Deferred Adaptive Compute Shading

Ian Mallett

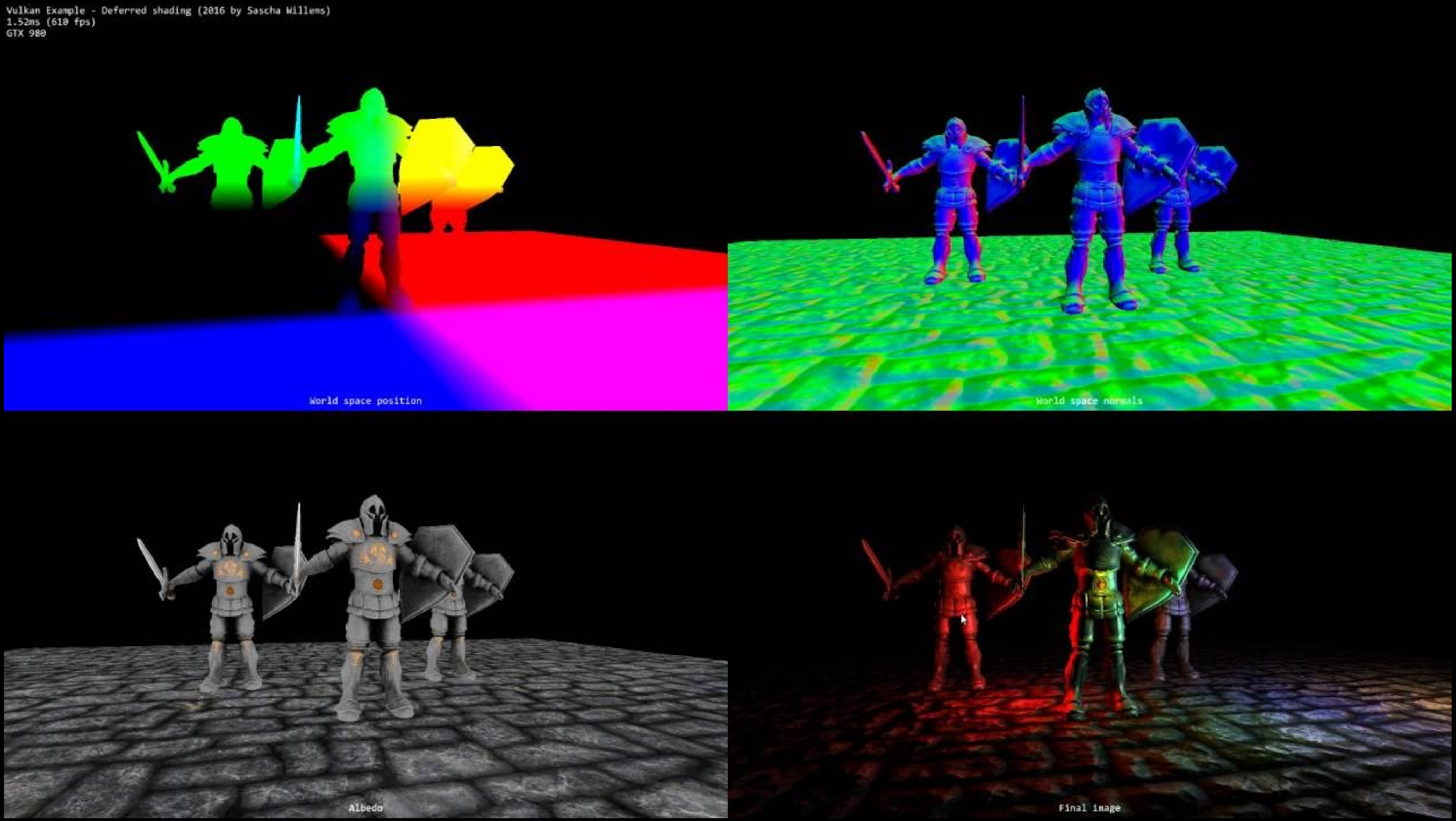
Cem Yuksel

Utah Graphics

2018

Fragments Are Expensive

- Deferred shading: reduces expensive overdraw



```
Vulkan Example - Deferred shading (2016 by Sascha Willems)
1.52ms (610 fps)
GTX 980
```

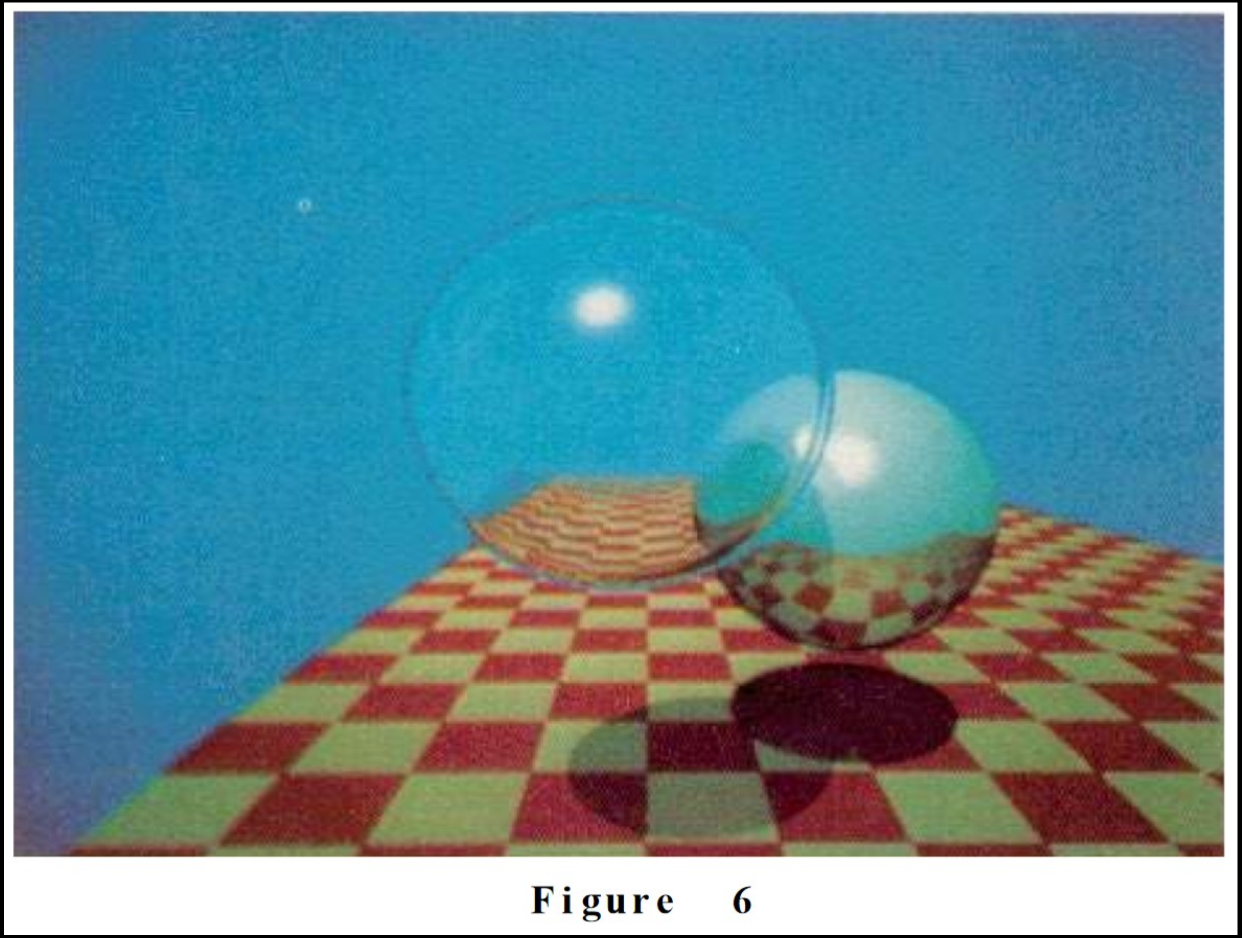
(image by Sascha Willems)

Fragments Are Expensive

- **Many** adaptive sampling algorithms in raytracing
  - With us since the beginning!

"I had written a draft of a SIGGRAPH conference submission and was rendering illustrations to be included in the paper. The submission deadline was near, but with 16x super-sampling, the estimated rendering times extended beyond the submission deadline. The spontaneous idea of ***adaptively super-sampling was a life saver*** *because it only added additional samples where needed. It was implemented within a couple of hours and the paper was edited to include this new idea while the illustrations were being rendered."*

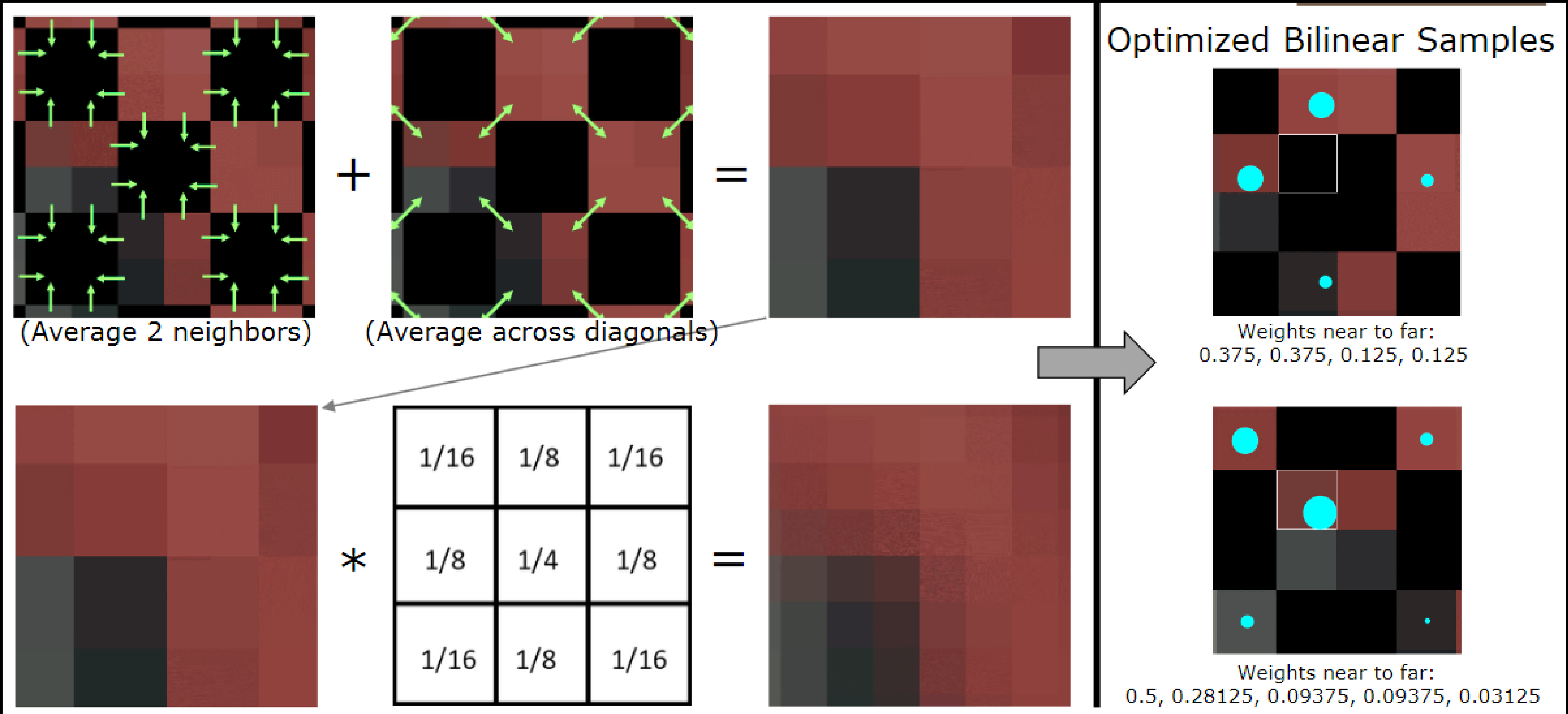
-J. Turner Whitted



(image by J. Turner Whitted)

Fragments Are Expensive

- Checkerboard rendering: reduce shades to < 1 / pixel. Adaptive sampling doesn’t map well to GPU!



1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

(slide by Alex Vlachos)

Deferred Adaptive Compute Shading

- Replacement for checkerboard rendering
  - (one-fewer pass, simple, provided code)

- Reduces shading adaptively

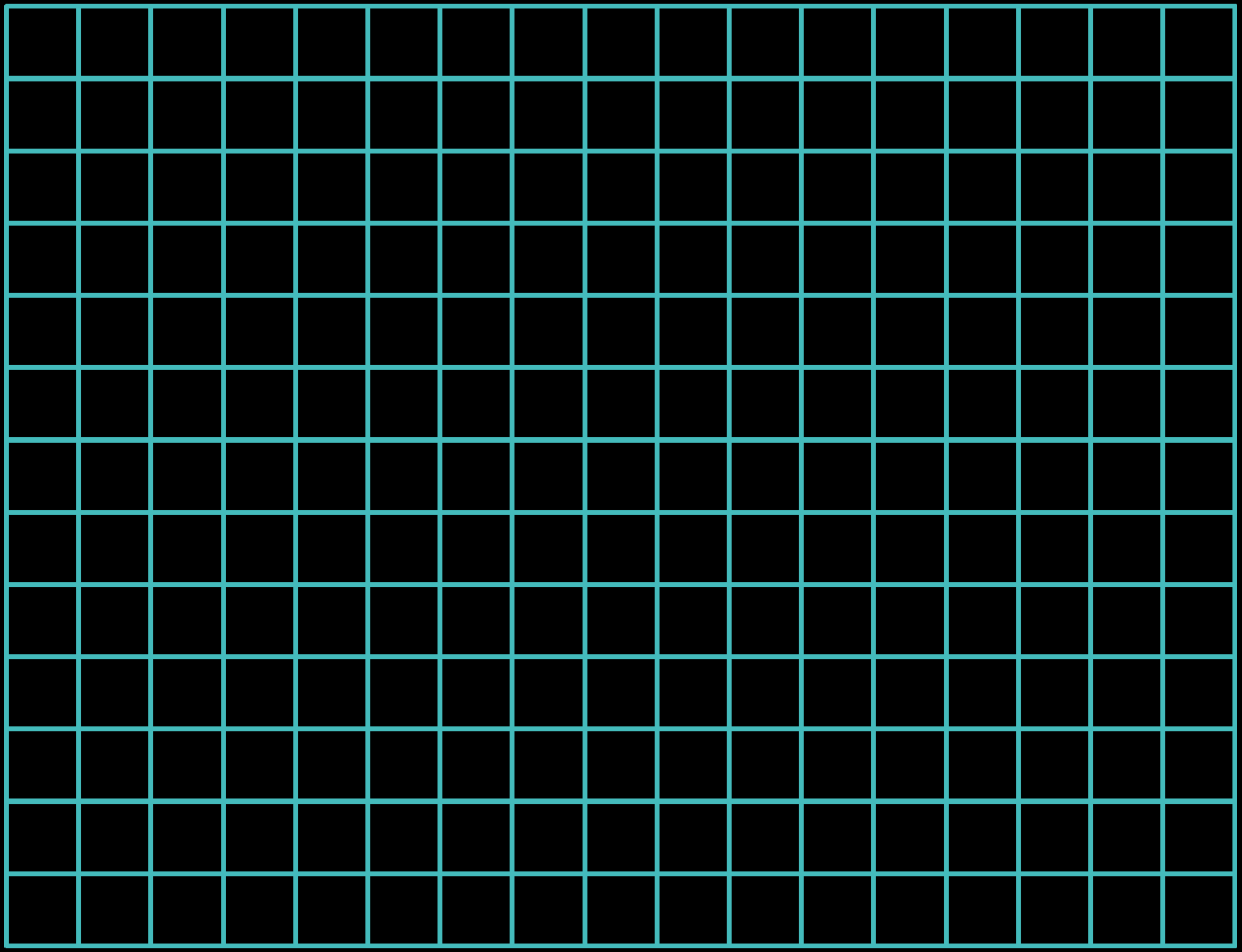
- Still GPU-friendly:
  - Typical results: 2–4× better quality/perf

Adaptive Subdivision

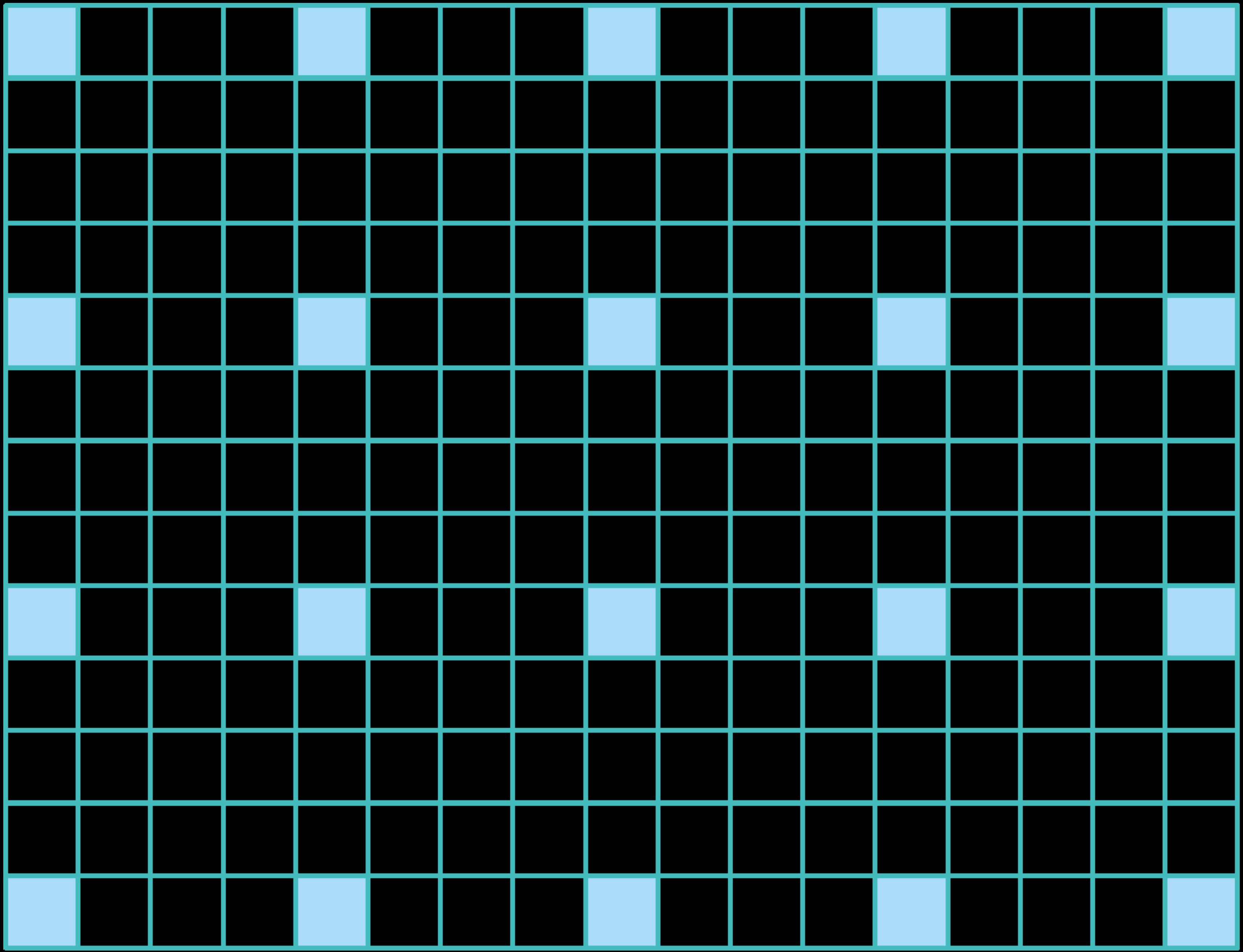
- Simple but proven adaptive subdivision scheme (inspired by V-Ray)
  - (Tried some others, but this one works best)

