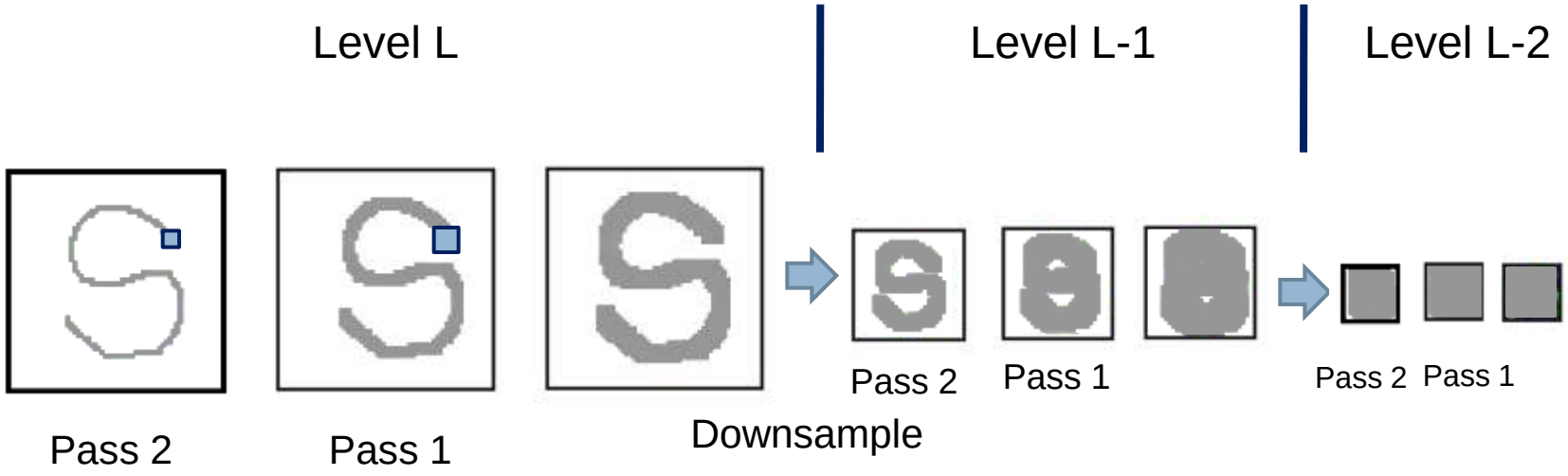
Working around the surface

- How to avoid computing the full volume?
  - Restrict computation to voxel subset (Lazy Synthesis)
  - Store data only around the surface

