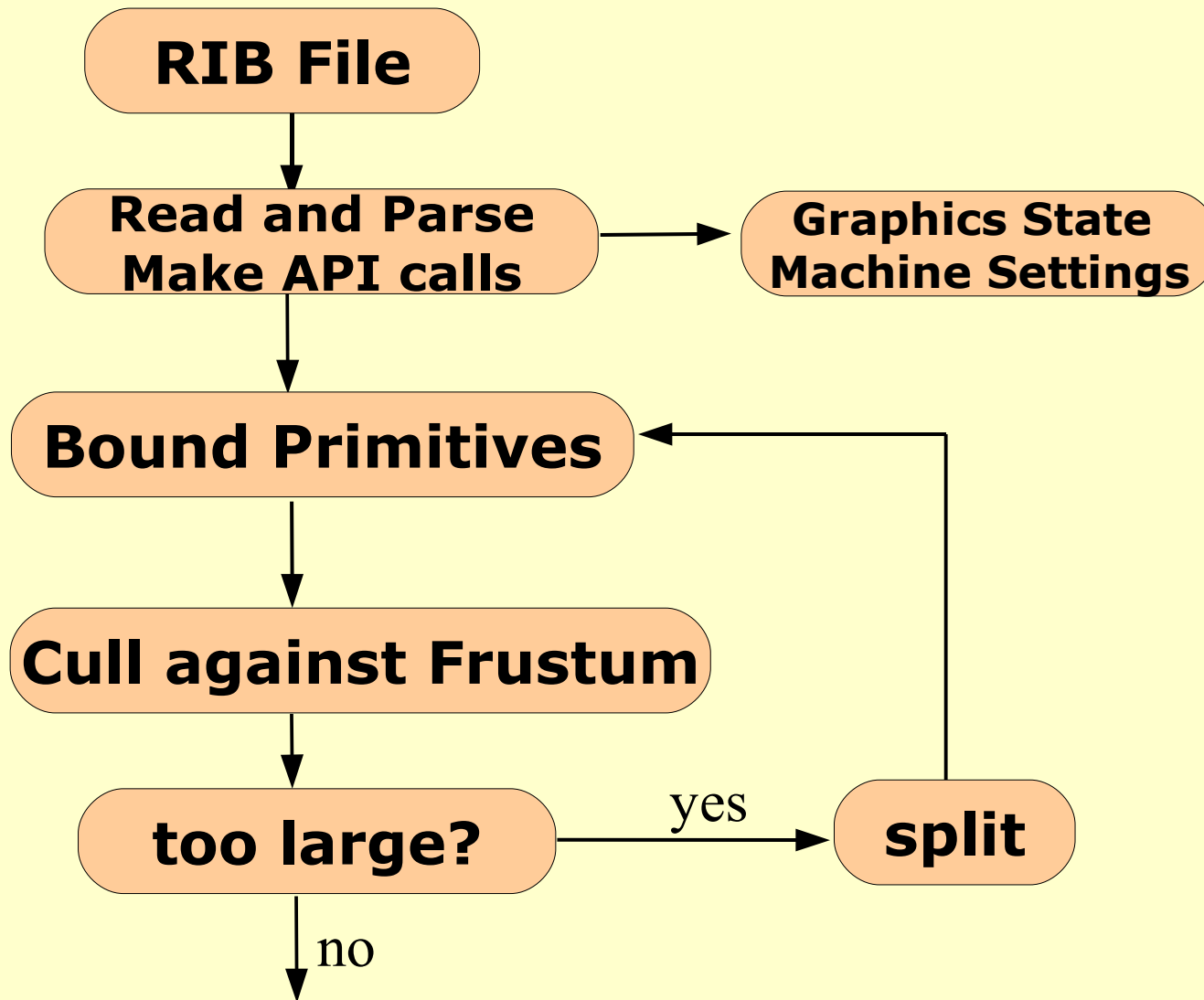


# The Reyes Rendering Pipeline



# *RenderMan Geometric Pipeline*

- **Read and Parse RIB file**
  - Translation from RIB to Ri call is straightforward.
  - Ri calls fall into two categories:
    - Those that affect the Graphics State Machine Settings
    - Those that are geometric primitives whose attributes are defined by the then-current version (settings) of the Graphics State Machine.
  - When a primitive arrives the top-of-stack set of attributes is attached to the primitive before it proceeds into the main **Reyes geometric processing engine.**

# *RenderMan Geometric Pipeline*

- **Bounding Primitives**

- Compute a camera-space axis aligned bounding box for each primitive.
- RenderMan contains no unbounded primitives – e.g. Infinite planes
- Any single faced – back facing primitives are also culled at this point.