- Elegant rotational pattern

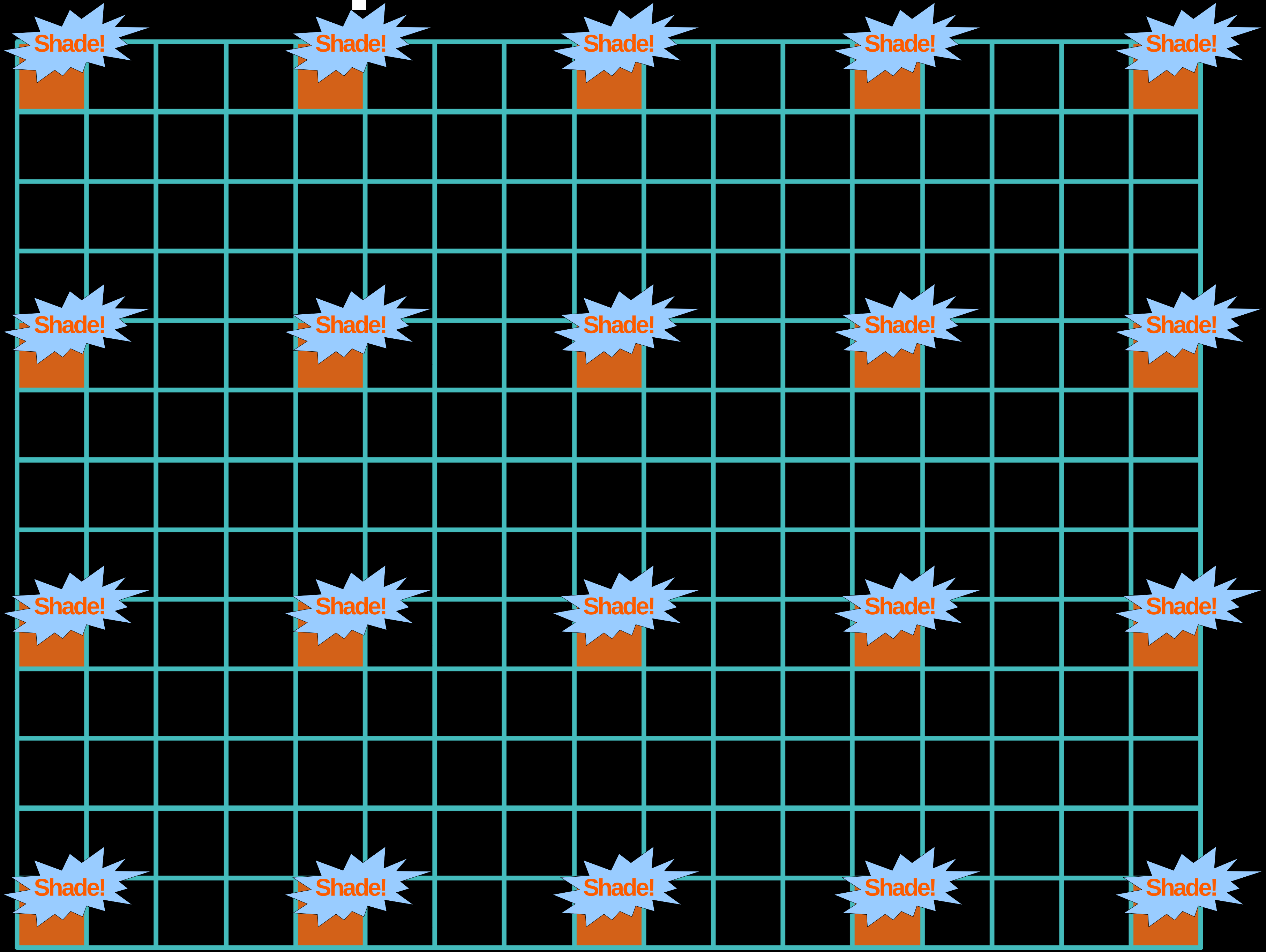
Adaptive Subdivision



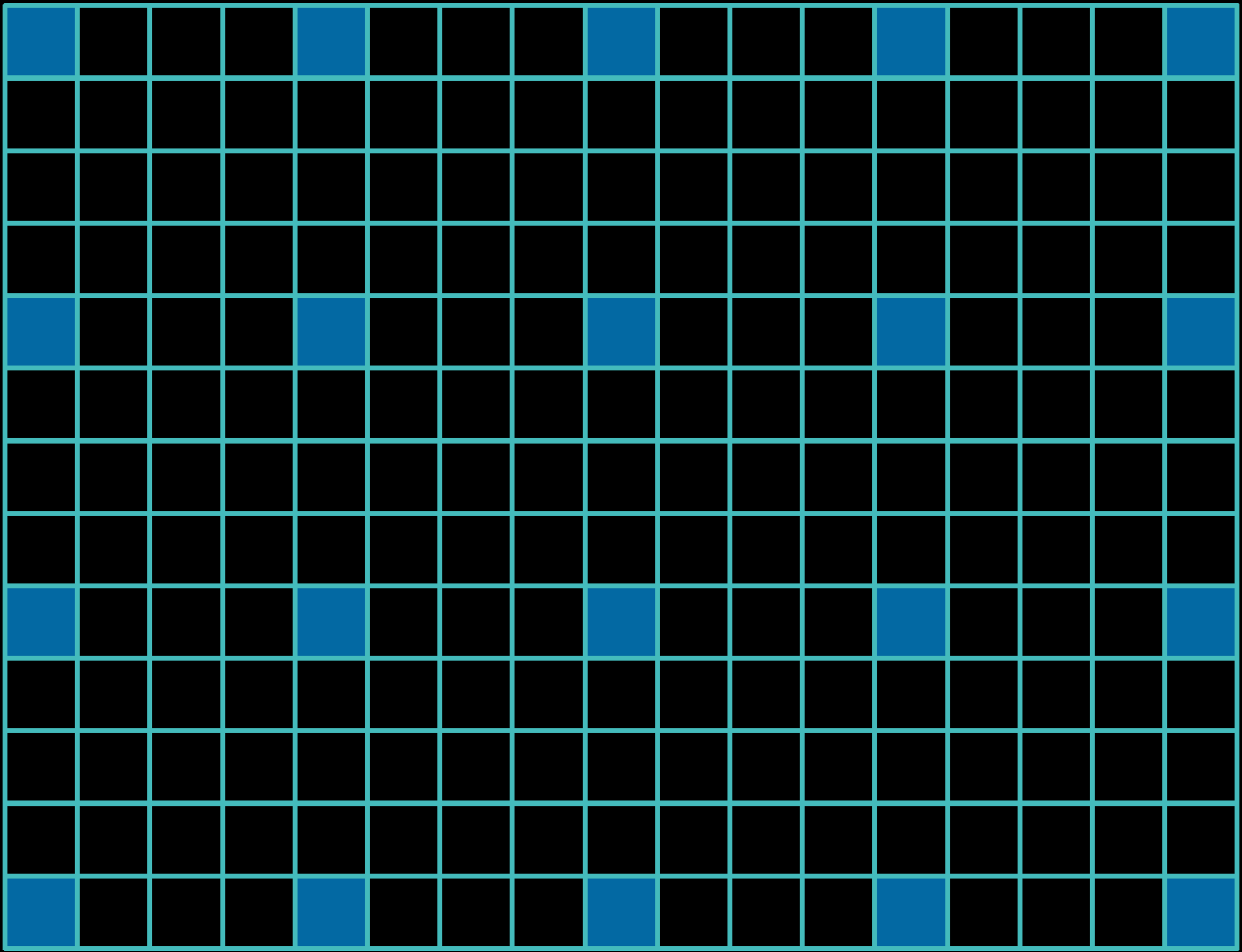
Adaptive Subdivision



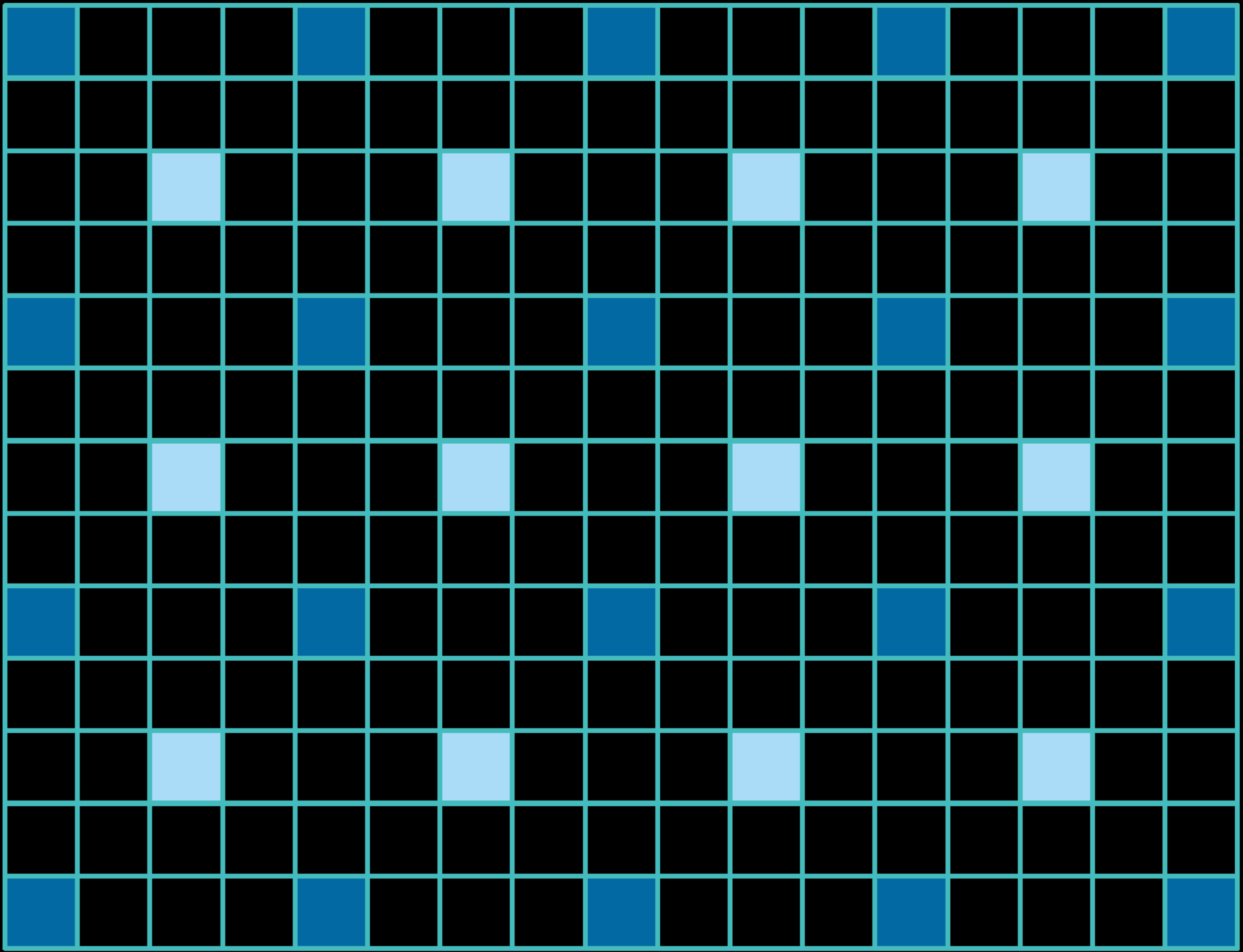
Adaptive Subdivision



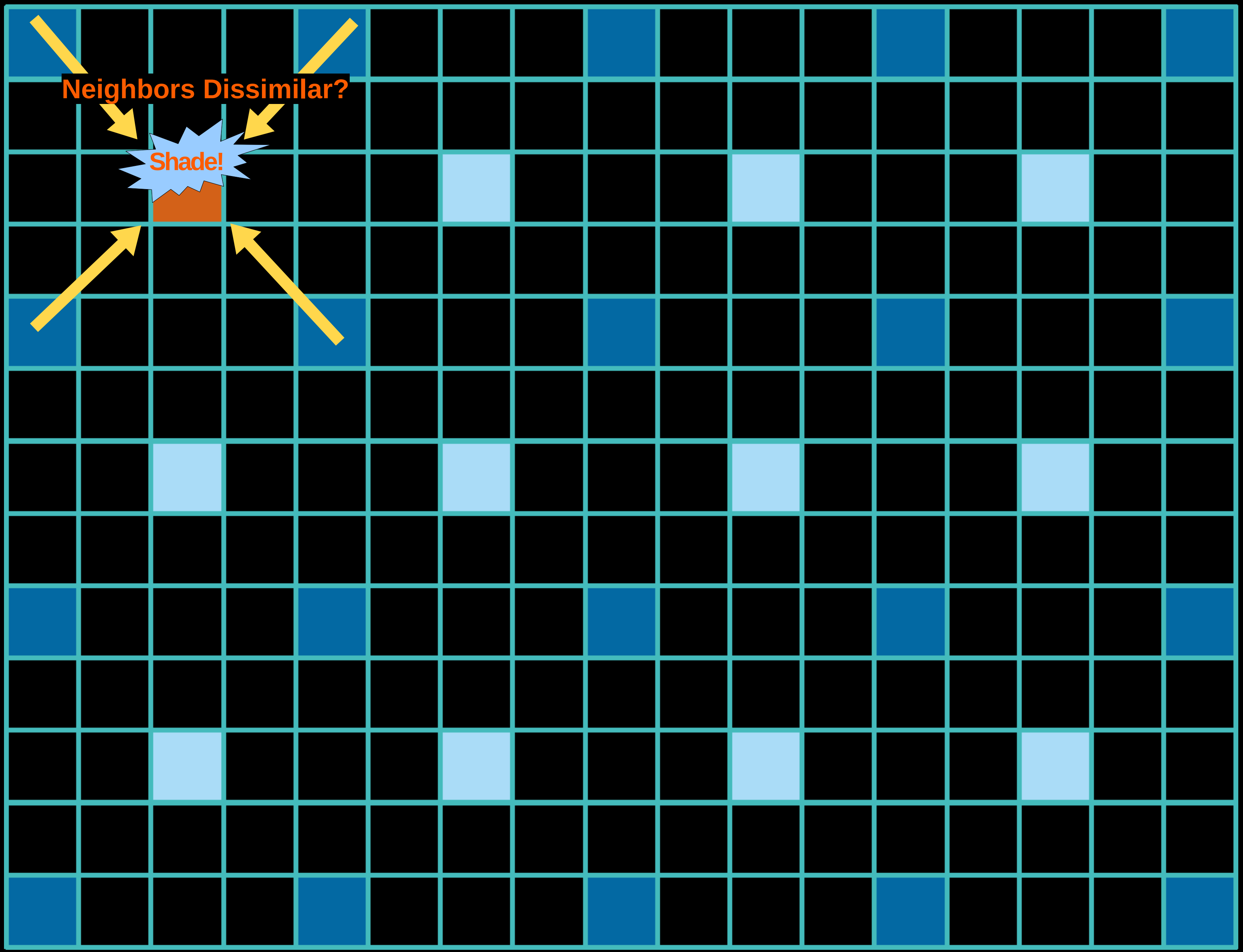
Adaptive Subdivision



Adaptive Subdivision



Adaptive Subdivision

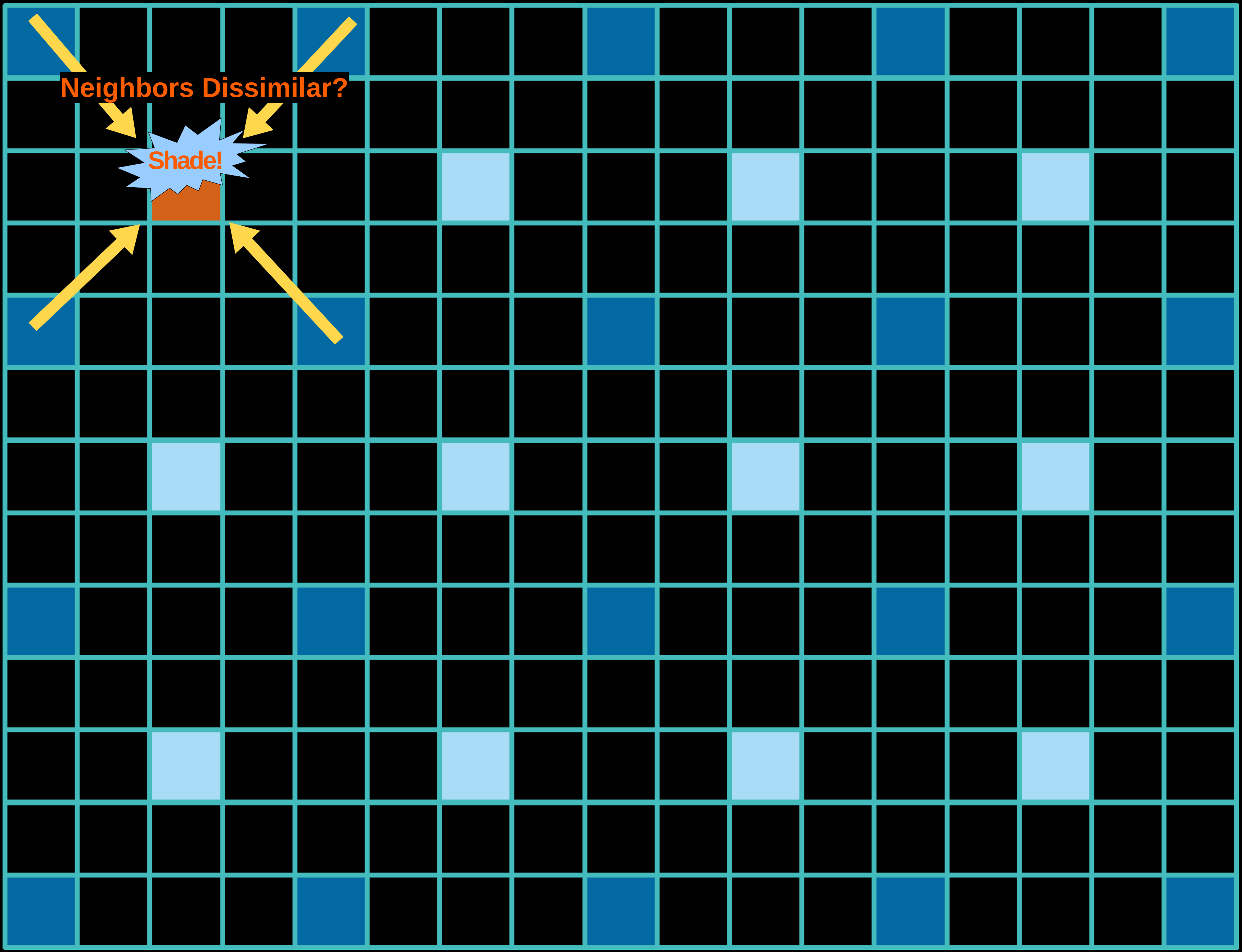


Adaptive Subdivision

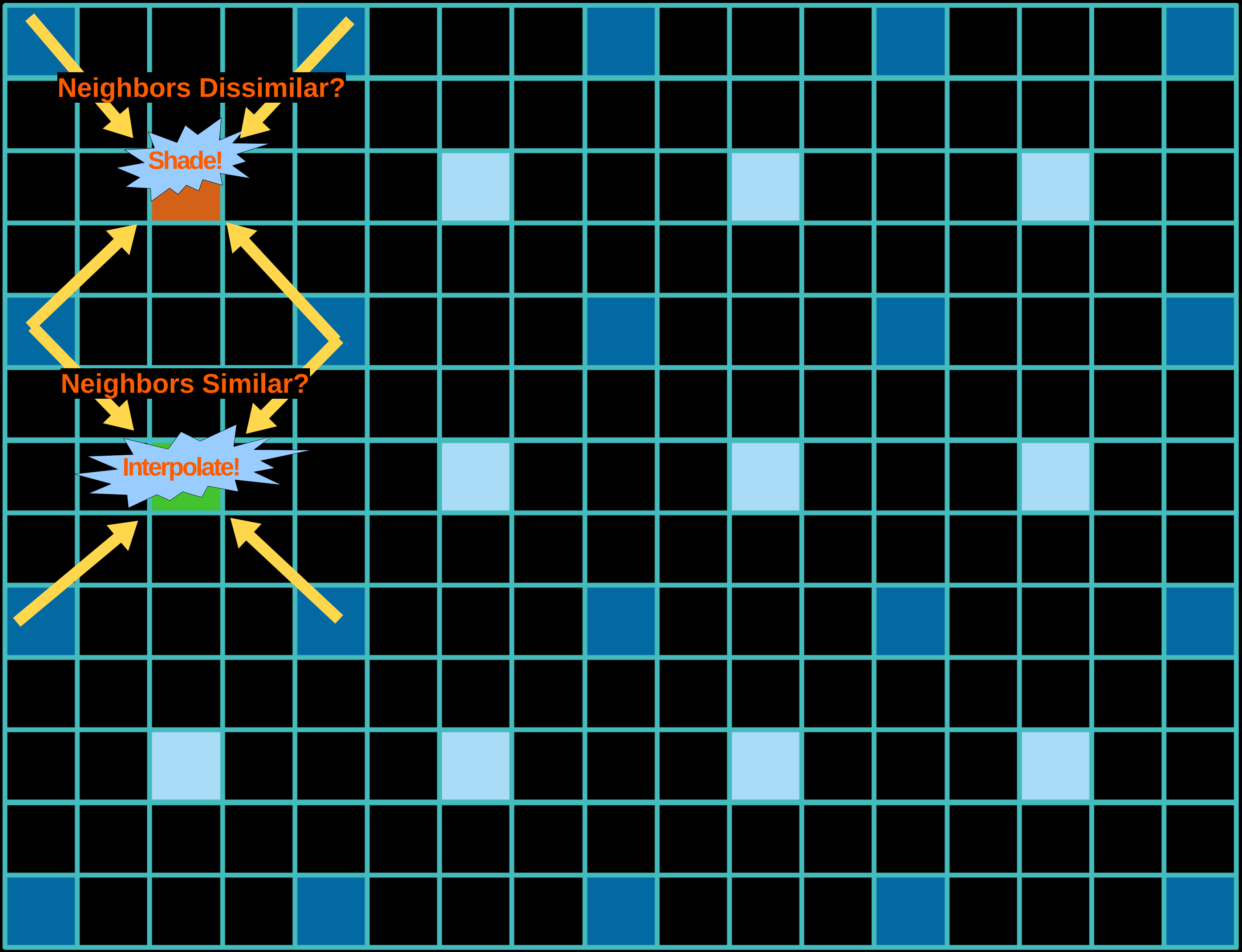
- “Similarity” given by user-defined metric

- We suggest:
  - “Dissimilar” if material IDs different
  - “Dissimilar” if final colors differ by threshold or more
  - Look at other G-buffer features?