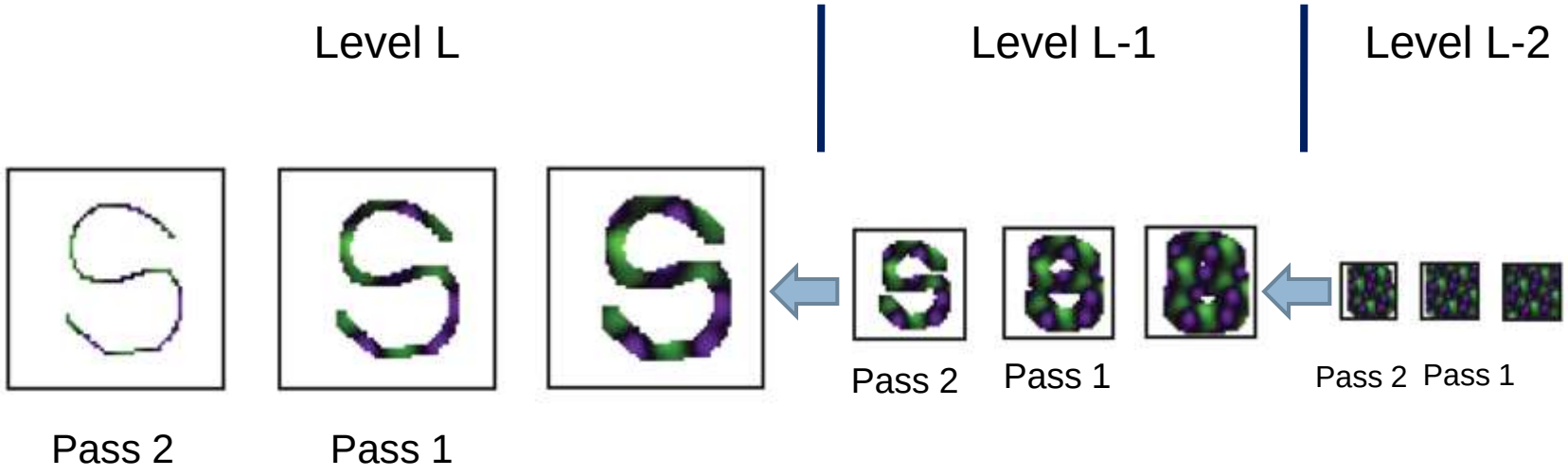
Lazy synthesis



Lazy synthesis



Lazy synthesis

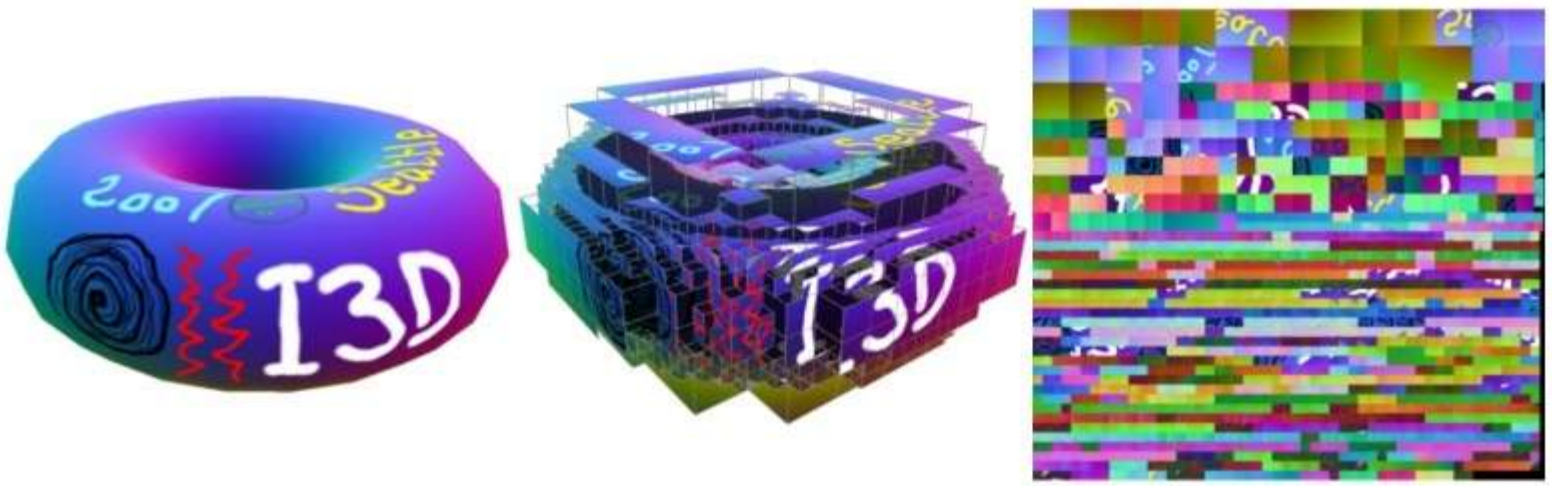


Working around the surface only

- We don’t want to allocate a full volume!