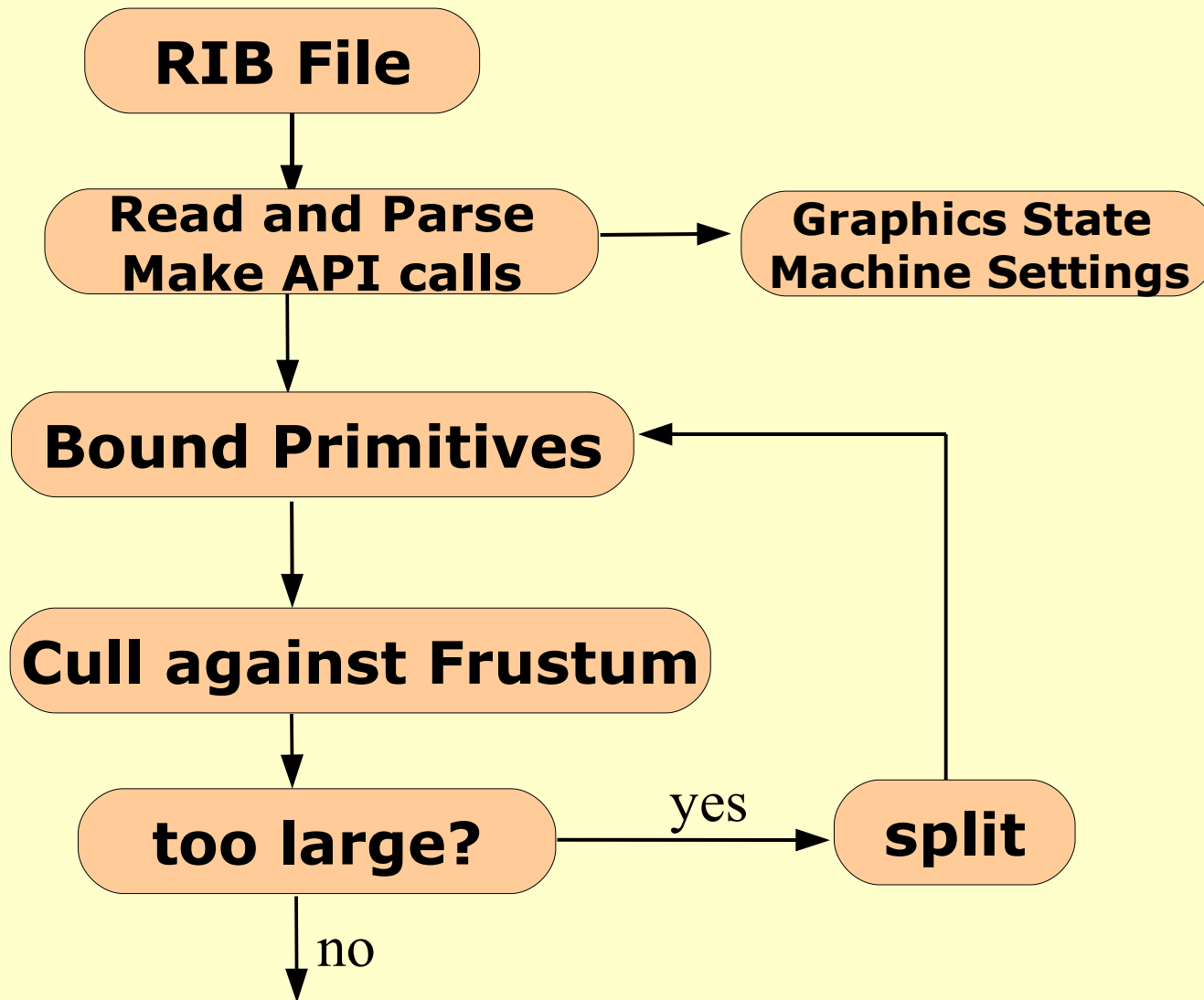
# *RenderMan Geometric Pipeline*

- **Cull against Frustum**
  - Bounding box is checked against the frustum boundaries (determined by the camera settings in the Graphics State Machine).
    - If bounding box is entirely outside the frustum the primitive is culled.
    - If the bounding box is at least partly inside the frustum continue to splitting.