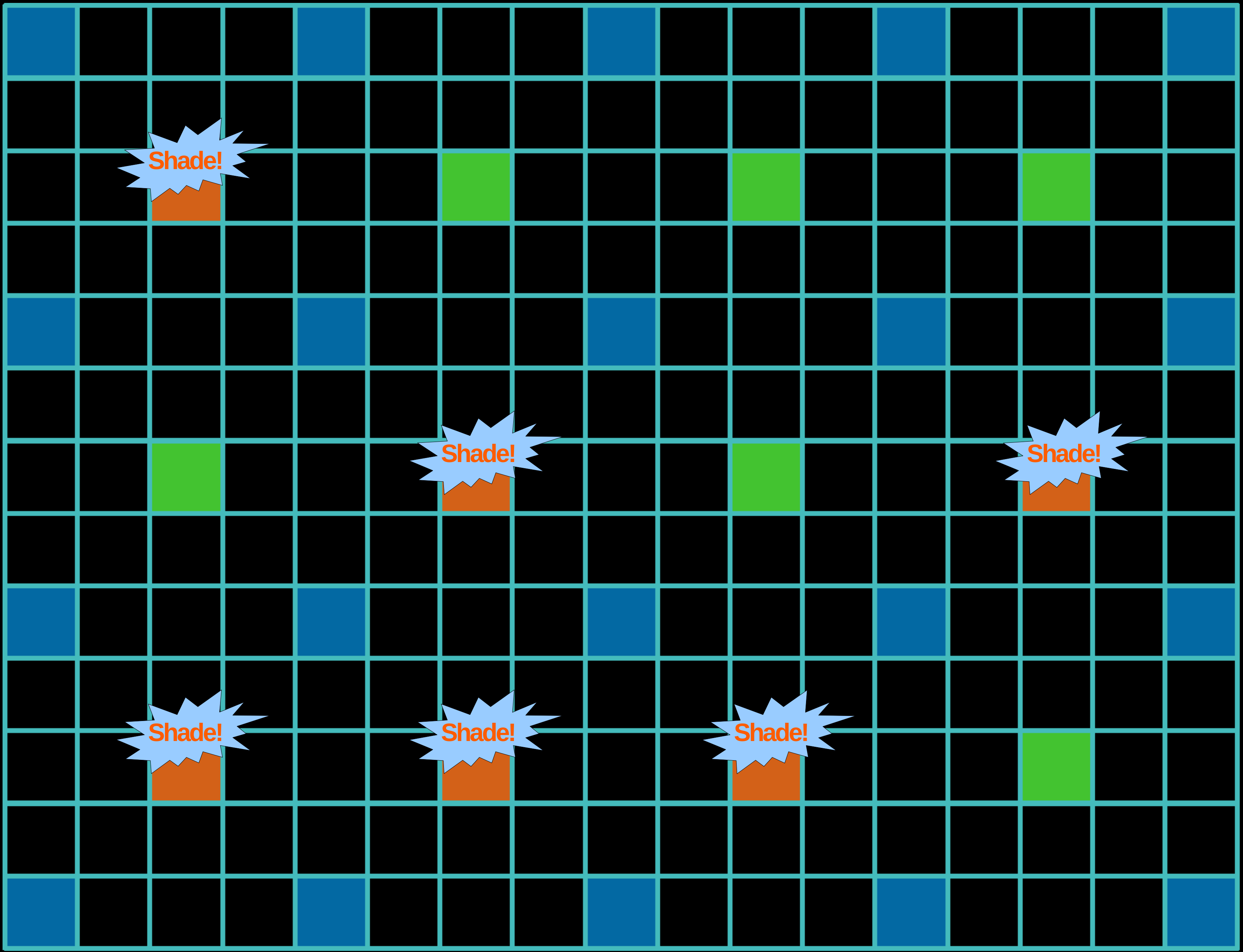
Adaptive Subdivision



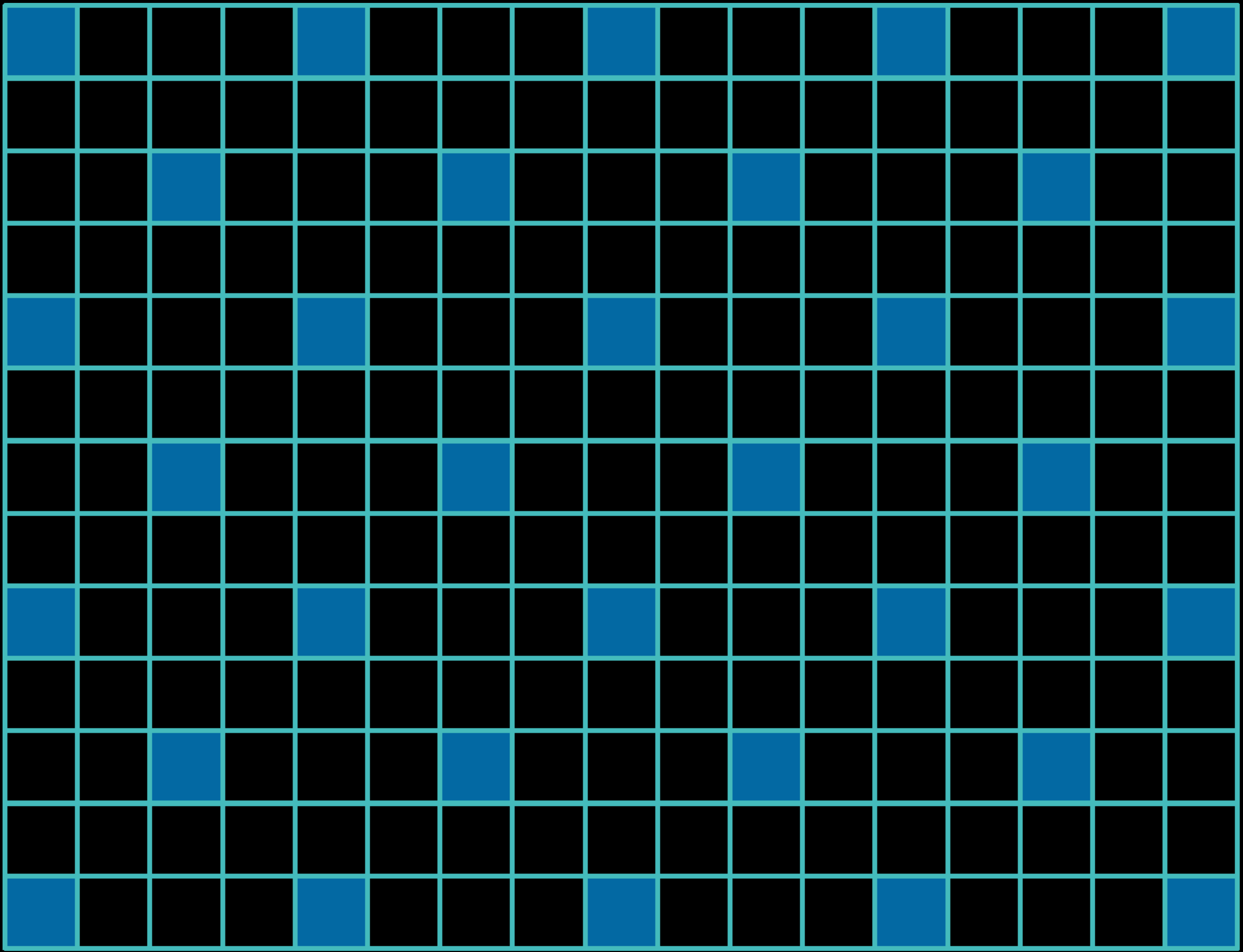
Adaptive Subdivision



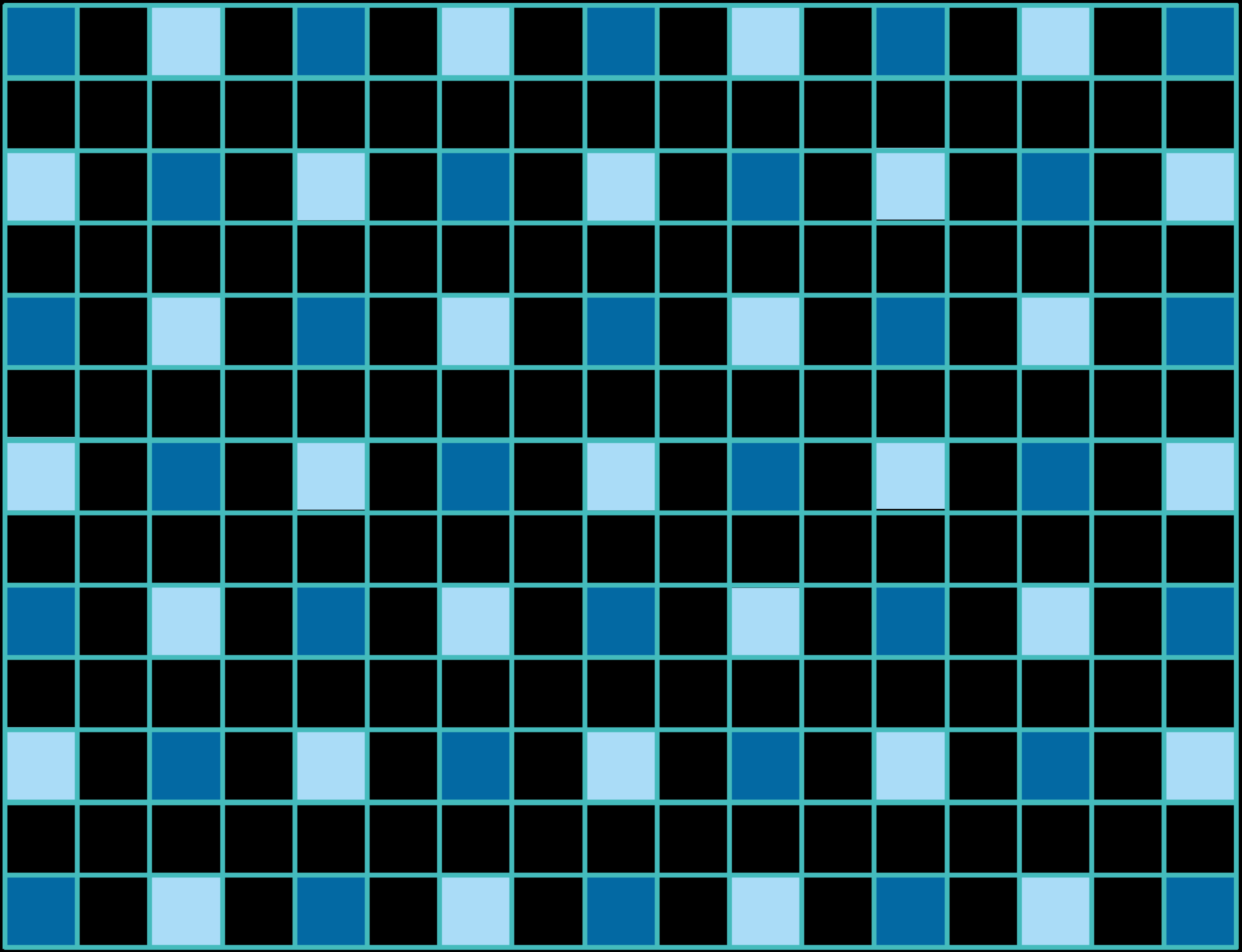
Adaptive Subdivision



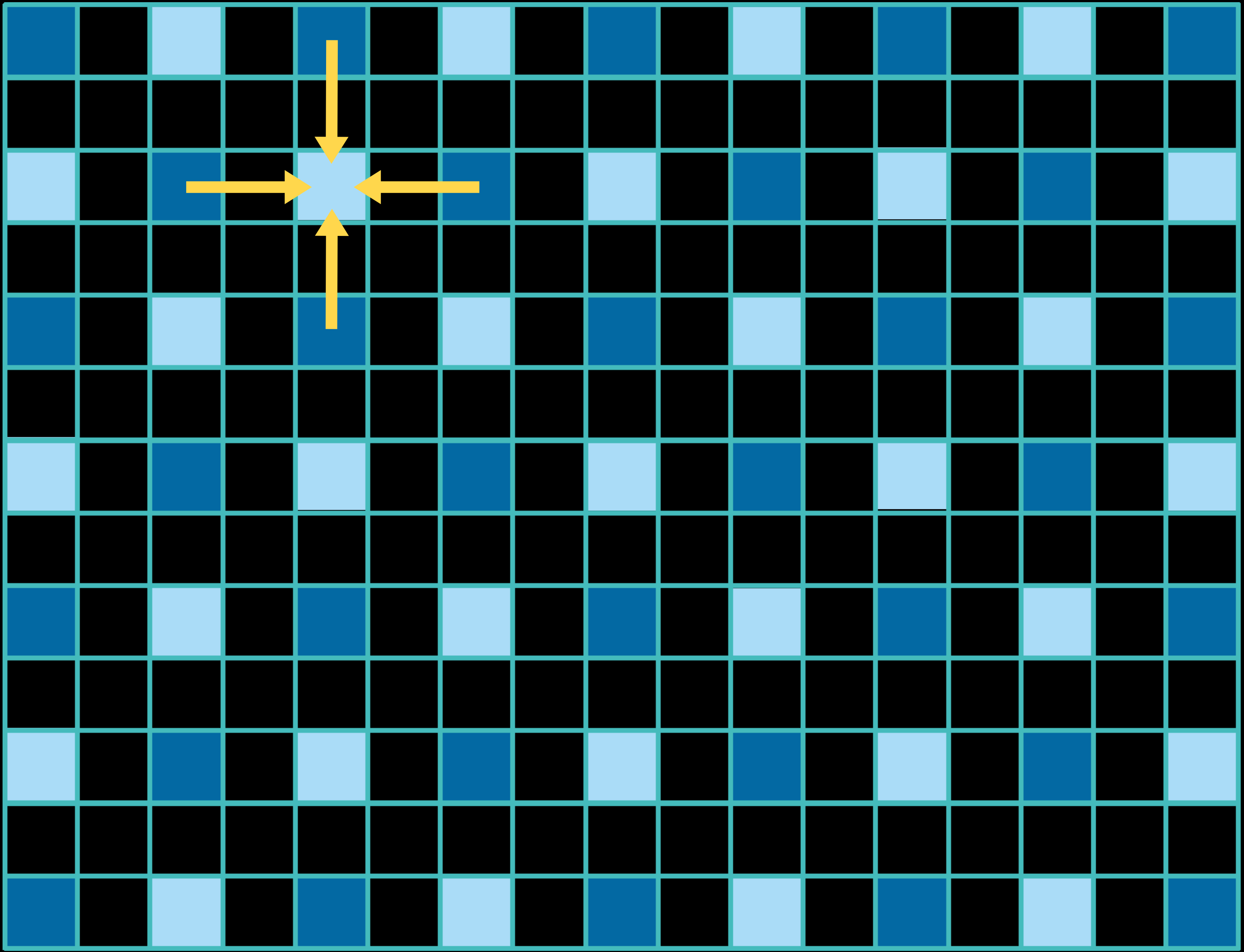
Adaptive Subdivision



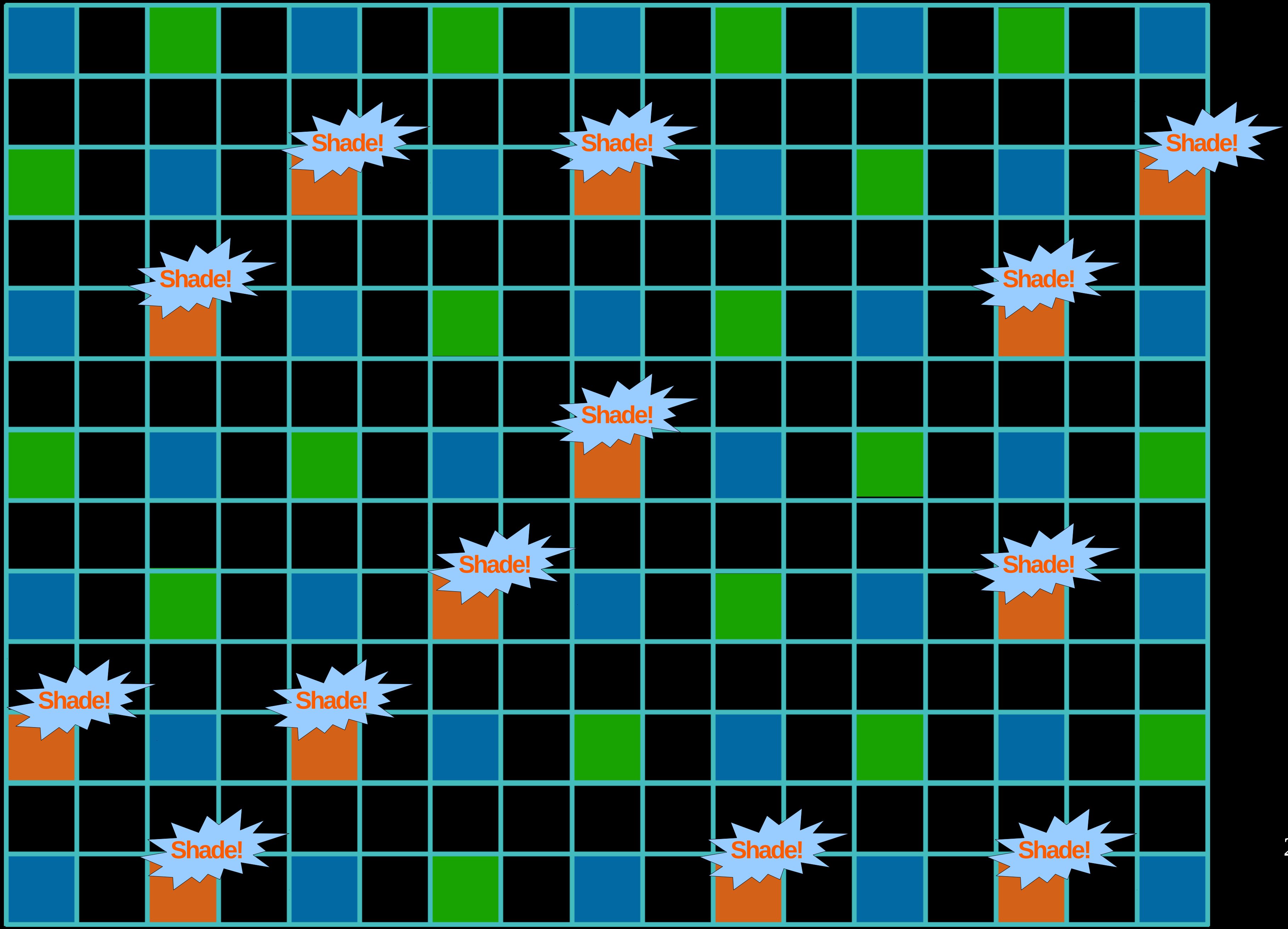
Adaptive Subdivision



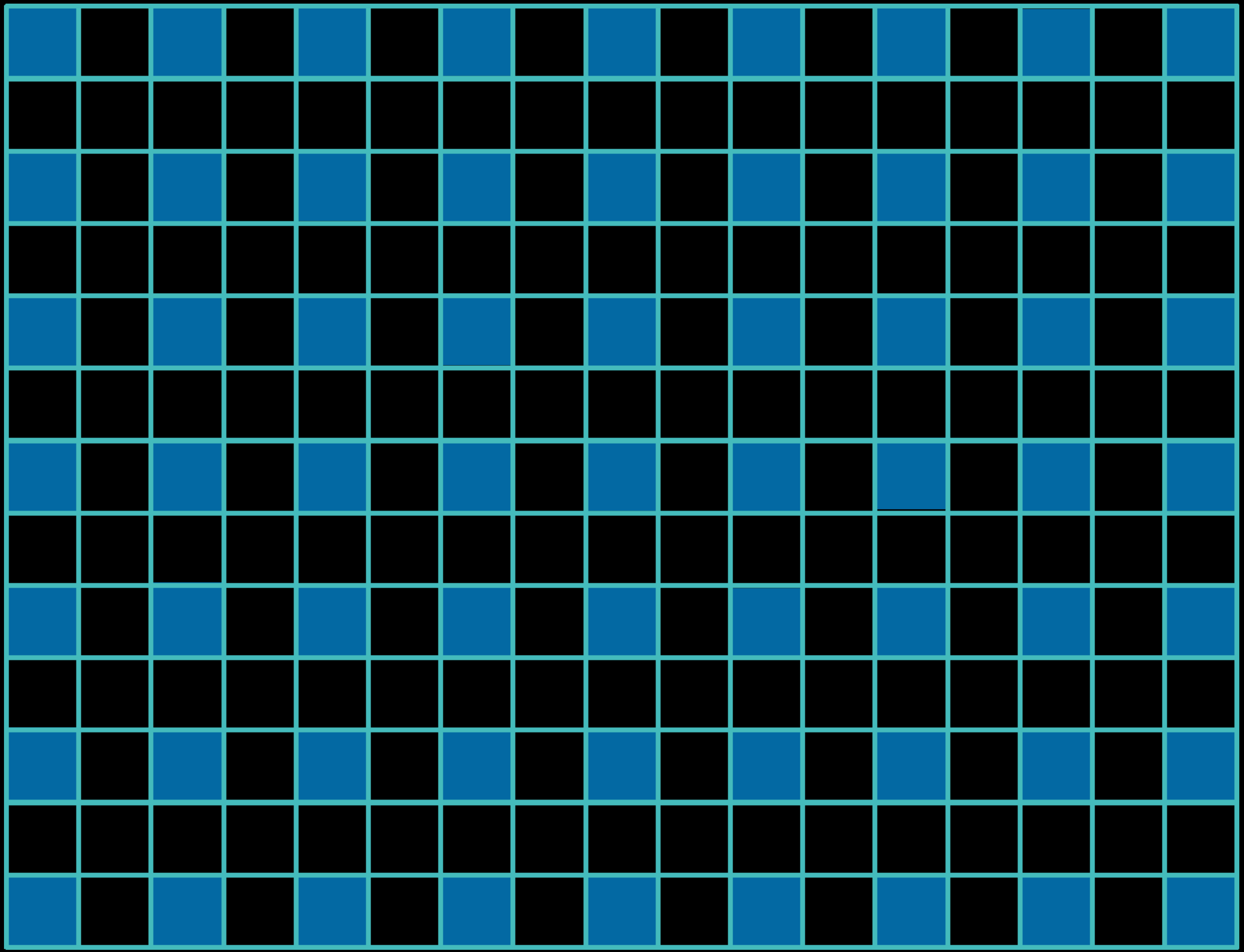
Adaptive Subdivision



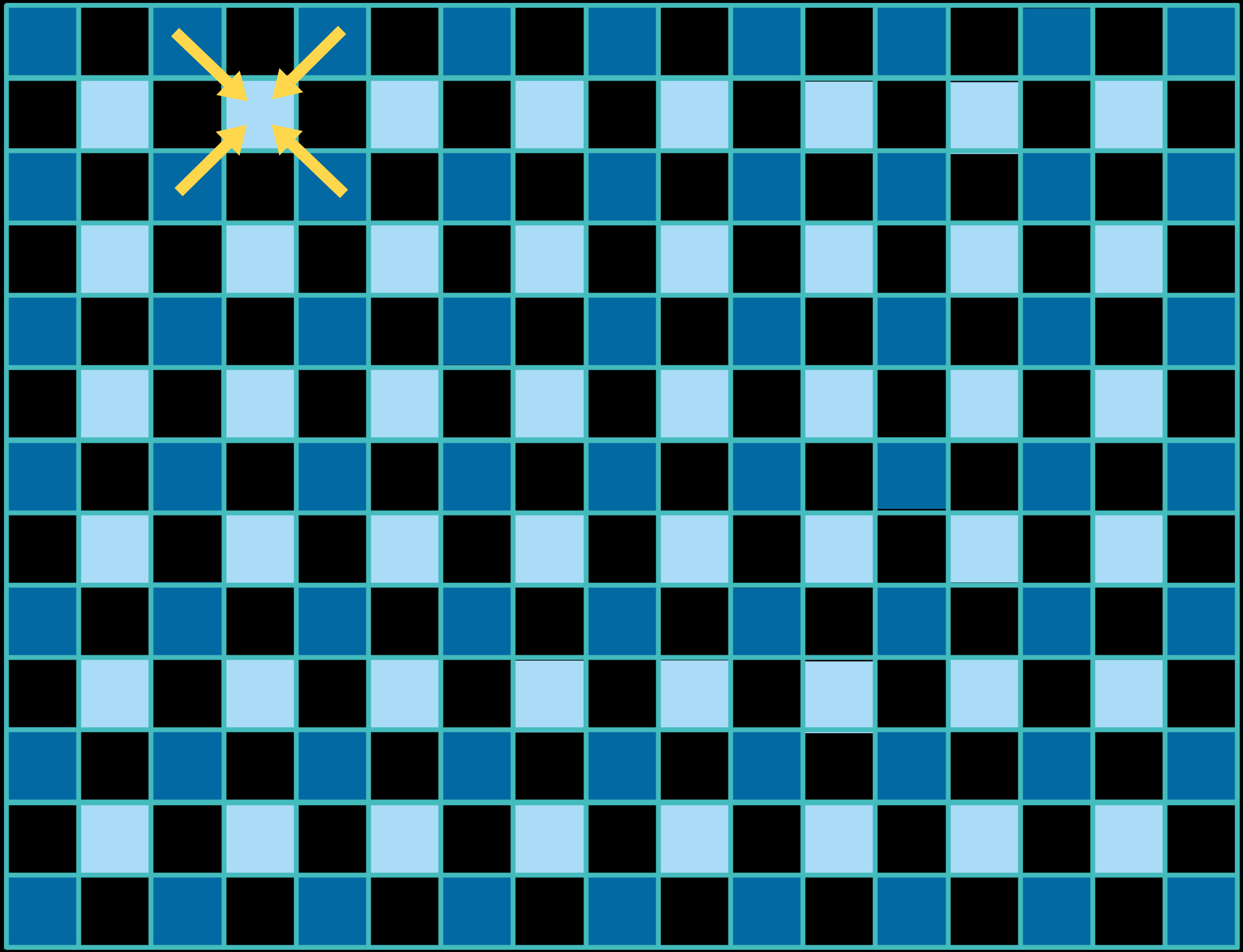
Adaptive Subdivision



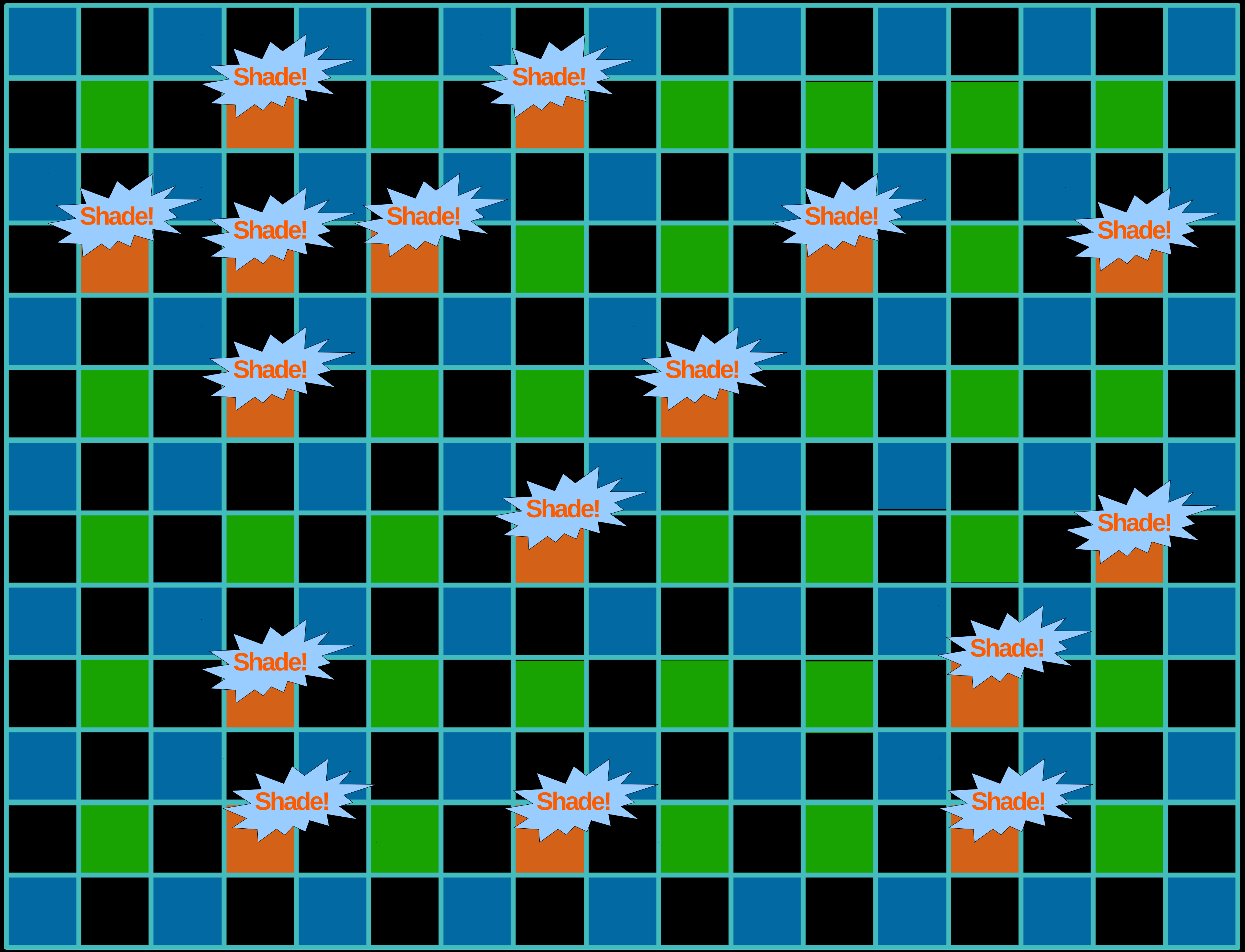
Adaptive Subdivision



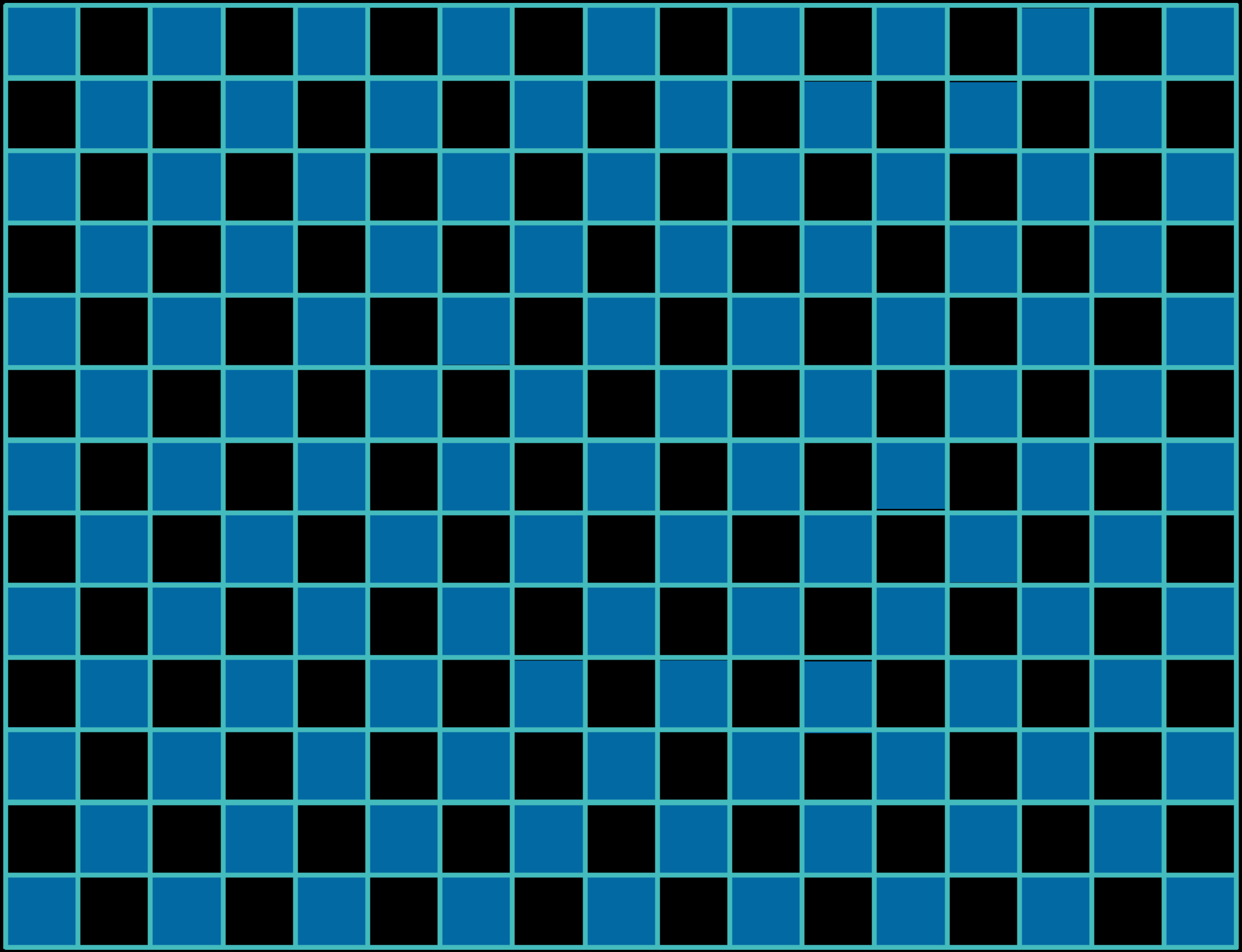