  ➔ Sparse volume data structure

- TileTree [LD07]
  - 2D tiles surrounding the surface



Contributions / overview

- 3D candidate pre-computation
  - Reduces dependency chain
- Parallel, spatially deterministic solid synthesis
  - Focus synthesis around surface
- Results

Performance

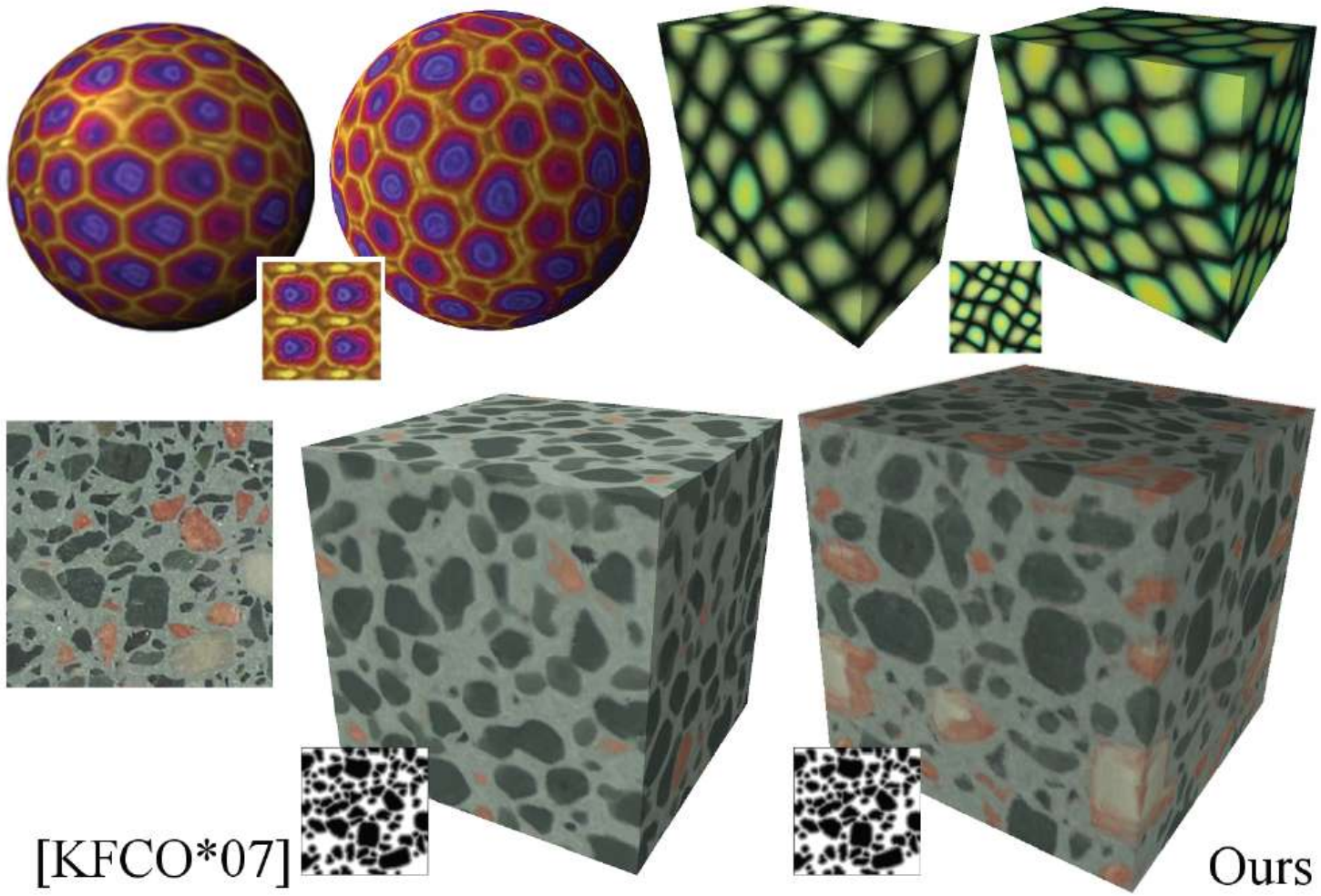
- Pre-computation (Intel Core2 6400)