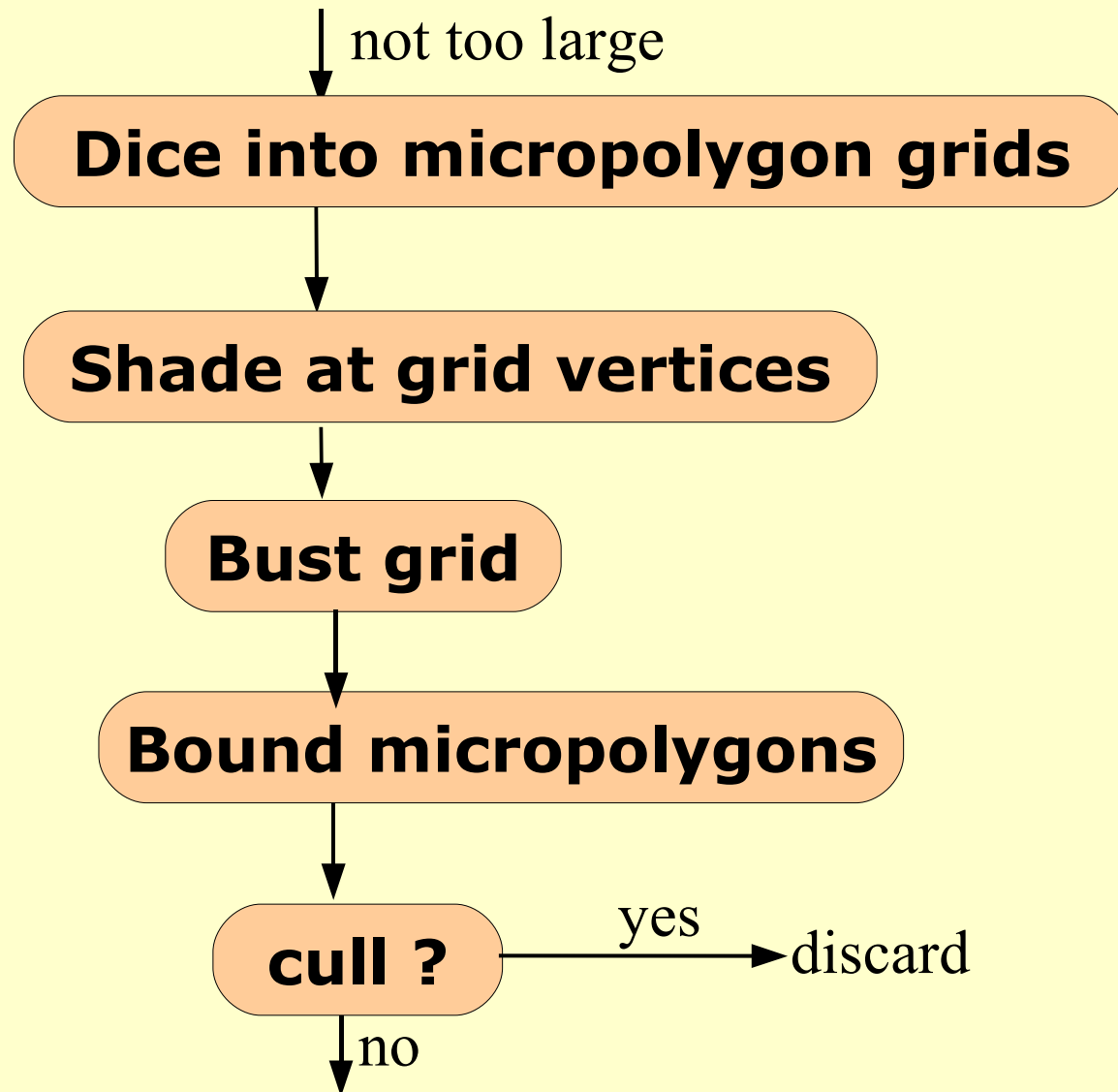
# *RenderMan Geometric Pipeline*

- **Size testing for primitives - too large?**
  - All primitives must be “small enough” - metric?
  - A primitive that is too large must be split – splitting algorithm varies from one primitive type to another.
  - After splitting newly created sub-primitives are sent back through the bounding, culling, size testing loop once again.

# The Reyes Rendering Pipeline



# *The Reyes Rendering Pipeline*



# *RenderMan Geometric Pipeline*

- **Dicing into micropolygon grids**
  - Once past the size test (and the diceability test), primitives are tessellated into grids.
  - Grids are rectangular arrays of quadrilateral facets – actually represented as tiny bilinear patches – adjacent vertices do not need to be stored more than once.
  - Regardless of the original primitive type all grids look the same to the rest of the pipeline.
  - Micropolygon size can be controlled by RI GSM settings and (except for rough test renders) are typically no bigger than one quarter of a single pixel in screen space – this allows finer detail and better anti-aliasing in final render.

# *RenderMan Geometric Pipeline*

- **Dicing into micropolygon grids**