Adaptive Subdivision



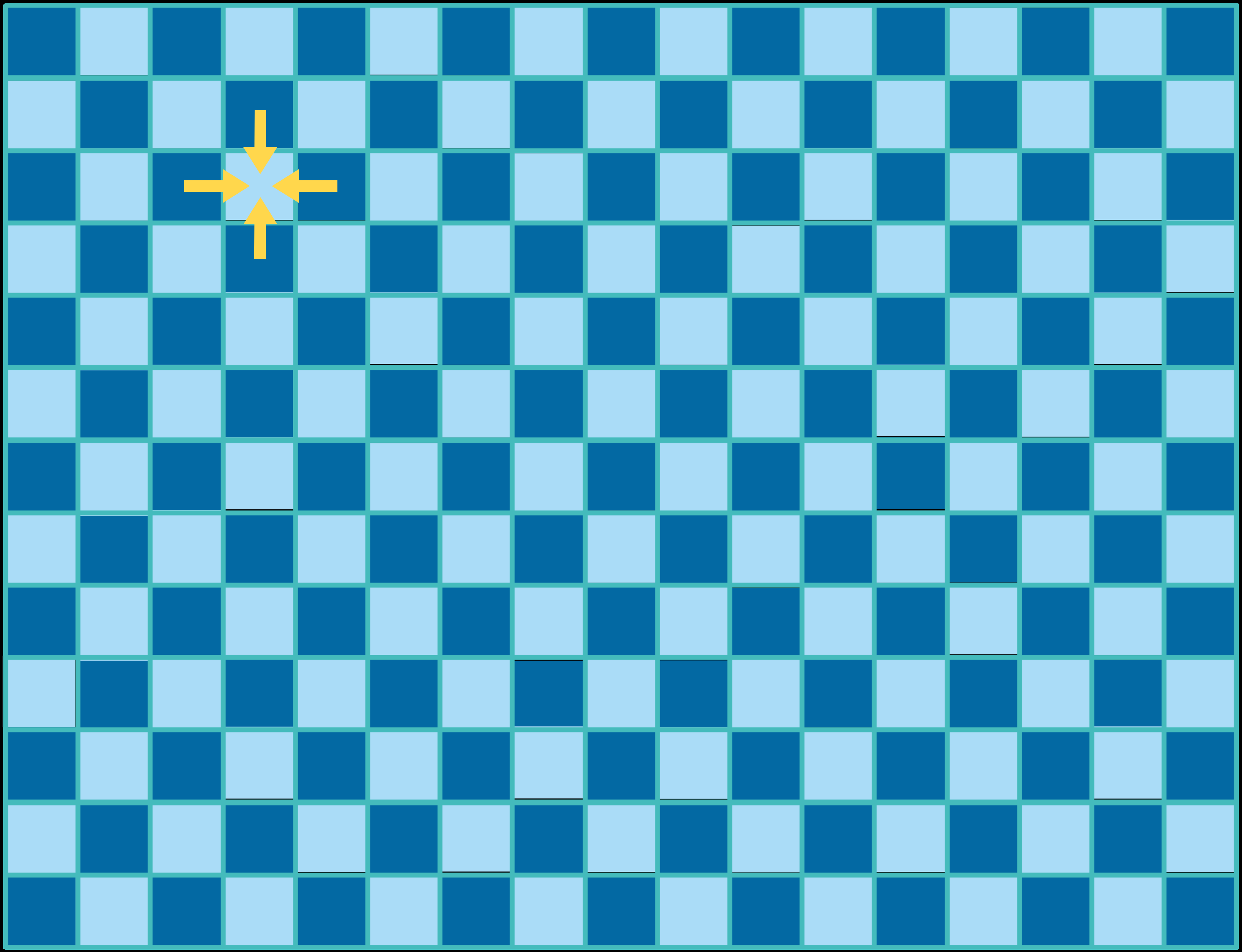
Adaptive Subdivision



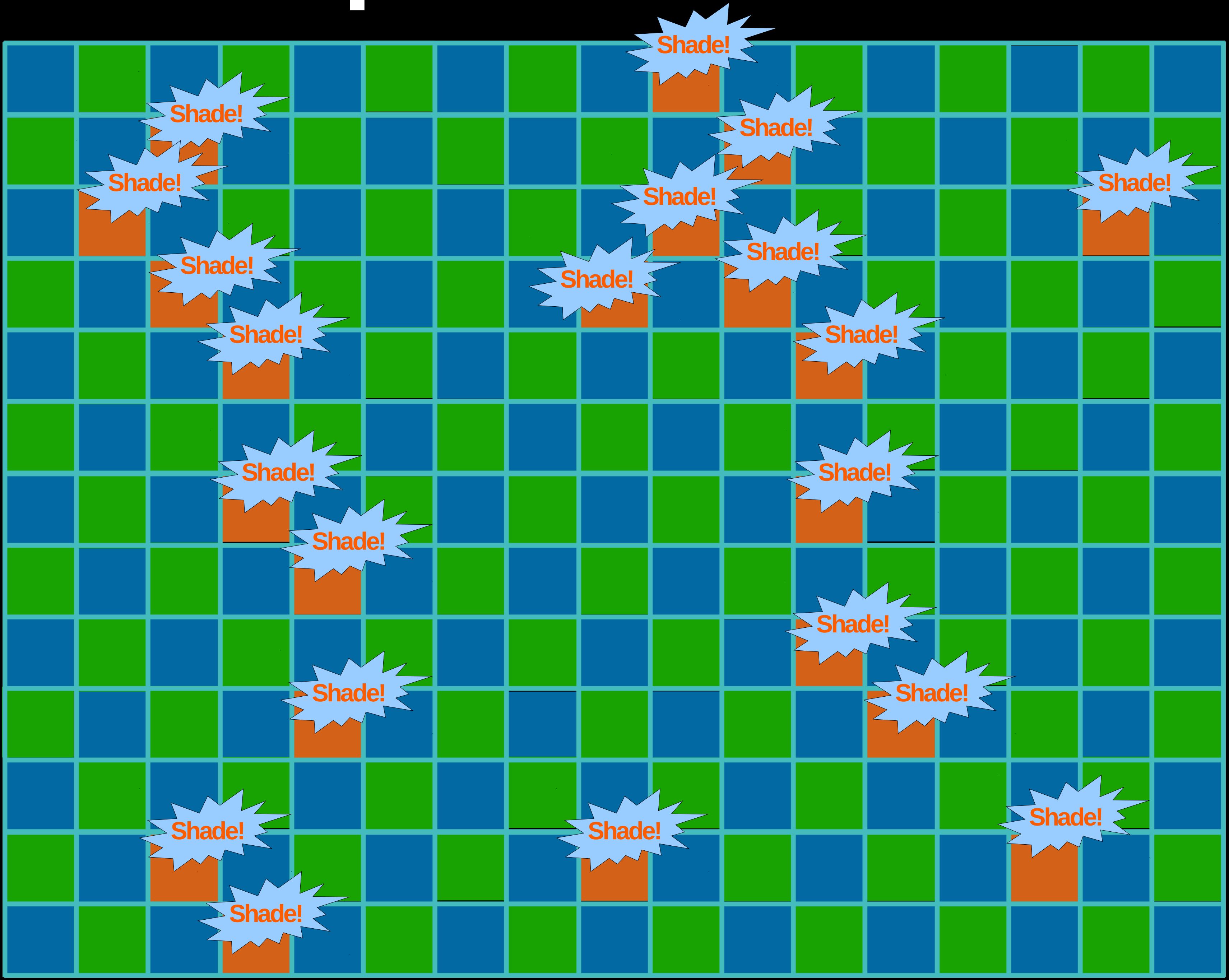
Adaptive Subdivision



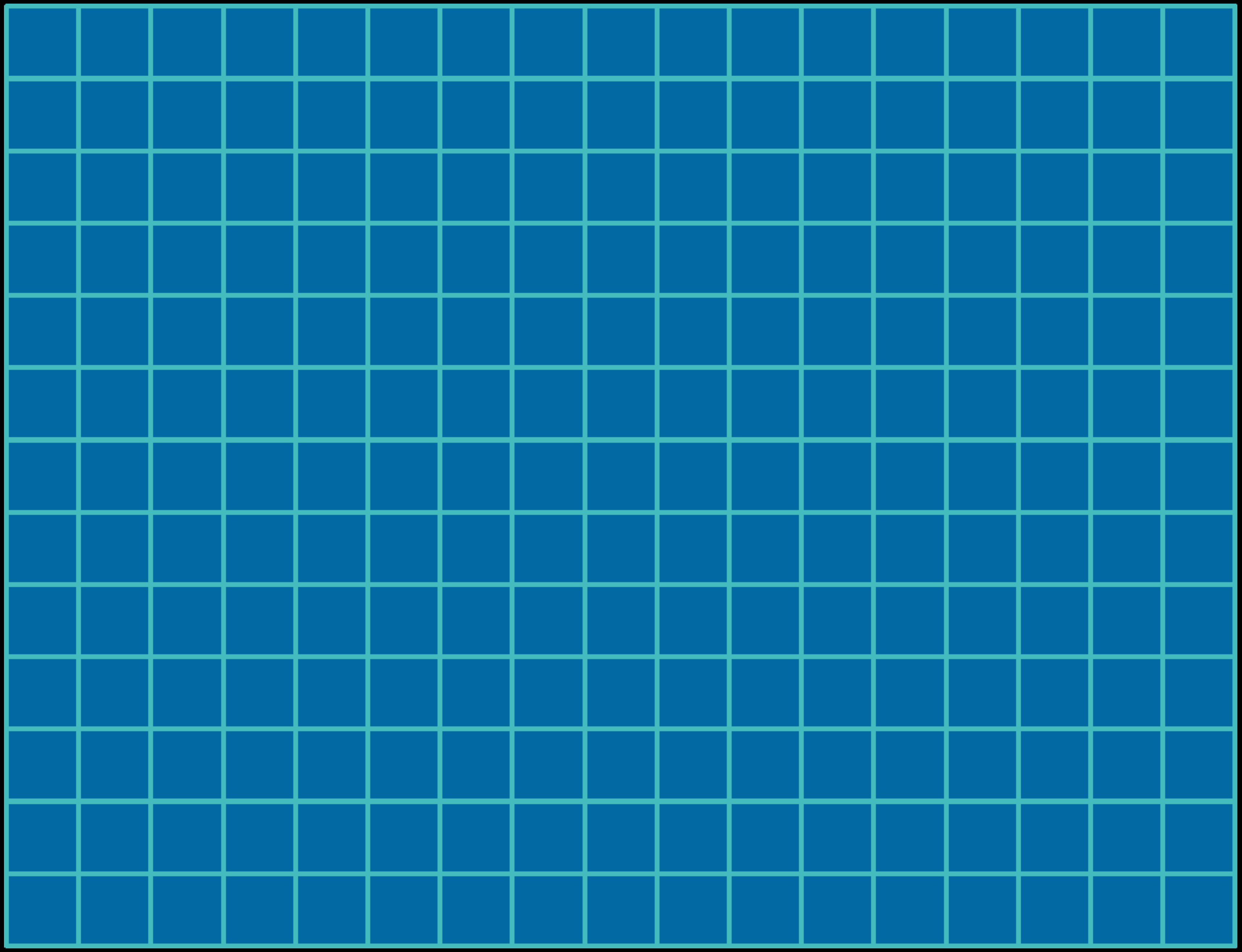
Adaptive Subdivision



Adaptive Subdivision

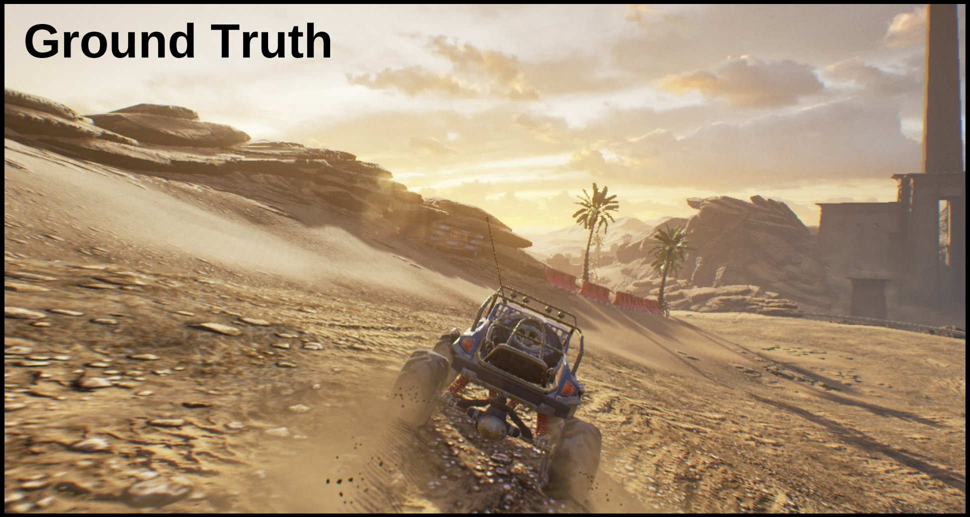


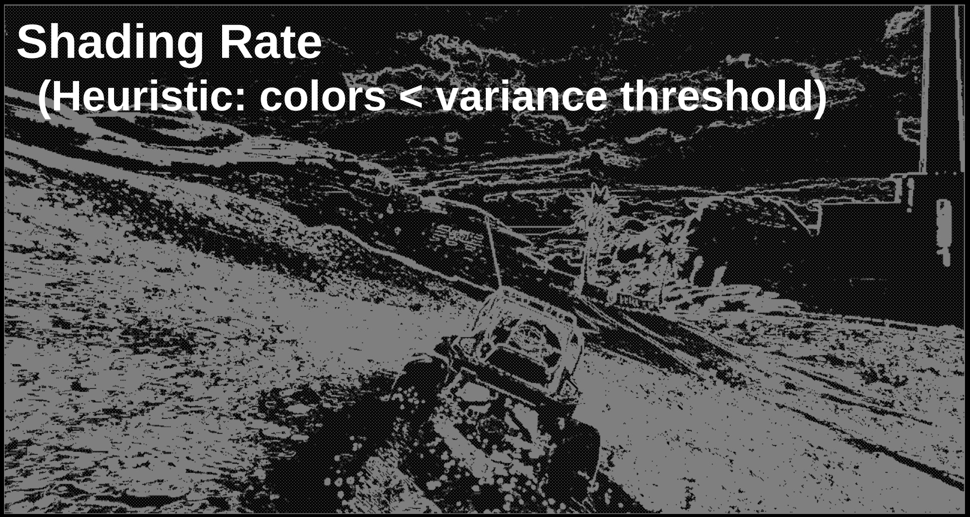
Adaptive Subdivision



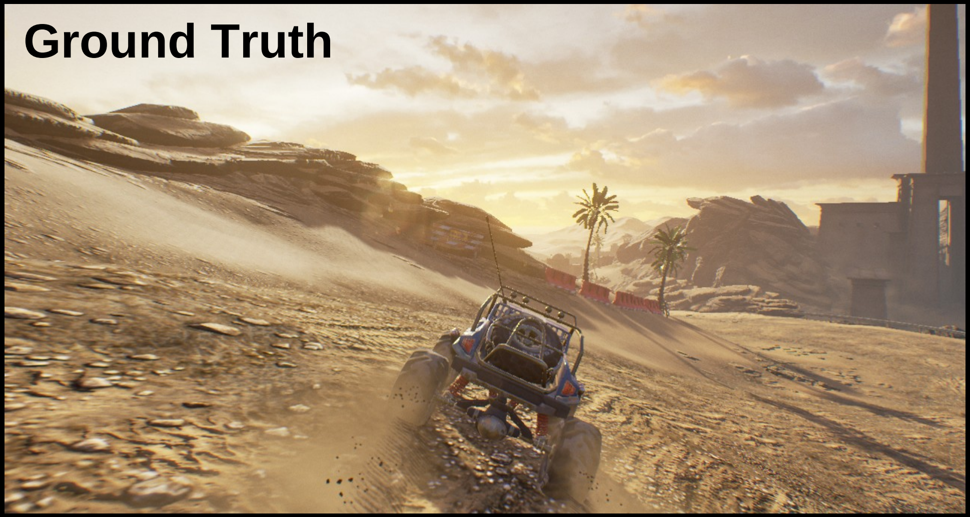
Adaptive Subdivision

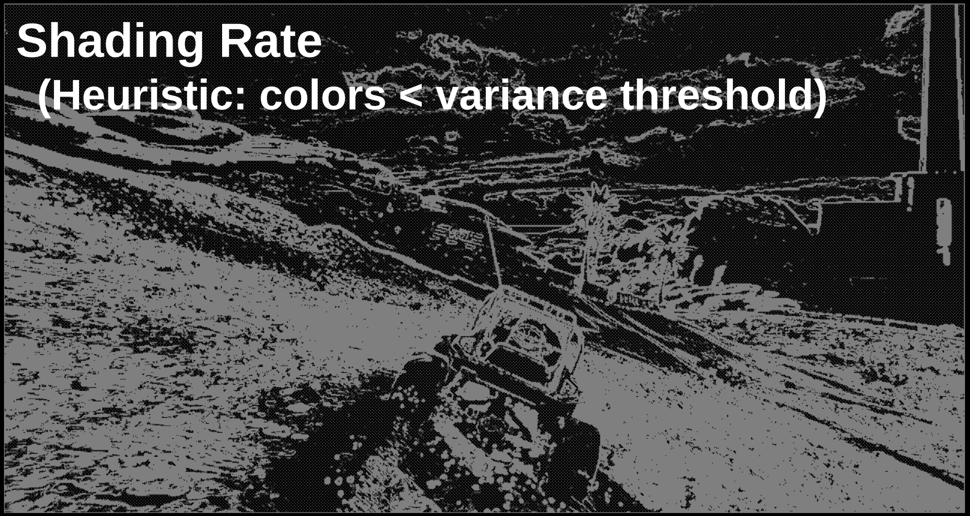
Adaptive Subdivision

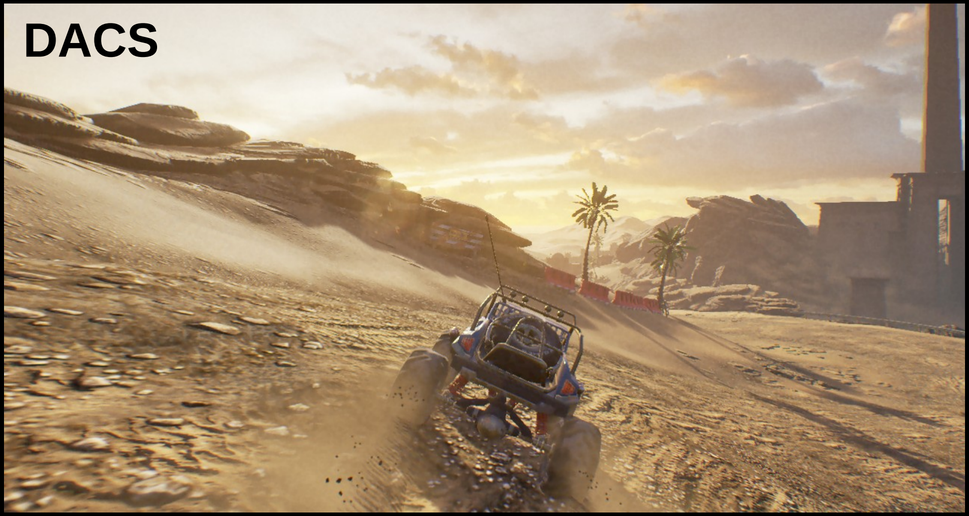




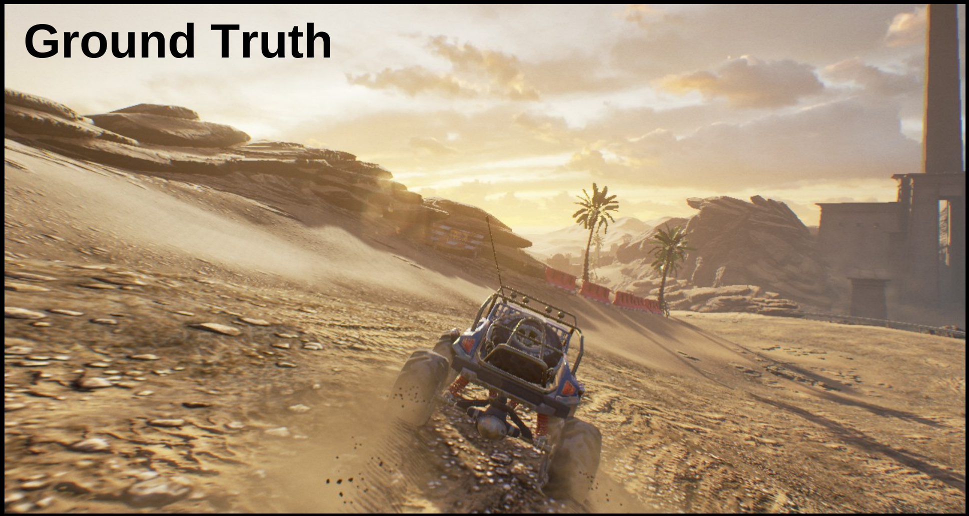