Exemplar size	Pre-computation Time
64x64	7 seconds
128x128	27 seconds

- Synthesis on GPU (Geforce 8800 Ultra)
  - **Full volume** synthesis

Volume size	Synthesis Time
64x64x64	220 milliseconds
128x128x128	1700 milliseconds

Comparison



10 to 90 minutes

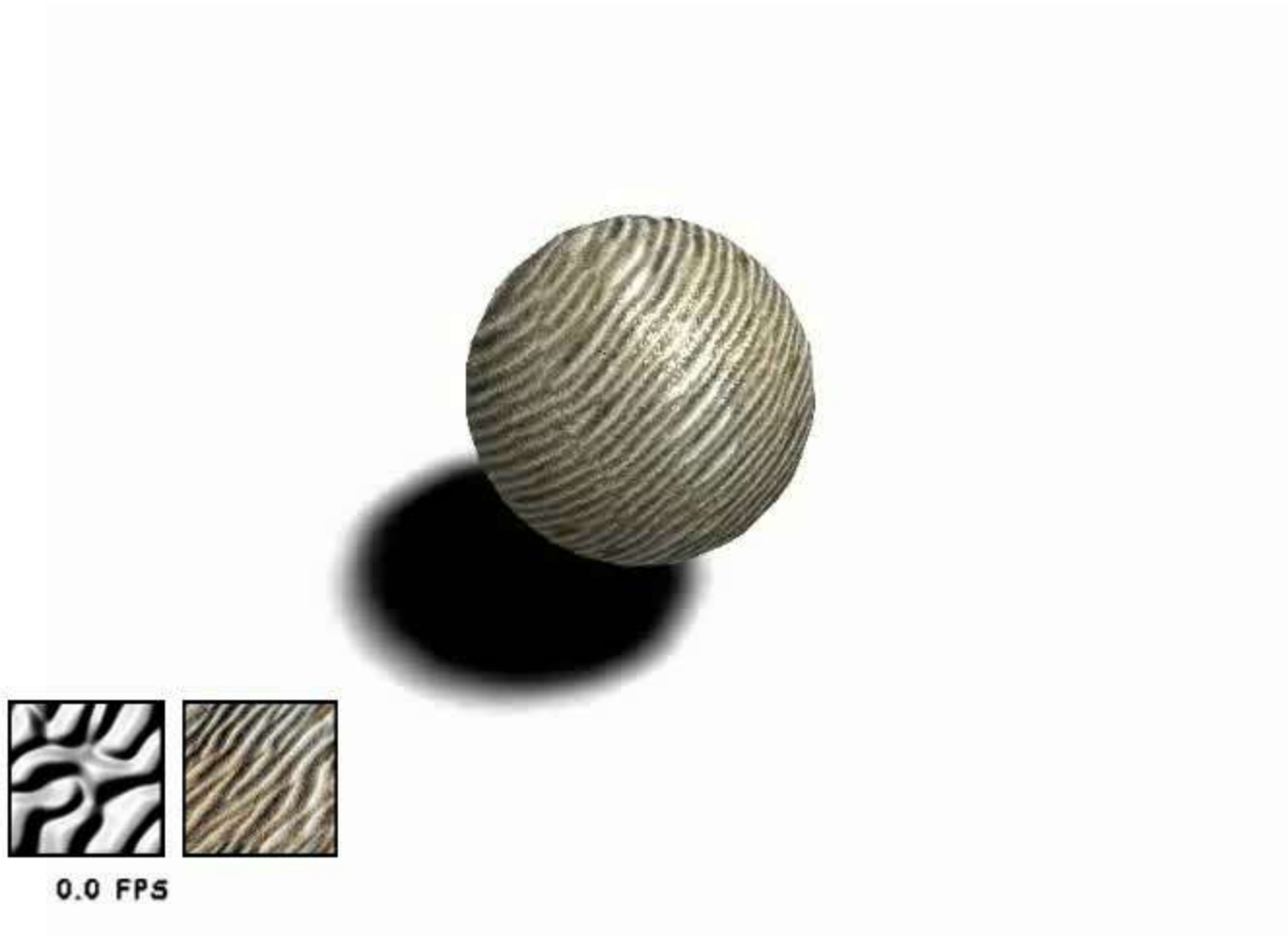
2 seconds

Result – Volumetric Rendering

Results – Realtime Slicing

906.1 FPS

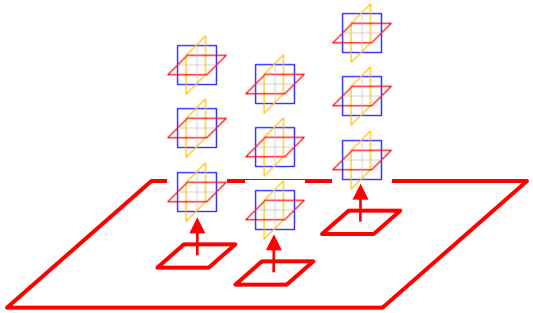
Result – Realtime Slicing

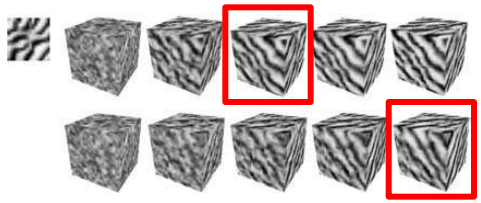


Result – Realtime Fractures

Summary

- 3D Candidate pre-computation
  - Enforce color consistency
  - Enable coherence in all directions
  - Reduced dependency
- Lazy parallel solid synthesis
  - Only evaluates what is necessary
- Interactive solid texturing
  - GPU implementation