Adaptive Subdivision

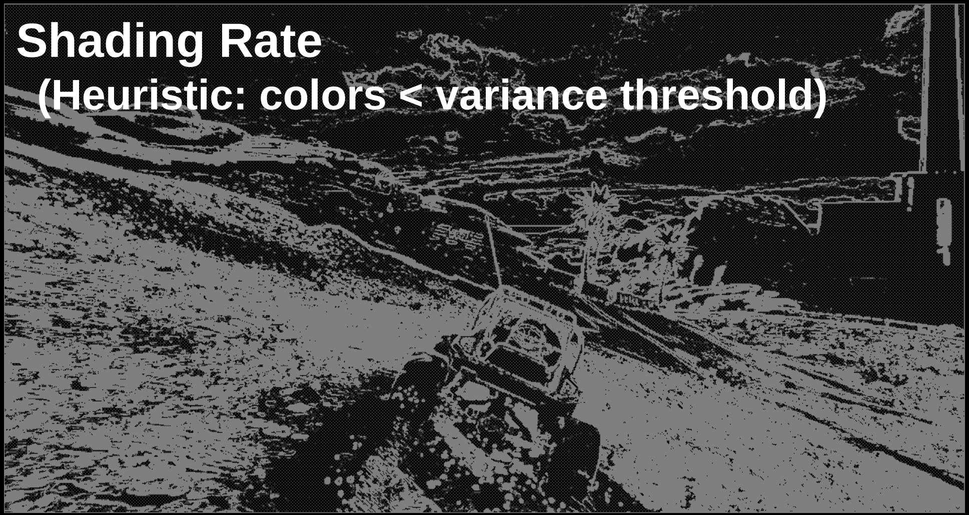


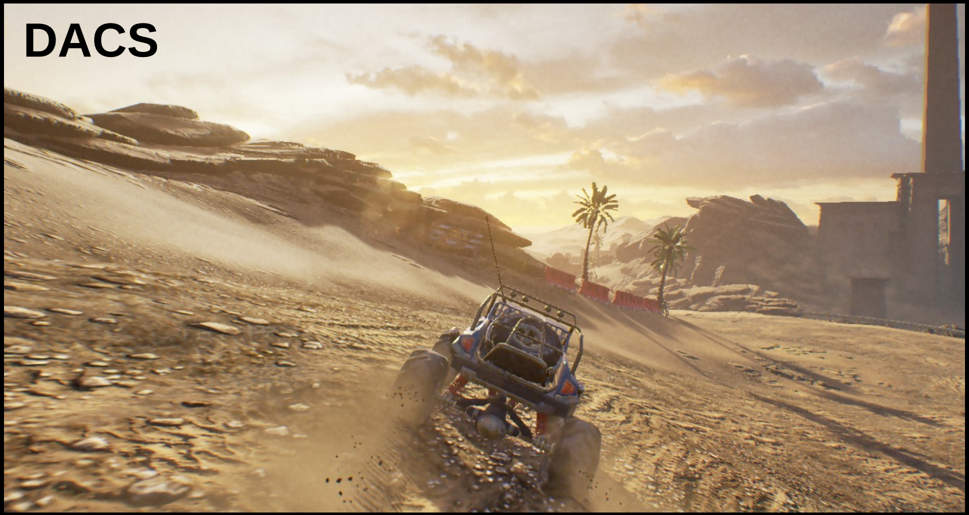


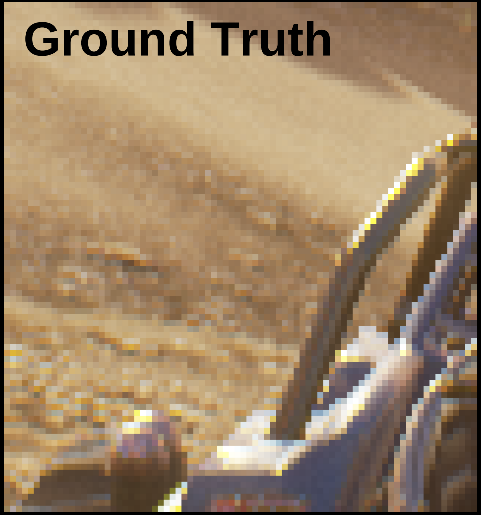


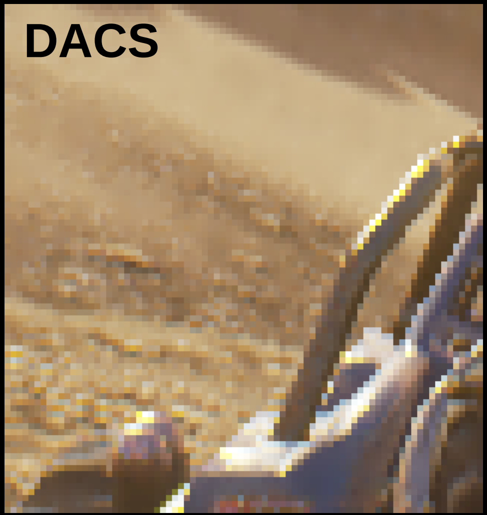
Adaptive Subdivision


Ground Truth


Shading Rate
(Heuristic: colors < variance threshold)


DACS


Ground Truth


DACS

Deferred Adaptive Compute Shading

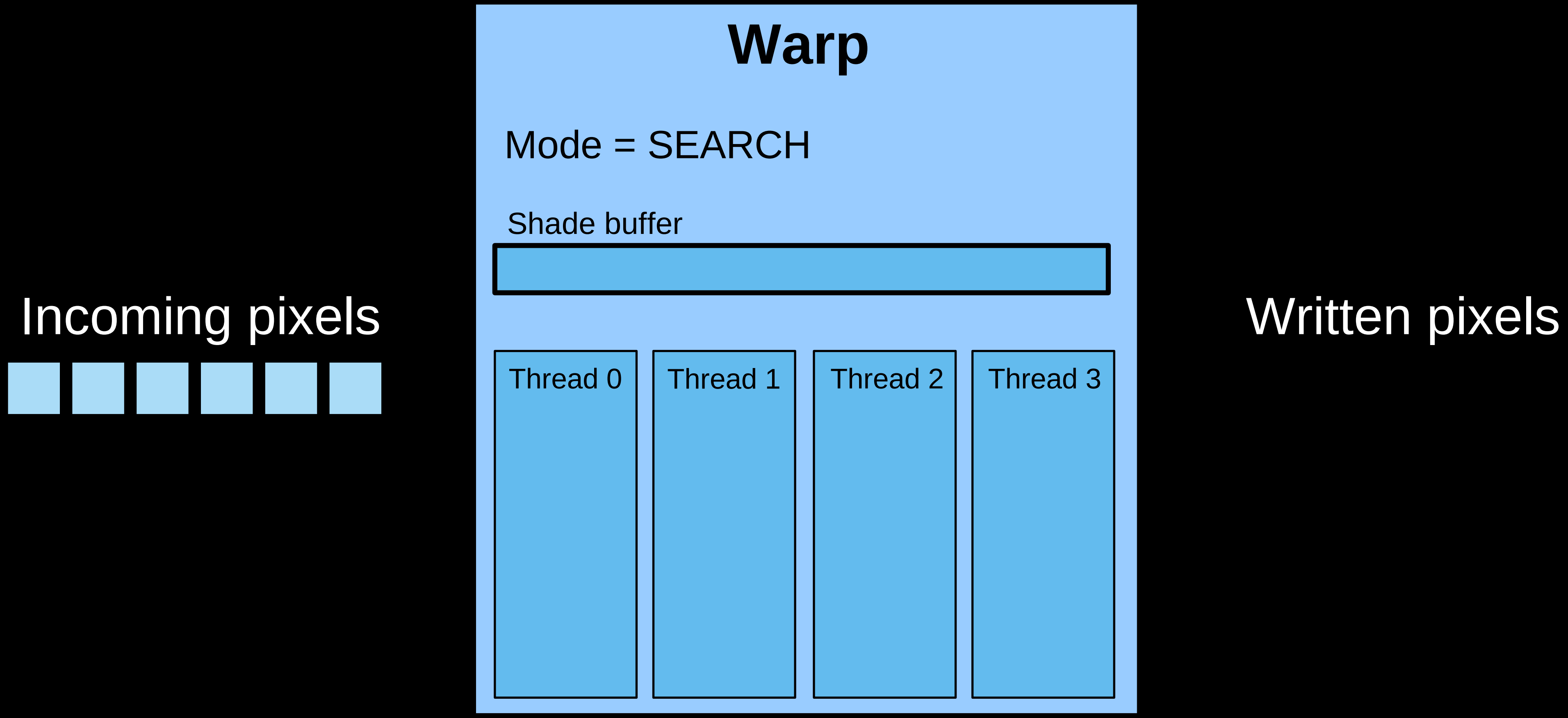
Implementation on Current GPUs

Warp Divergence

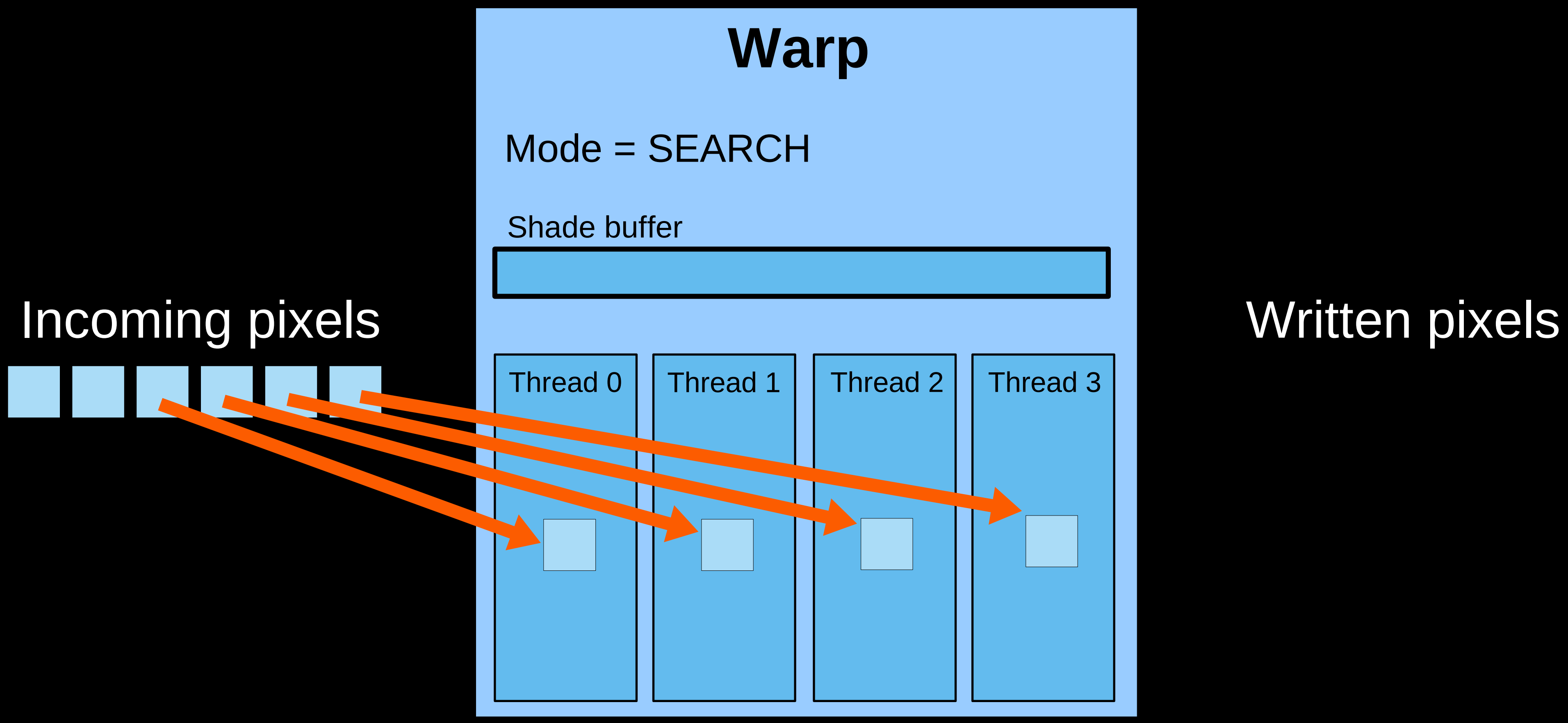
- Cannot skip pixels like this!
  - GPU still does the work (it’s just wasted)!

- Solution: warps switch between “search/interpolate” mode and “shade” mode

Mode Switching



Mode Switching



Warp

Mode = SEARCH

Shade buffer

Incoming pixels

Thread 0

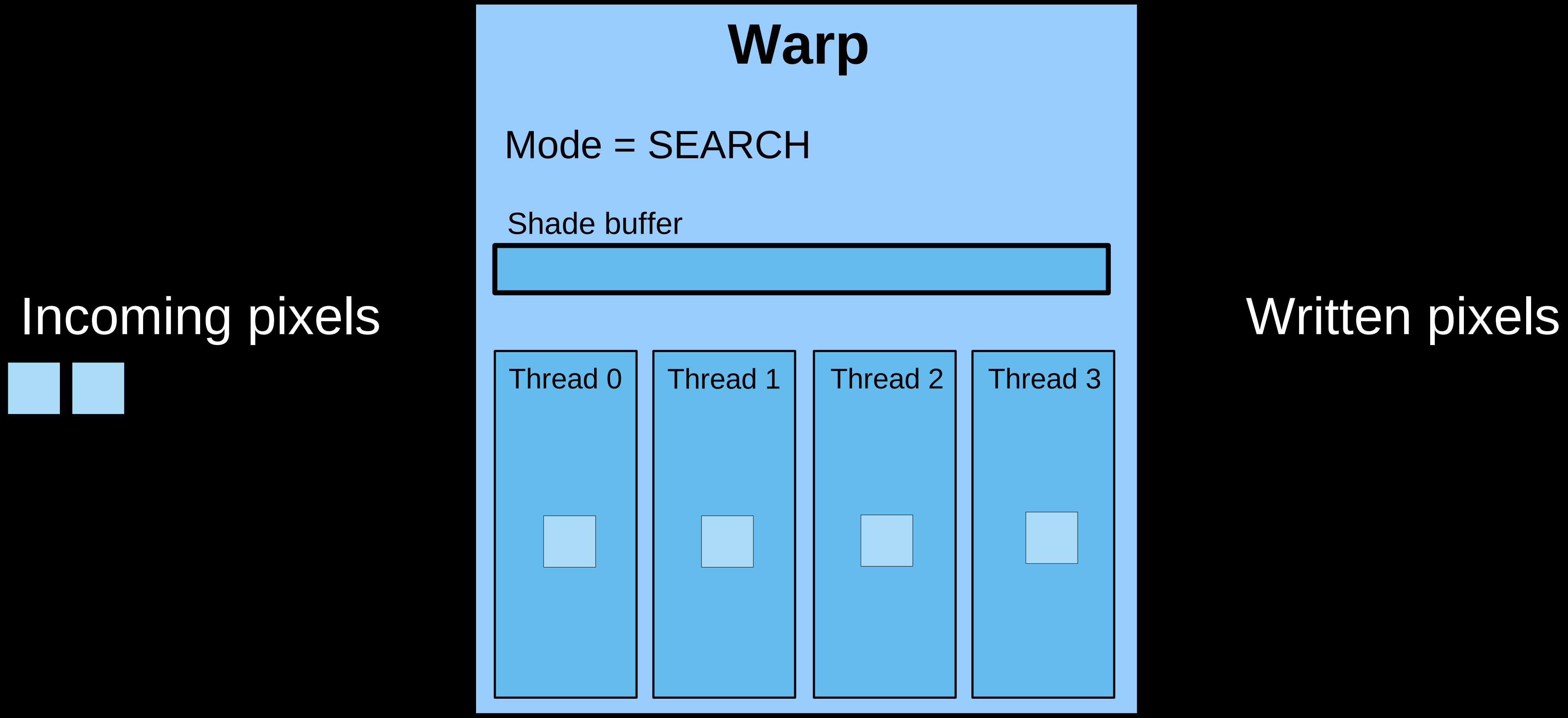
Thread 1

Thread 2

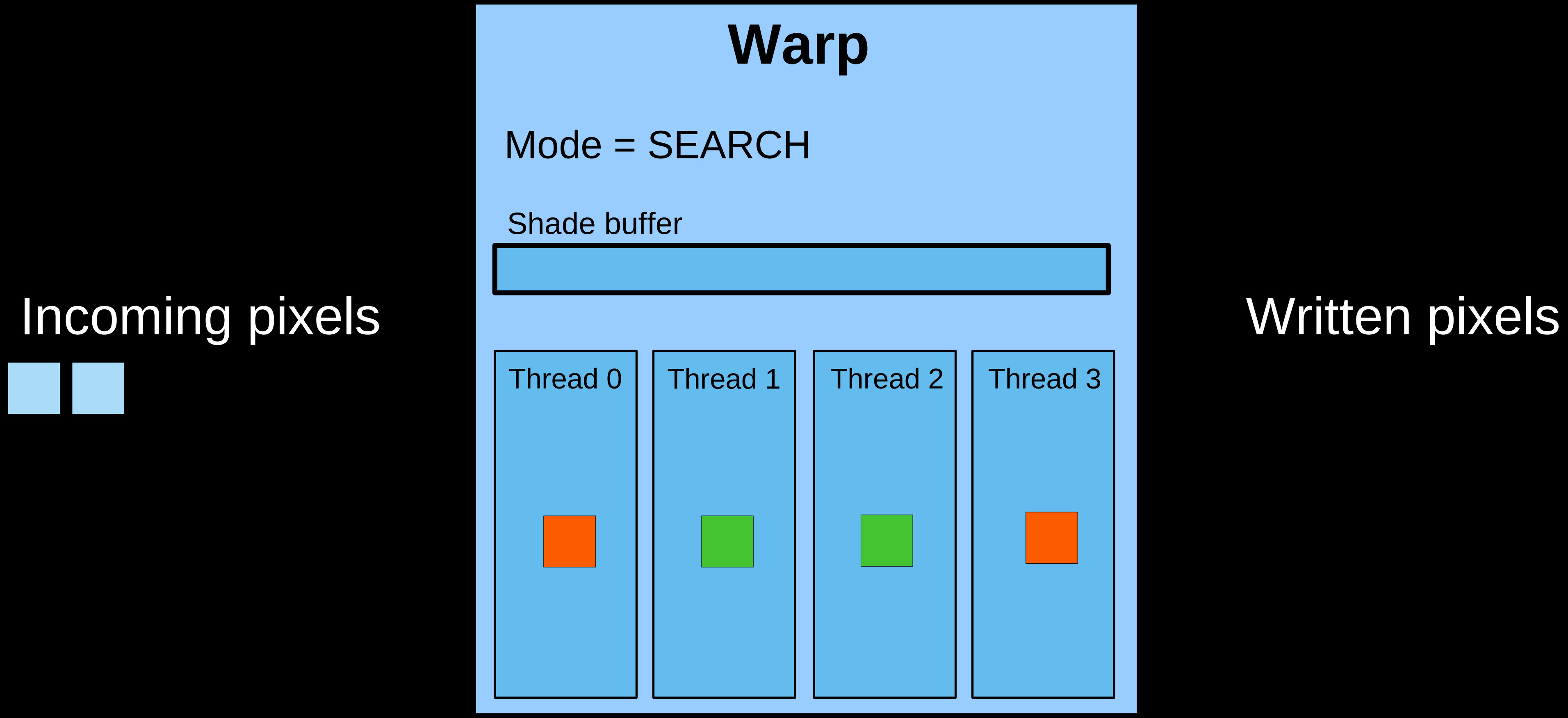
Thread 3

Written pixels

Mode Switching



Mode Switching



Warp

Mode = SEARCH

Shade buffer

Incoming pixels

Written pixels

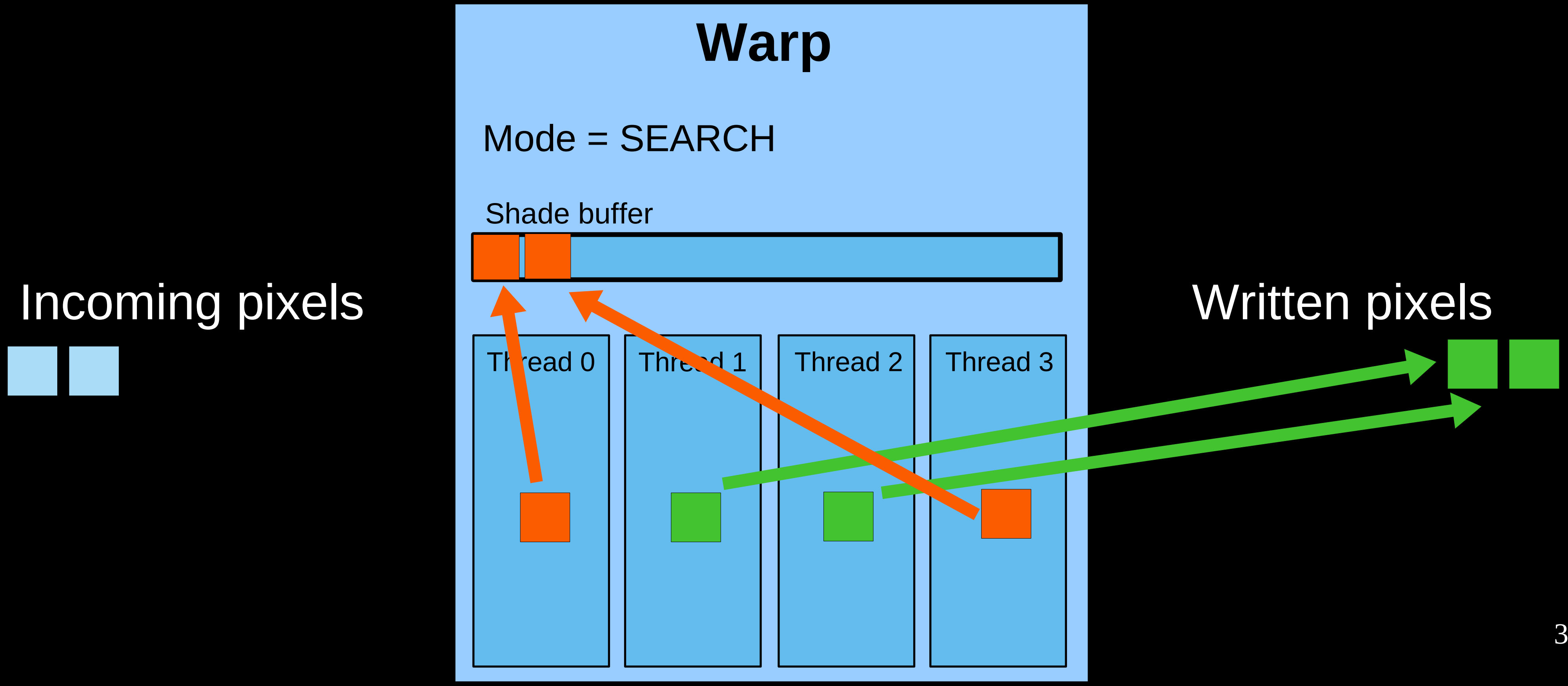
Thread 0

Thread 1

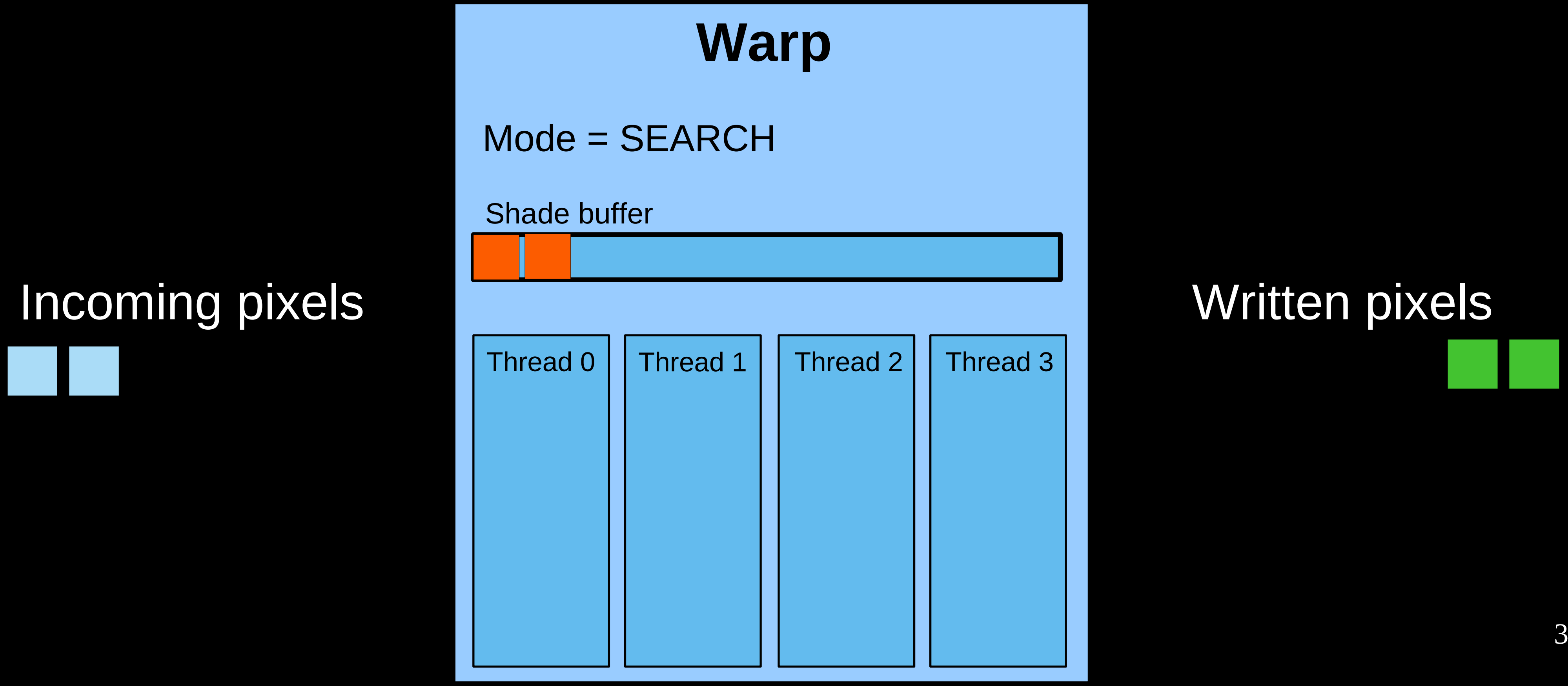
Thread 2

Thread 3

Mode Switching



Mode Switching



Warp

Mode = SEARCH

Shade buffer

Incoming pixels

Written pixels

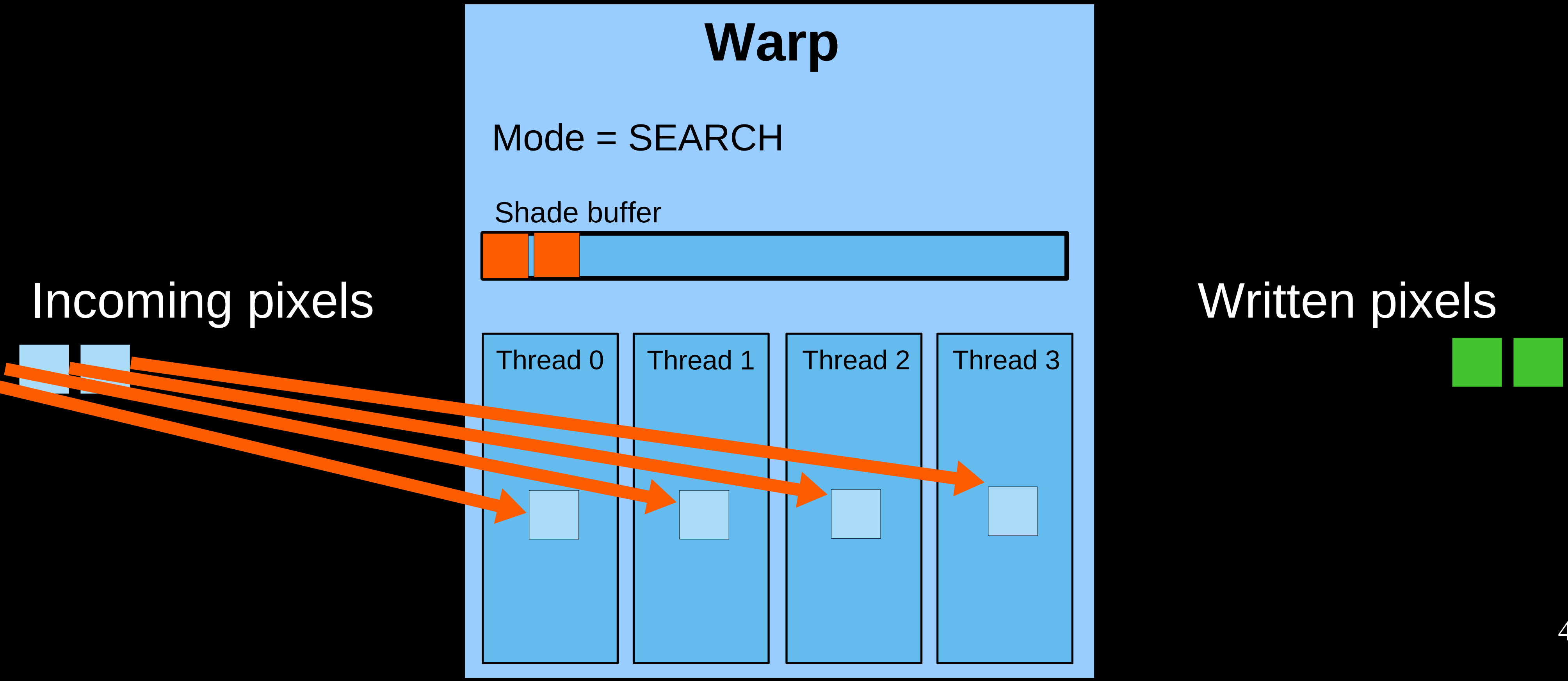
Thread 0

Thread 1

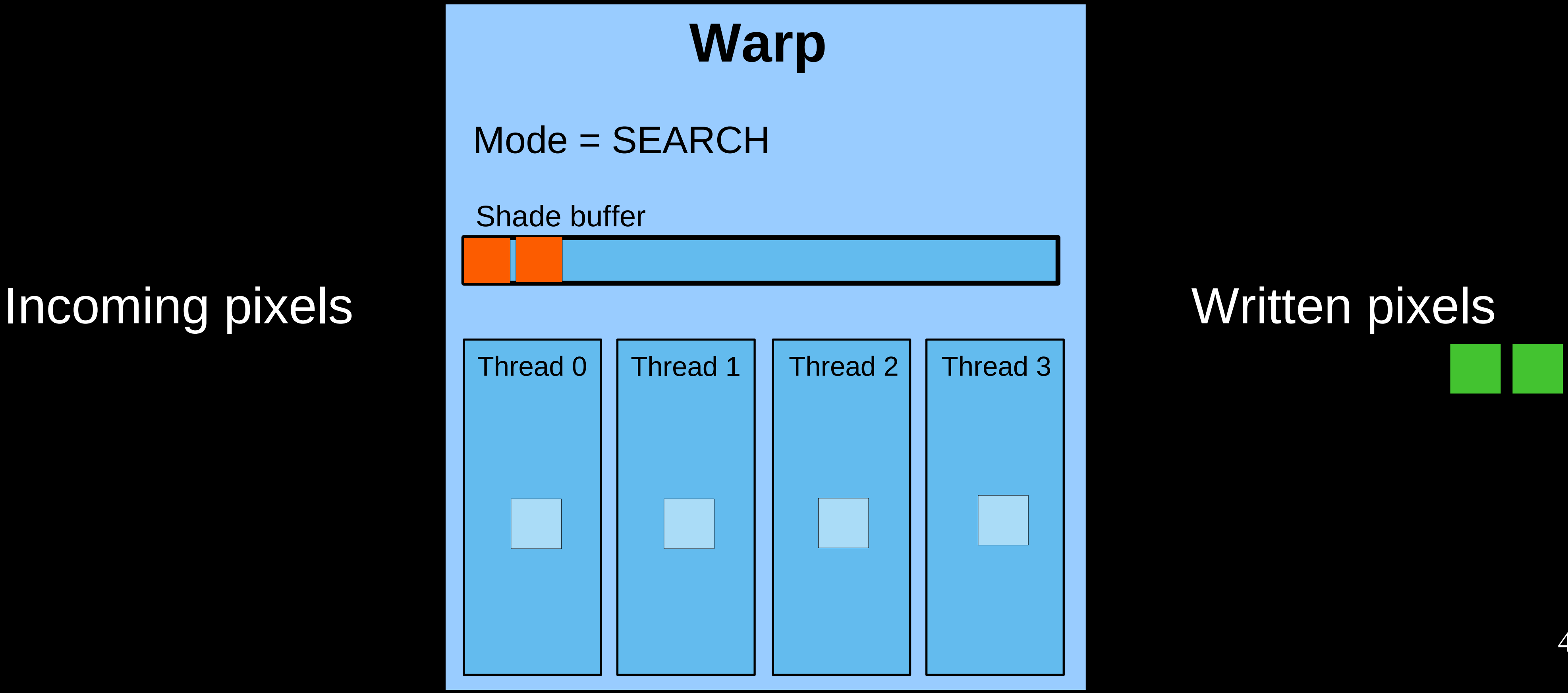
Thread 2

Thread 3

Mode Switching



Mode Switching



Incoming pixels

Warp

Mode = SEARCH

Shade buffer

Thread 0

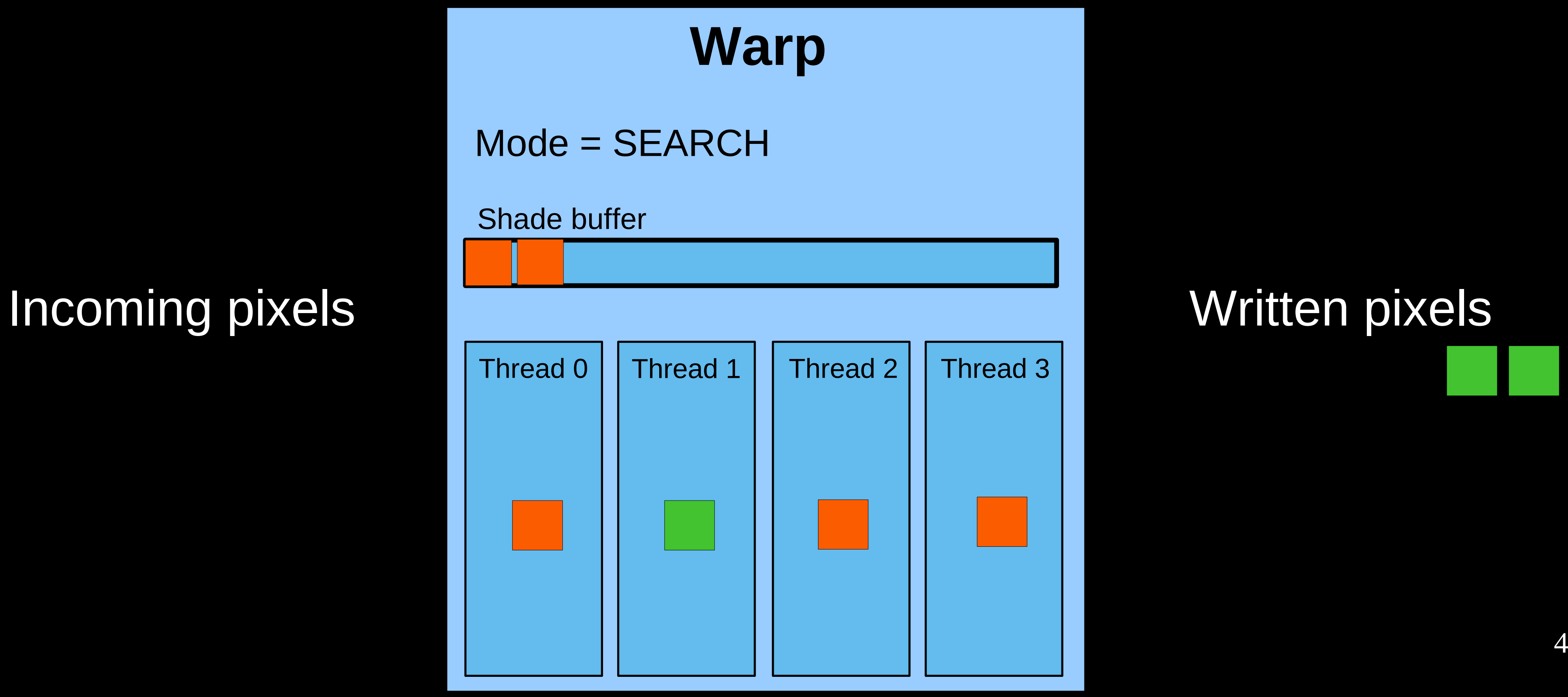
Thread 1

Thread 2

Thread 3

Written pixels

Mode Switching



Incoming pixels

Warp

Mode = SEARCH

Shade buffer

Thread 0

Thread 1

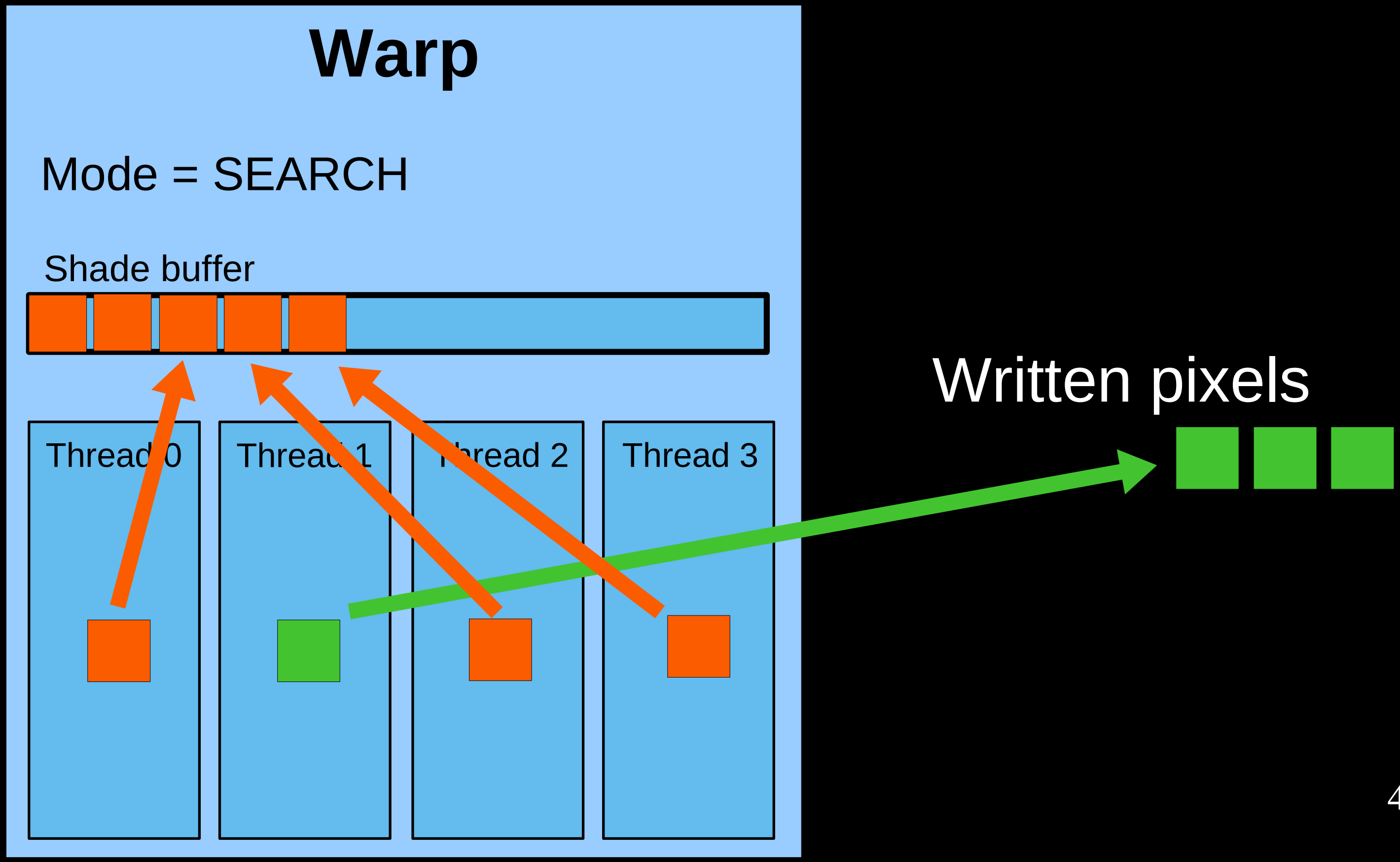
Thread 2

Thread 3

Written pixels

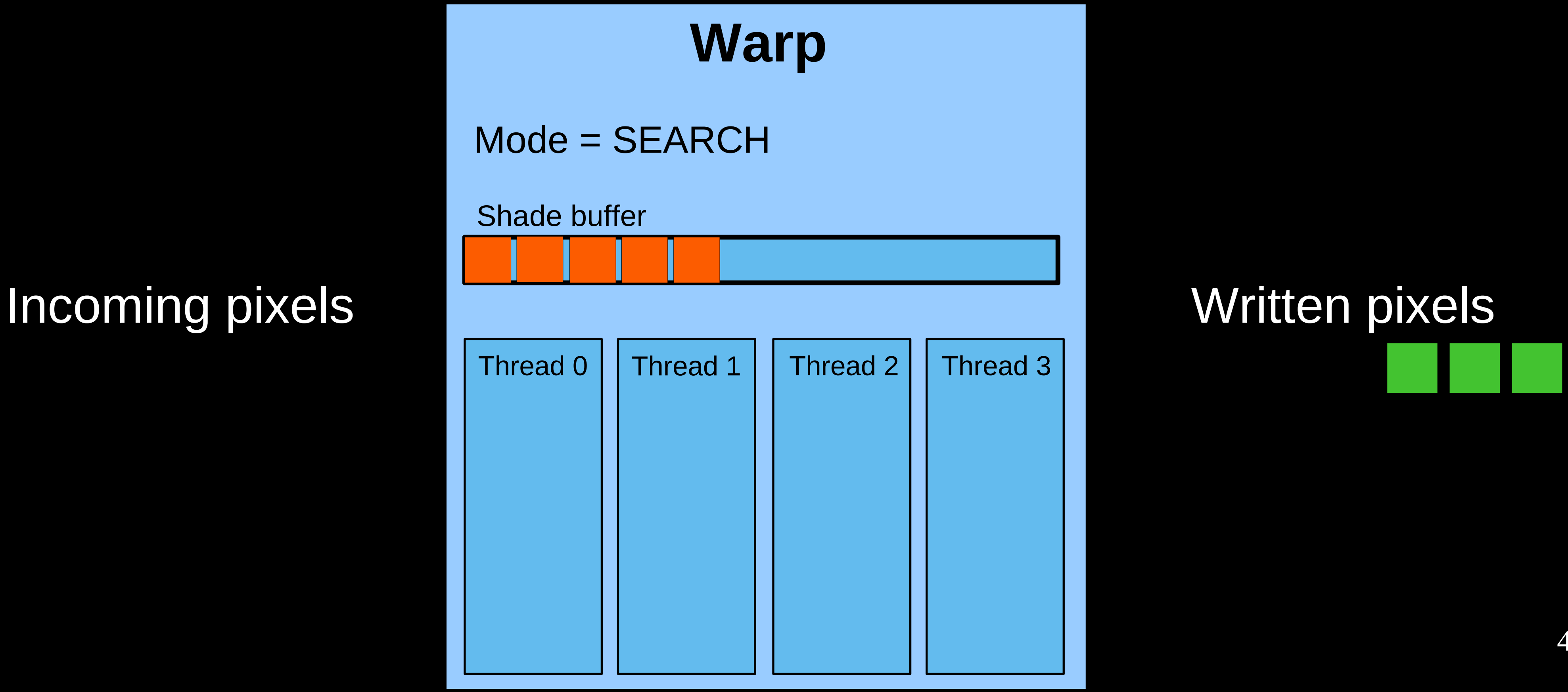
Mode Switching

Incoming pixels



Written pixels

Mode Switching



Incoming pixels

Warp

Mode = SEARCH

Shade buffer

Thread 0

Thread 1

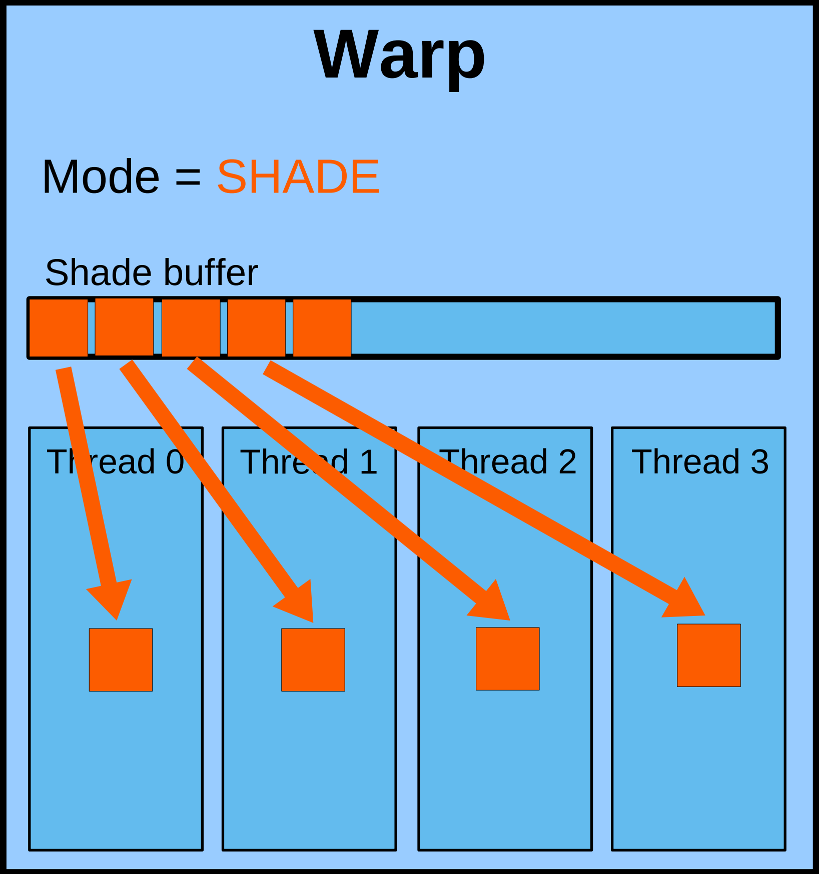
Thread 2

Thread 3

Written pixels

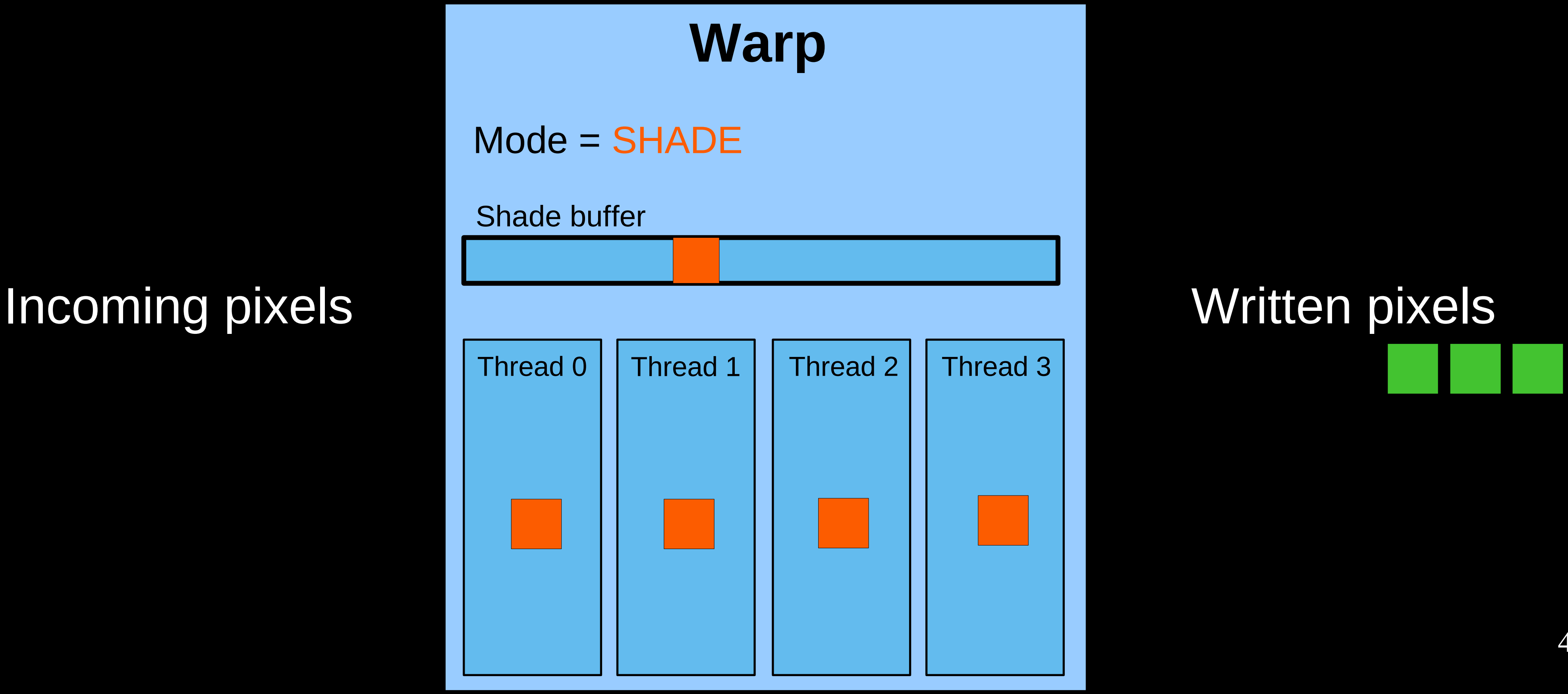
Mode Switching

Incoming pixels



Written pixels

Mode Switching



Warp

Mode = SHADE

Shade buffer

Incoming pixels

Thread 0

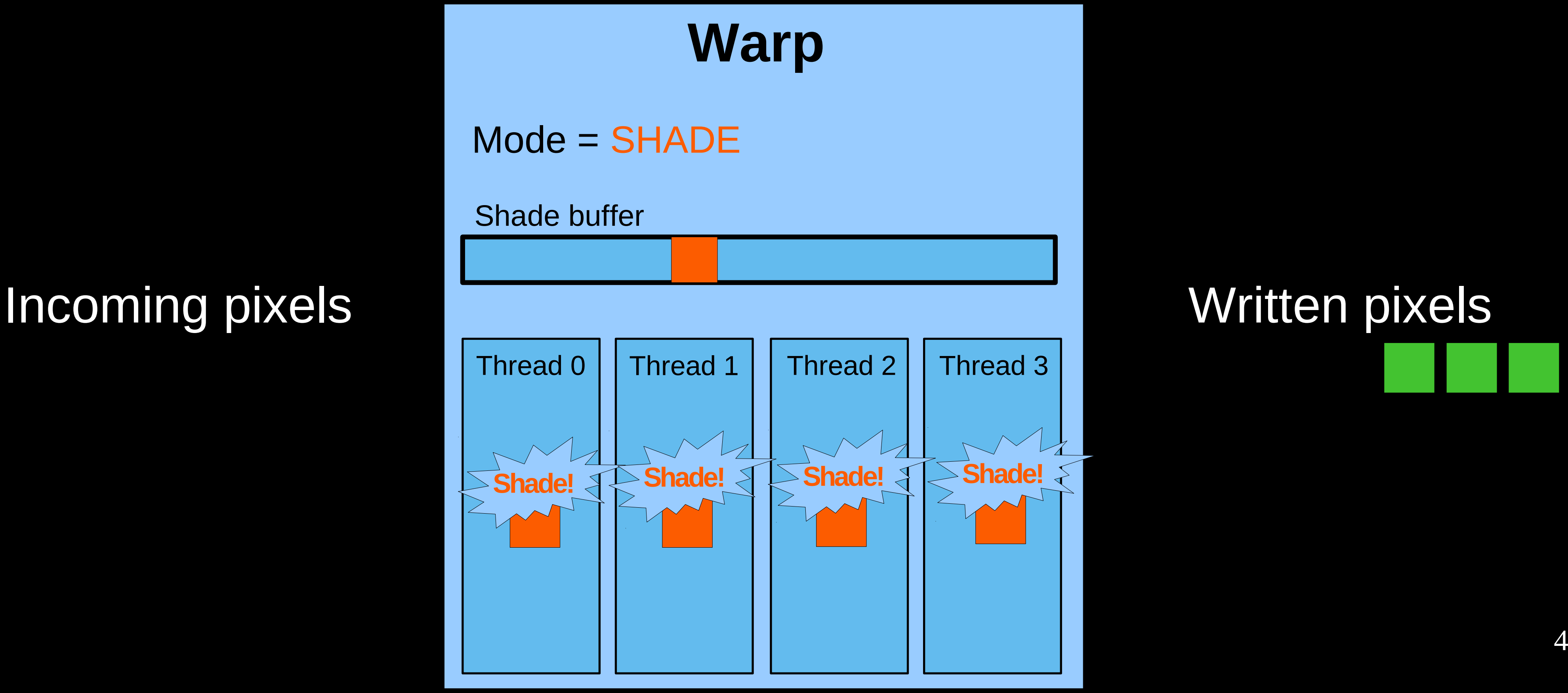
Thread 1

Thread 2

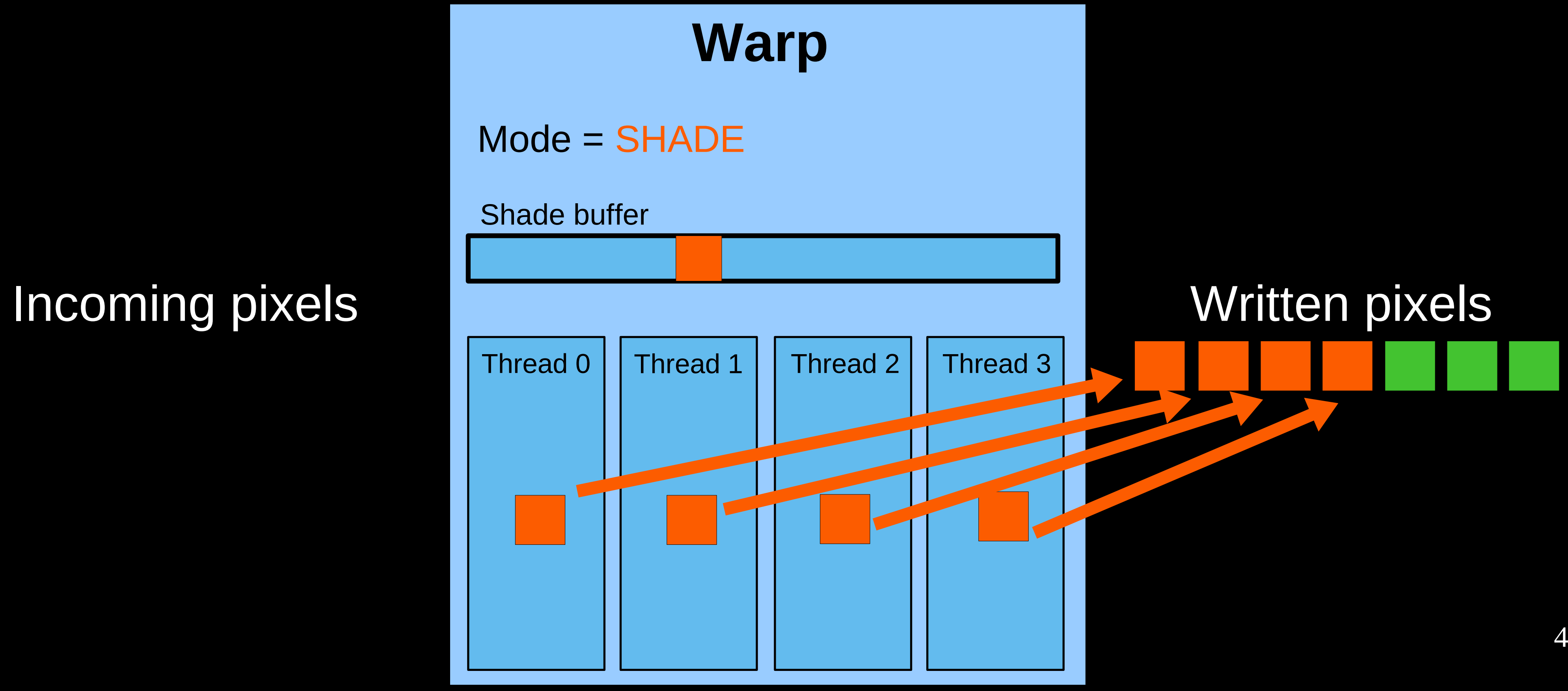
Thread 3

Written pixels

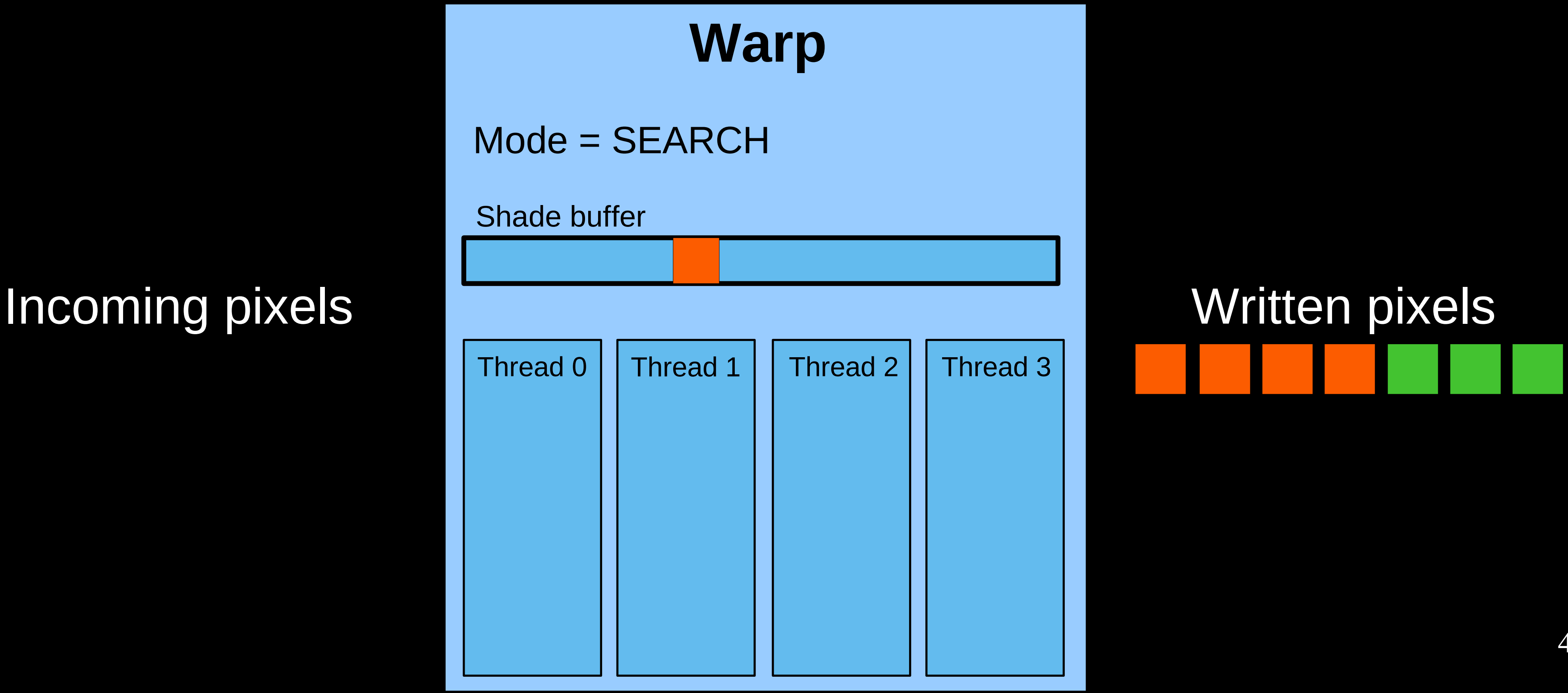
Mode Switching



Mode Switching



Mode Switching



Incoming pixels

Warp

Mode = SEARCH

Shade buffer

Thread 0

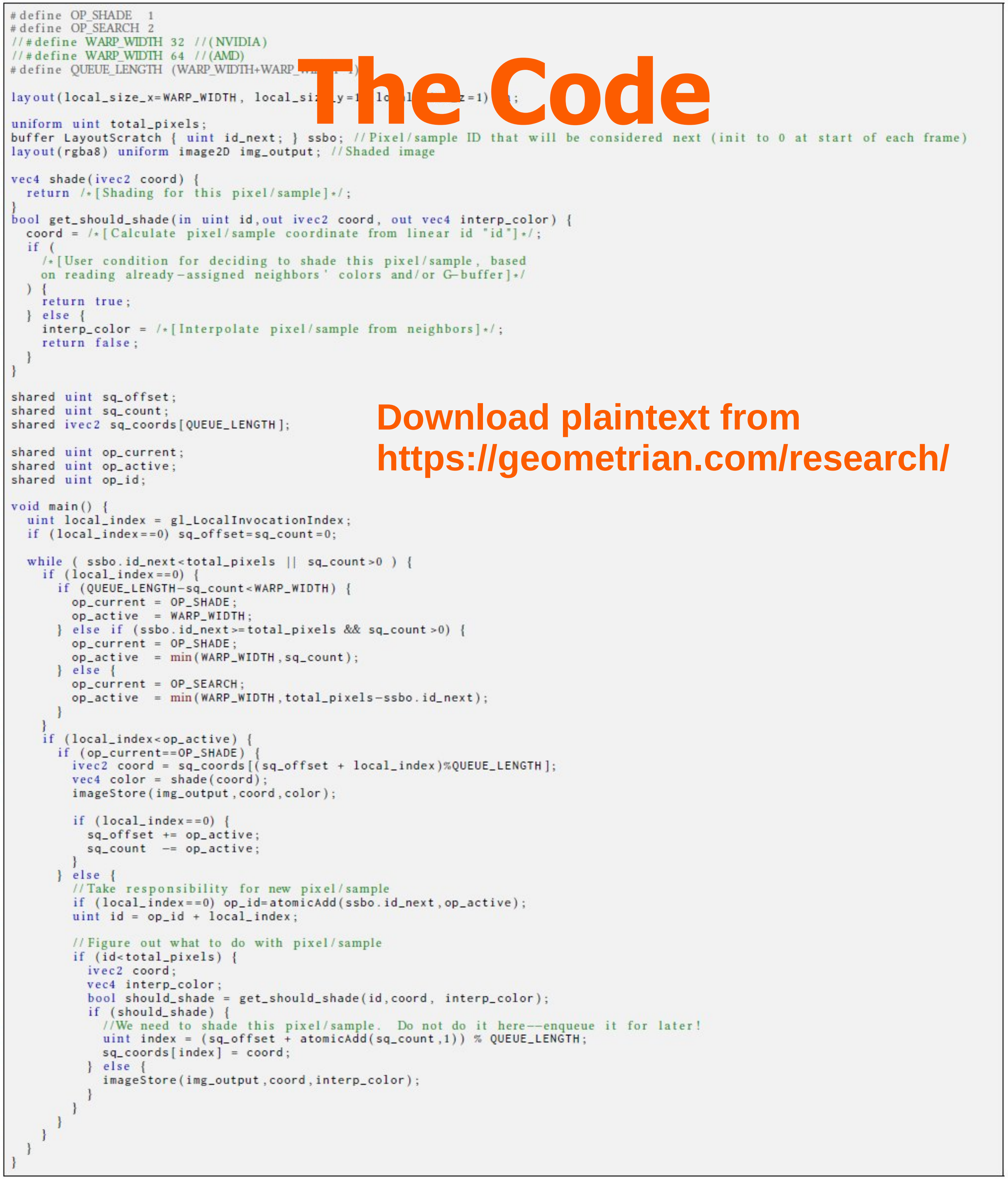
Thread 1

Thread 2

Thread 3

Written pixels

The Code



```
#define OP_SHADE  1
#define OP_SEARCH 2
//#define WARP_WIDTH 32 //(NVIDIA)
//#define WARP_WIDTH 64 //(AMD)
#define QUEUE_LENGTH (WARP_WIDTH+WARP_WIDTH-1)

layout(local_size_x=WARP_WIDTH, local_size_y=1, local_size_z=1) in;

uniform uint total_pixels;
buffer LayoutScratch { uint id_next; } ssbo; //Pixel/sample ID that will be considered next (init to 0 at start of each frame)
layout(rgba8) uniform image2D img_output; //Shaded image

vec4 shade(ivec2 coord) {
  return /*[Shading for this pixel/sample]*/;
}
bool get_should_shade(in uint id,out ivec2 coord, out vec4 interp_color) {
  coord = /*[Calculate pixel/sample coordinate from linear id "id"]*/;