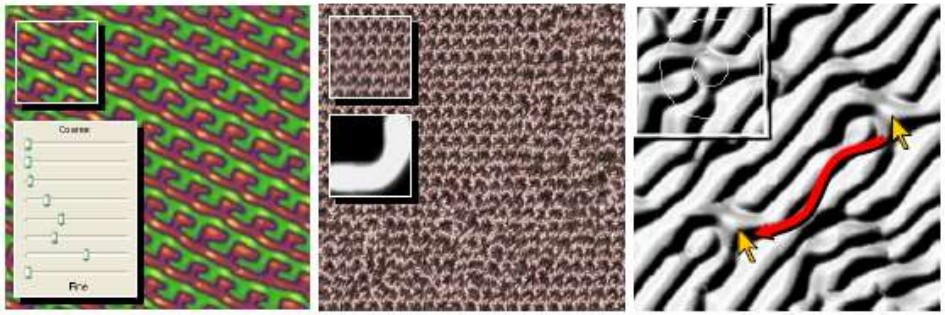




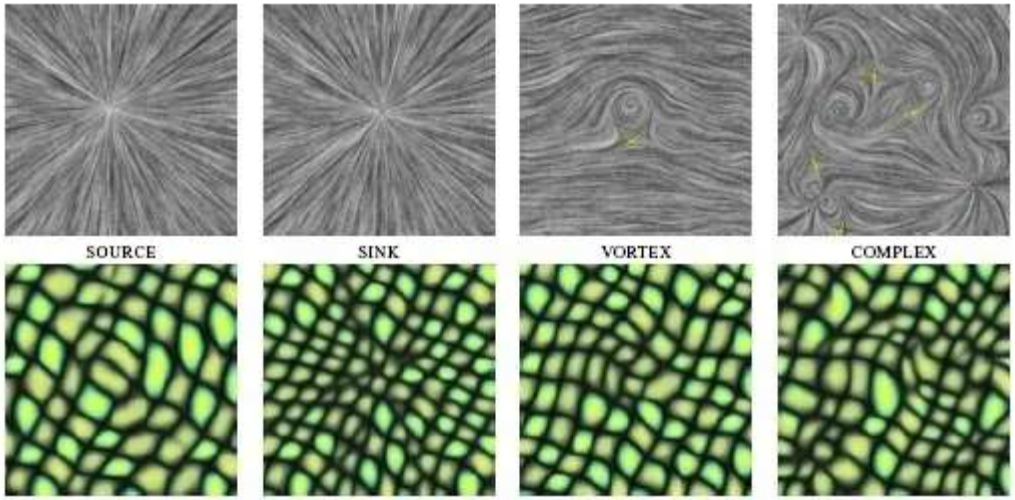
54MB instead of 3 GB

Future work

- More control



- Flow guided synthesis

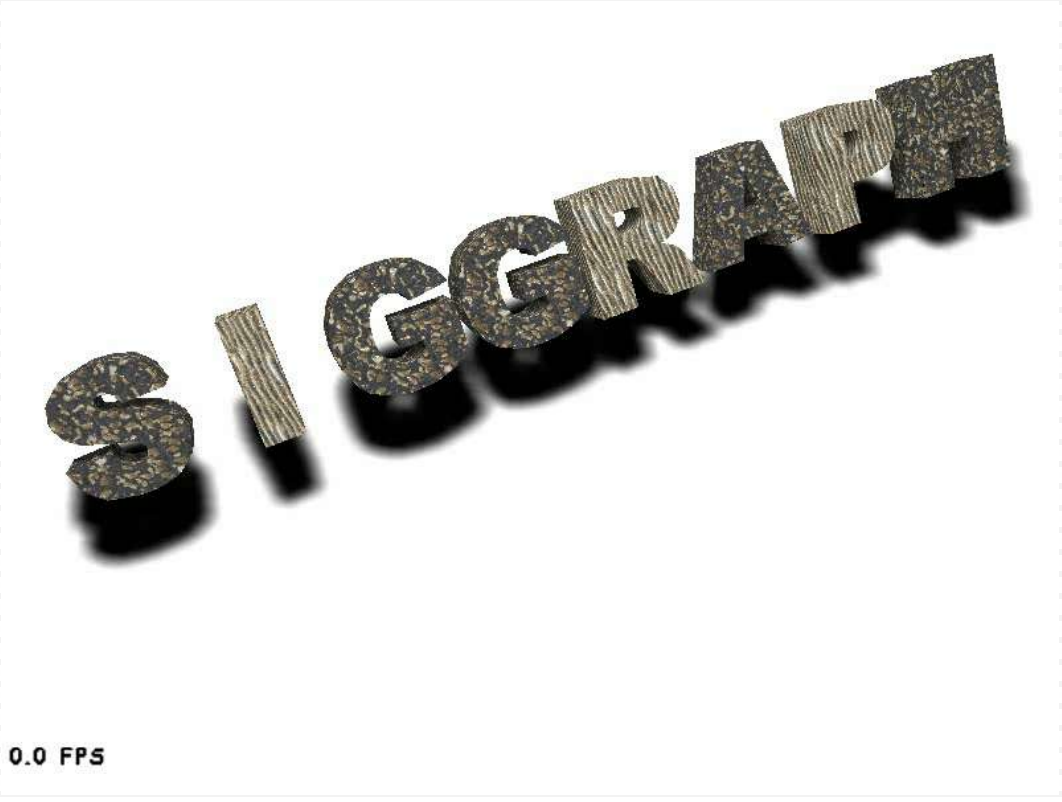


Acknowledgements

- Baining Guo and Li-Yi Wei
- The anonymous reviewers
- Su Wang for 3D modeling
- Work partly supported by NSFC grant 10547002

Thank you

Any questions ?



;)

Surfaces