  if (
    /*[User condition for deciding to shade this pixel/sample, based
    on reading already-assigned neighbors' colors and/or G-buffer]*/
  ) {
    return true;
  } else {
    interp_color = /*[Interpolate pixel/sample from neighbors]*/;
    return false;
  }
}

shared uint sq_offset;
shared uint sq_count;
shared ivec2 sq_coords[QUEUE_LENGTH];

shared uint op_current;
shared uint op_active;
shared uint op_id;

void main() {
  uint local_index = gl_LocalInvocationIndex;
  if (local_index==0) sq_offset=sq_count=0;

  while ( ssbo.id_next<total_pixels || sq_count>0 ) {
    if (local_index==0) {
      if (QUEUE_LENGTH-sq_count<WARP_WIDTH) {
        op_current = OP_SHADE;
        op_active  = WARP_WIDTH;
      } else if (ssbo.id_next>=total_pixels && sq_count>0) {
        op_current = OP_SHADE;
        op_active  = min(WARP_WIDTH,sq_count);
      } else {
        op_current = OP_SEARCH;
        op_active  = min(WARP_WIDTH,total_pixels-ssbo.id_next);
      }
    }
    if (local_index<op_active) {
      if (op_current==OP_SHADE) {
        ivec2 coord = sq_coords[(sq_offset + local_index)%QUEUE_LENGTH];
        vec4 color = shade(coord);
        imageStore(img_output,coord,color);

        if (local_index==0) {
          sq_offset += op_active;
          sq_count  -= op_active;
        }
      } else {
        //Take responsibility for new pixel/sample
        if (local_index==0) op_id=atomicAdd(ssbo.id_next,op_active);
        uint id = op_id + local_index;

        //Figure out what to do with pixel/sample
        if (id<total_pixels) {
          ivec2 coord;
          vec4 interp_color;
          bool should_shade = get_should_shade(id,coord, interp_color);
          if (should_shade) {
            //We need to shade this pixel/sample. Do not do it here--enqueue it for later!
            uint index = (sq_offset + atomicAdd(sq_count,1)) % QUEUE_LENGTH;
            sq_coords[index] = coord;
          } else {
            imageStore(img_output,coord,interp_color);
          }
        }
      }
    }
  }
}
```

Download plaintext from https://geometrian.com/research/

Results

- Ran color-only simulated results on synthetic images

- Implemented in my deferred renderer
  - Comparison to simple checkerboard implementation
  - Note: no temporal filtering in any algorithm!

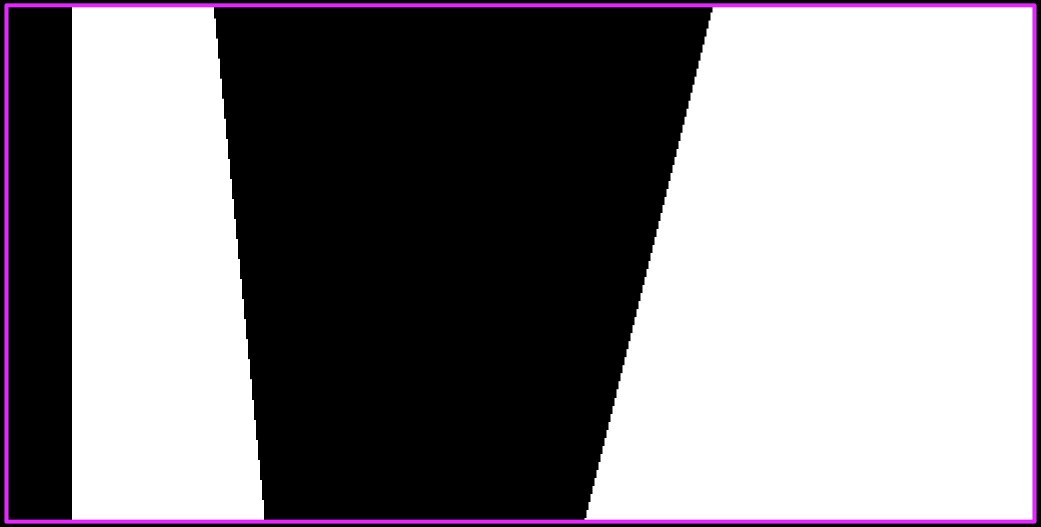
- Ran color-only simulated results on Unreal Engine frames (see video).
  - Timing not meaningful

Results: Synthetic Images

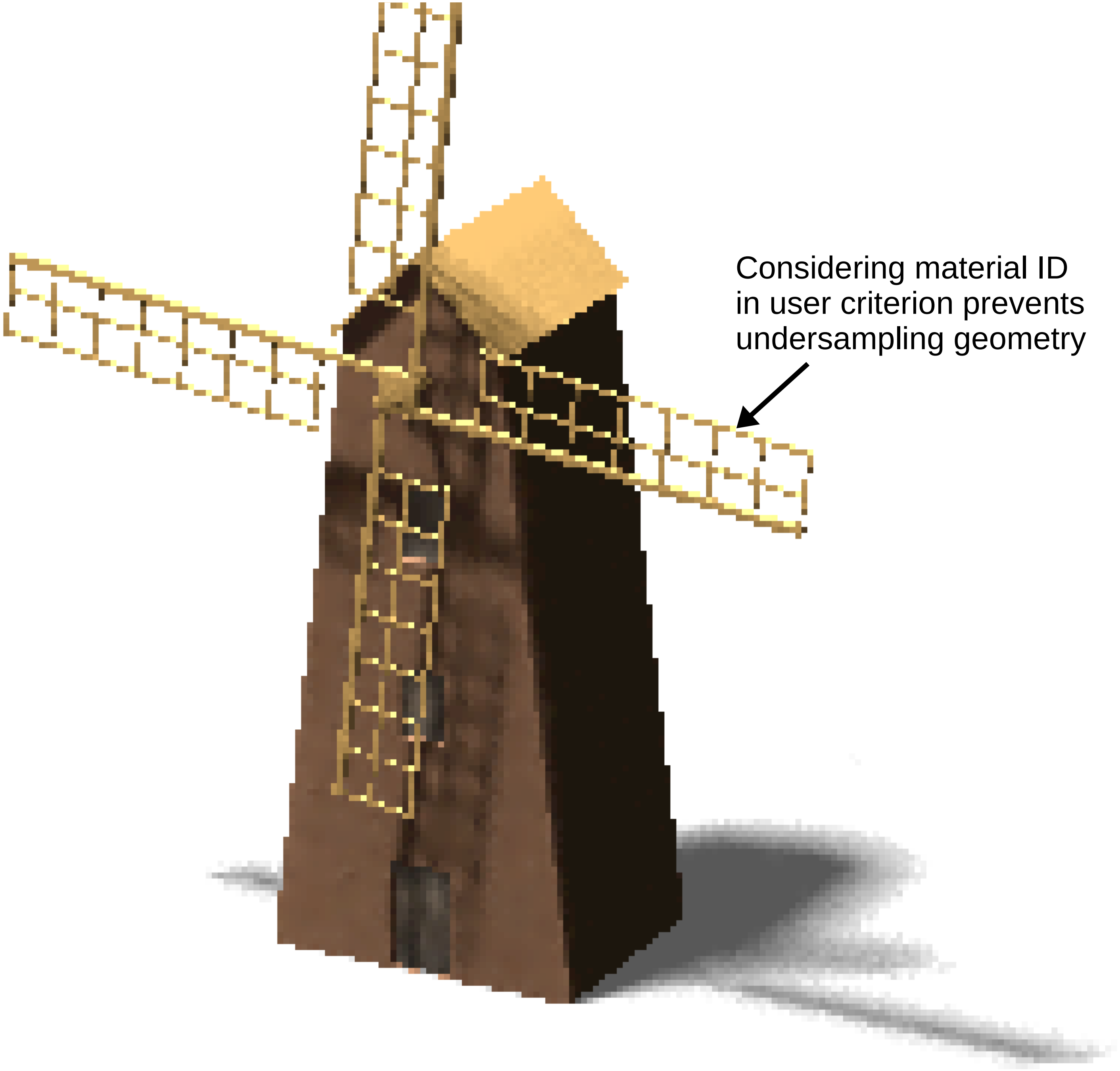
- Perfectly reconstructs gradient and step functions

- Gradient: characteristic of soft shadows, shaded regions

- Step: texture features, depth discontinuities

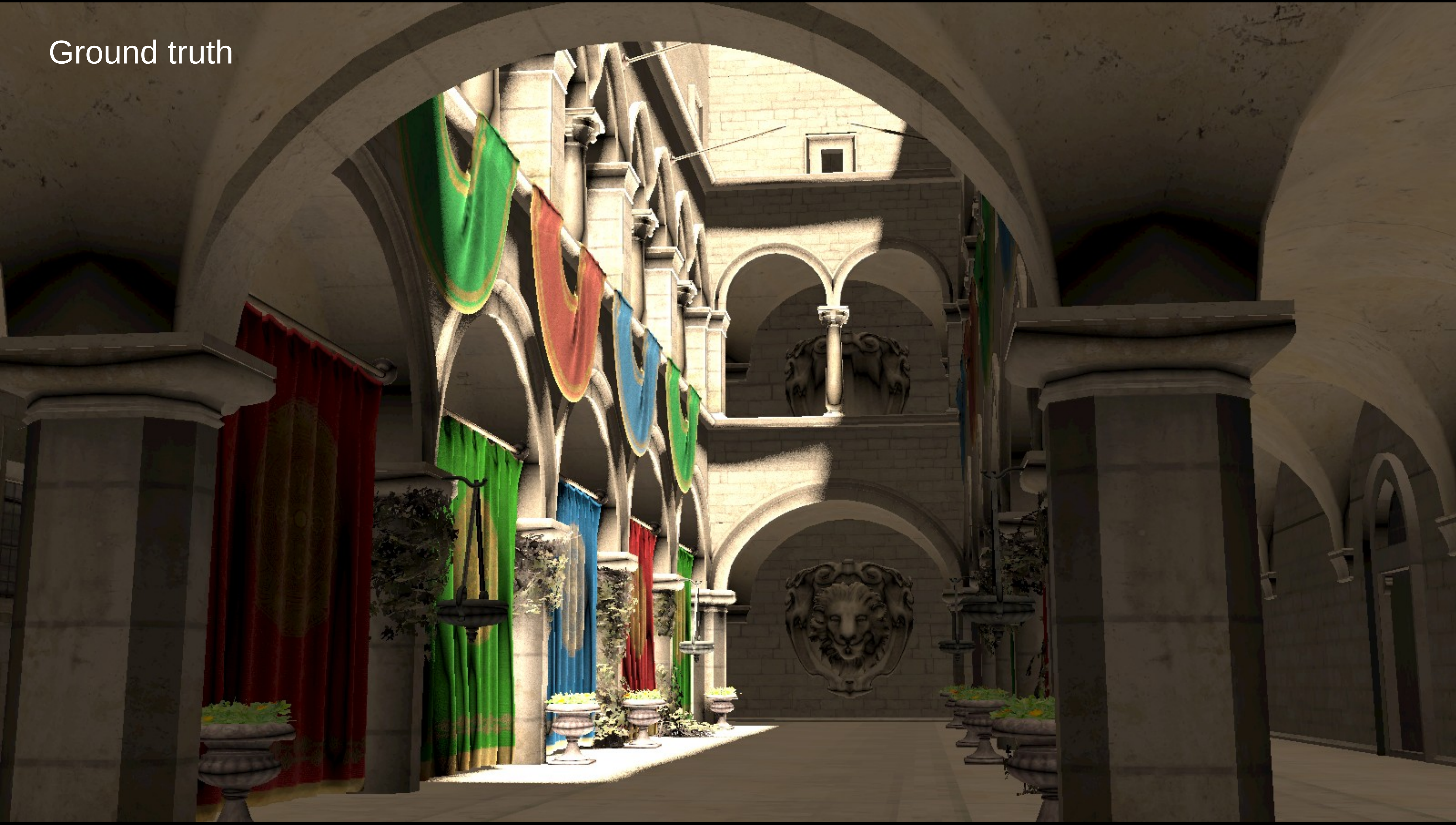


Results: Thin Features

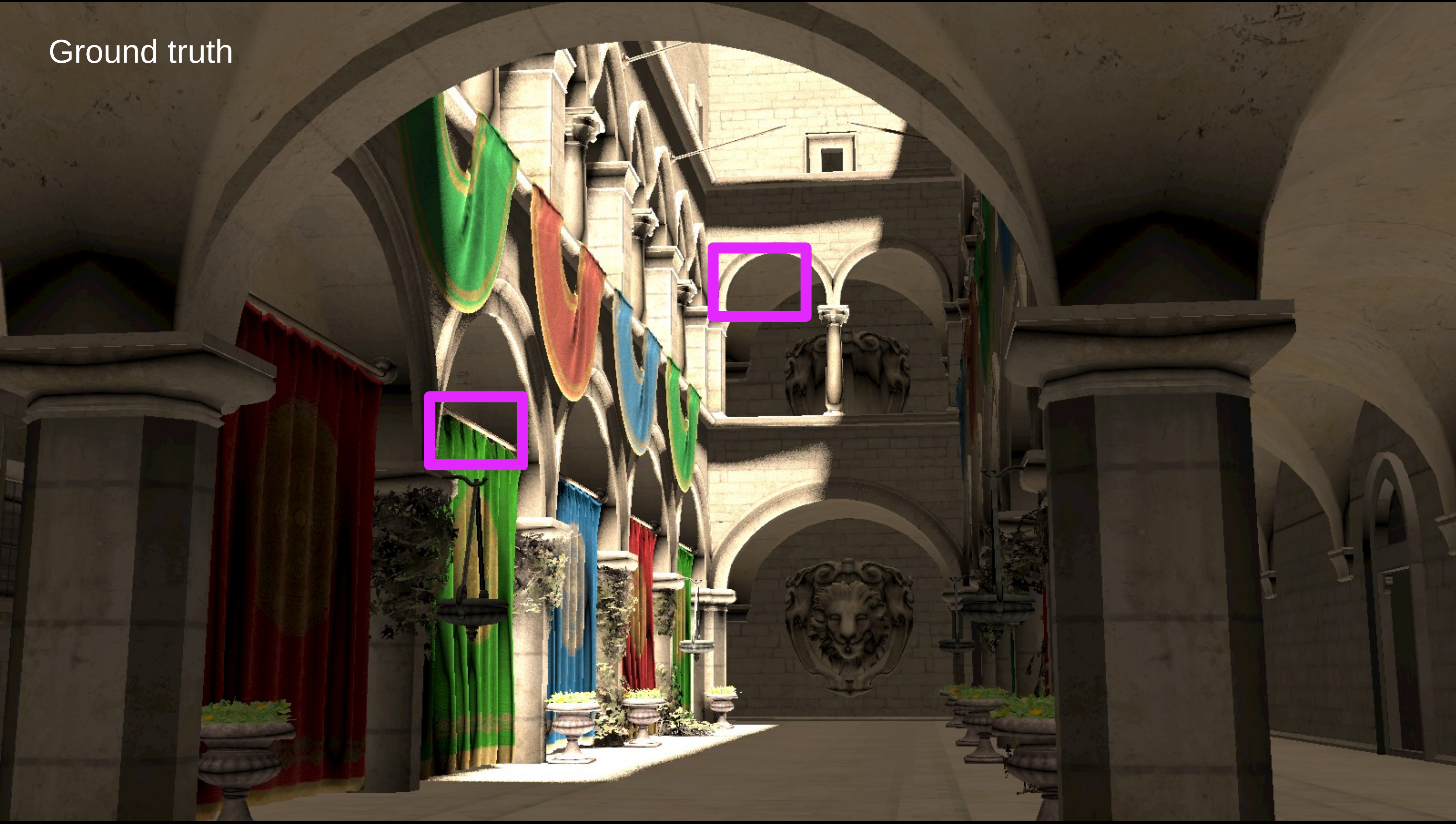


Considering material ID in user criterion prevents undersampling geometry

Results: Deferred Renderer

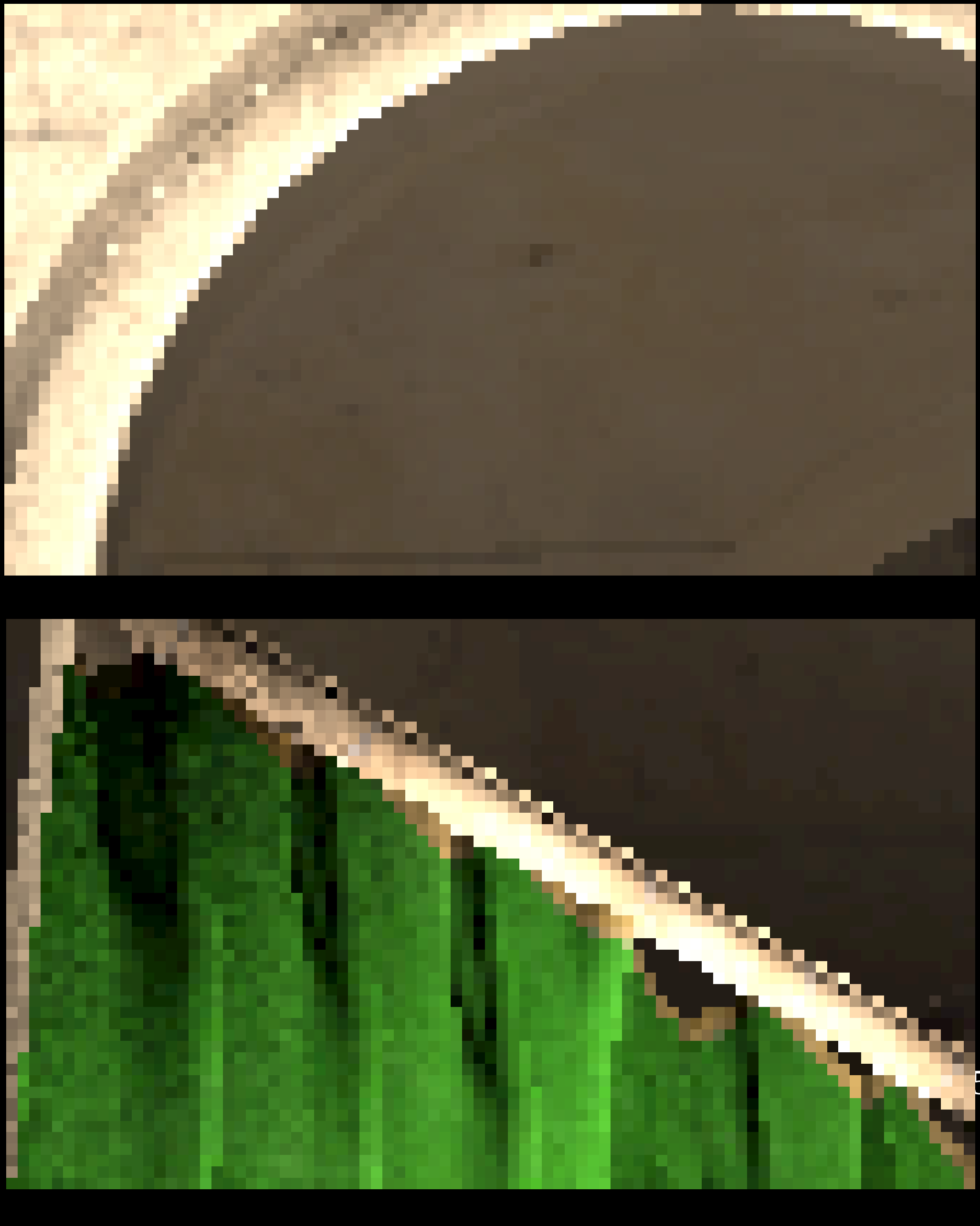


Results: Deferred Renderer



Results: Deferred Renderer

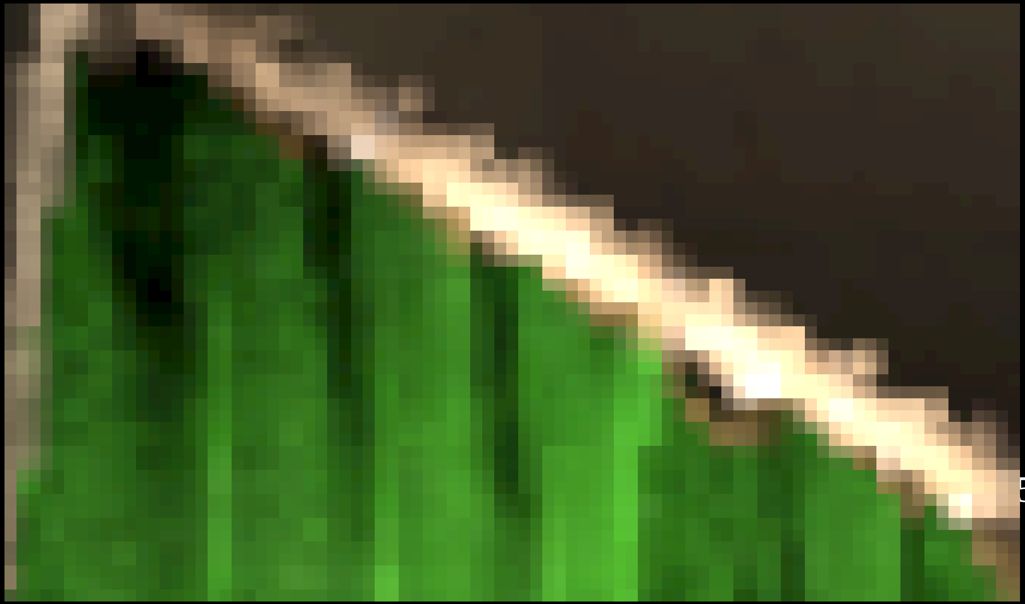
Ground Truth



Results: Deferred Renderer

Checkerboard

- Loss of detail in a single frame
- 1.89× speedup



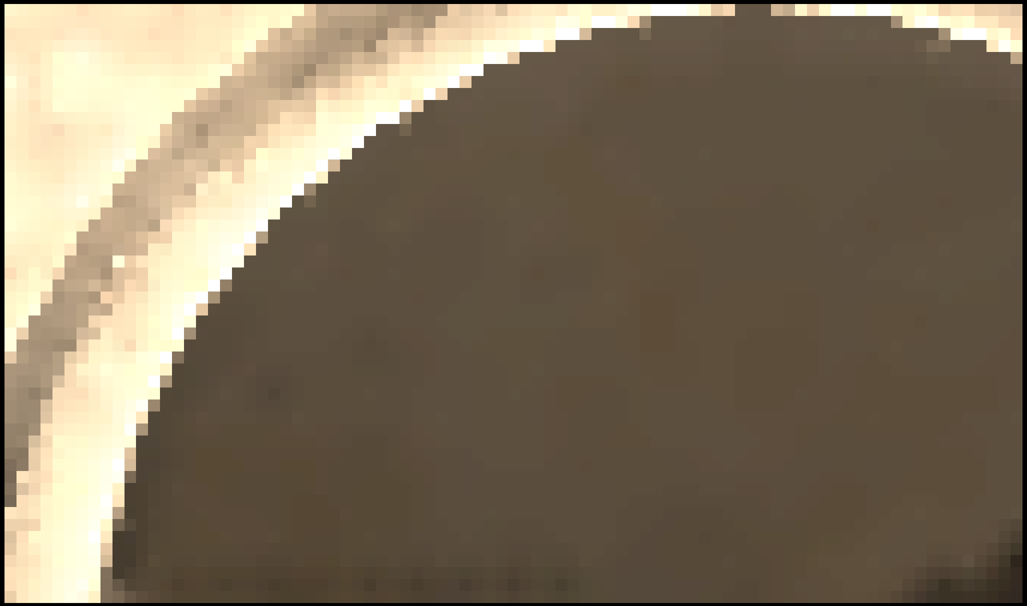
RMSE: 0.04218
PSNR: 27.53
MSSIM: 0.8954

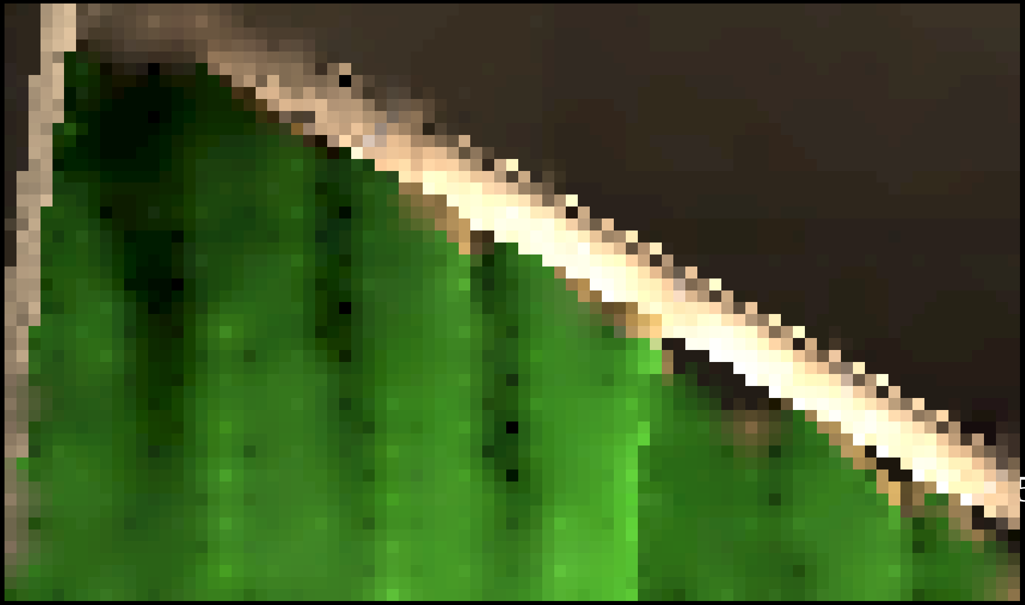
Speedup: 1.89× to GT

Results: Deferred Renderer

DACS

- “Equal” quality to checkerboard (same MSSIM)
- Better edge resolution
- 4.22× speedup!





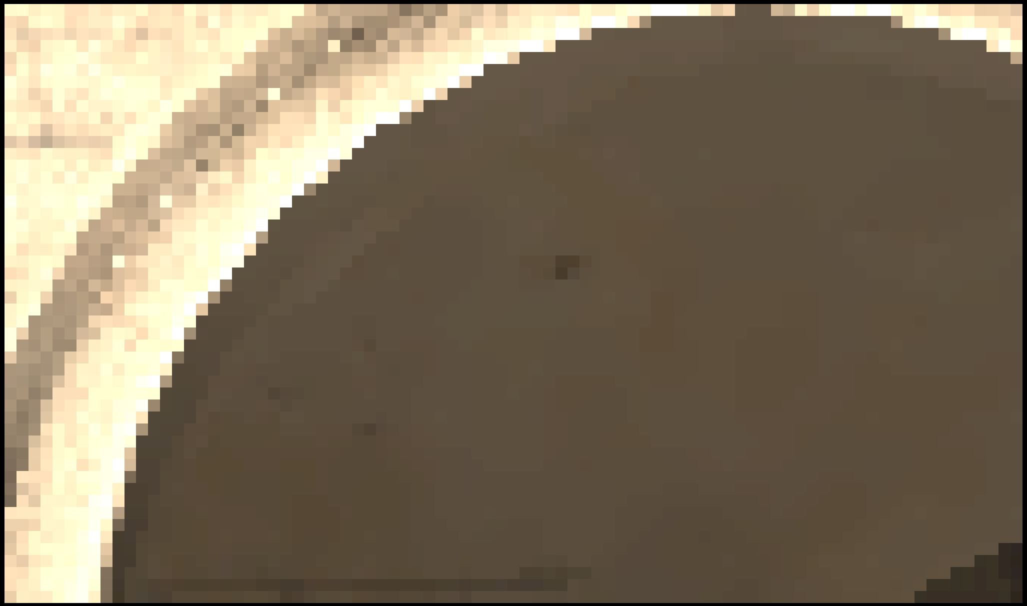
RMSE: 0.02647
PSNR: 31.57
MSSIM: 0.8881

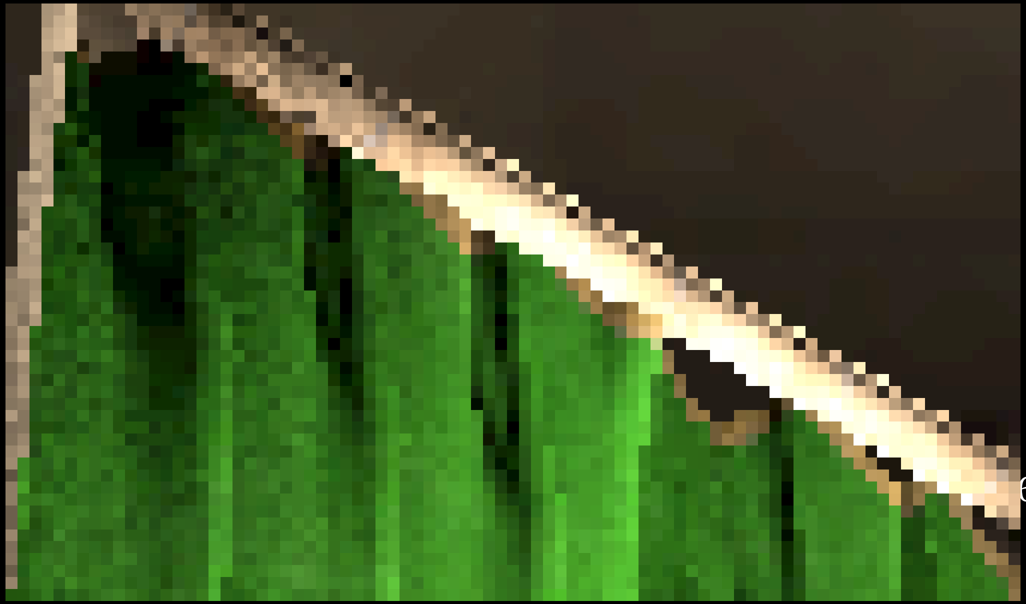
Speedup: 4.22× to GT
Speedup: 2.24× to checkerboard

Results: Deferred Renderer

DACS

- Equal time to checkerboard
- Far better quality



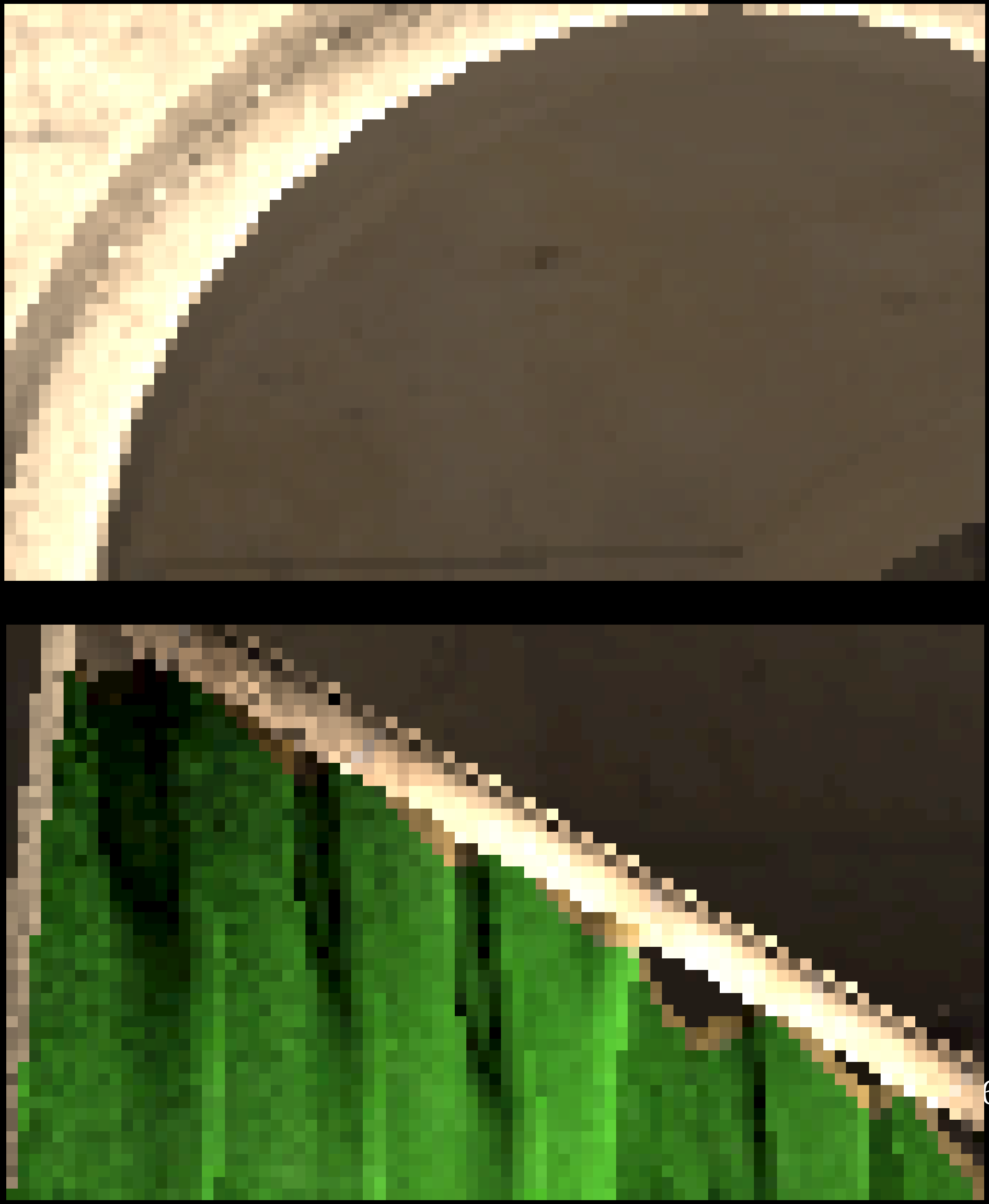


RMSE: 0.009076
PSNR: 40.85
MSSIM: 0.9620

Speedup: 1.89× to GT
Speedup: 1.00× to checkerboard

Results: Deferred Renderer

Ground Truth



Extensions and Applications

Adaptive Supersampling

Foveated Rendering

- Temporal Filtering
- Framerate stabilization
- More G-Buffer Features
- Perceptual Heuristics
- Energy Tradeoff for Mobile

Conclusion

- Significantly reduces shading complexity
- Adaptive, yet runs efficiently on GPUs
- Simple implementation

Questions

(and video)