  - Since micropolygons have come from primitives that have already been bounded and culled, no clipping is required (some micropolygons will be culled later).
  - The grid contains all the original primitive's attributes.

# *RenderMan Geometric Pipeline*

- **Shade at grid vertices**
  - Application of the various shaders:
    - **Displacement Shader** – may move grid vertices or change normals
    - **Surface Shader** – requires evaluation of all Light Shaders attached to the original primitive.
    - **Light Shaders** run as coroutines of Surface shaders and results are cached for later possible use.

# *RenderMan Geometric Pipeline*

- **Shade at grid vertices**
  - When the Surface Shader has finished the color and opacity of every grid vertex has been computed.
  - Atmosphere Shader – makes adjustments to color and/or opacity to simulate fog or other volumetric effects.
  - Texturing seems to happen at this point – details are lacking.

# RenderMan Geometric Pipeline

- **Bust Grid – Bound micropolygons - cull**
  - The grid is busted into individual micropolygons which are bounded in ways similar to bounding primitives. Bounding boxes are tested against the frustum and as before all those lying outside the frustum are culled. Backface culling is also done here.

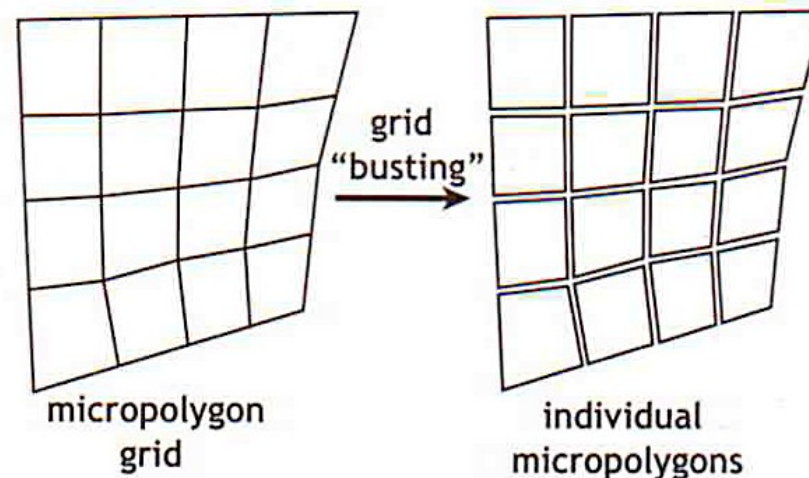
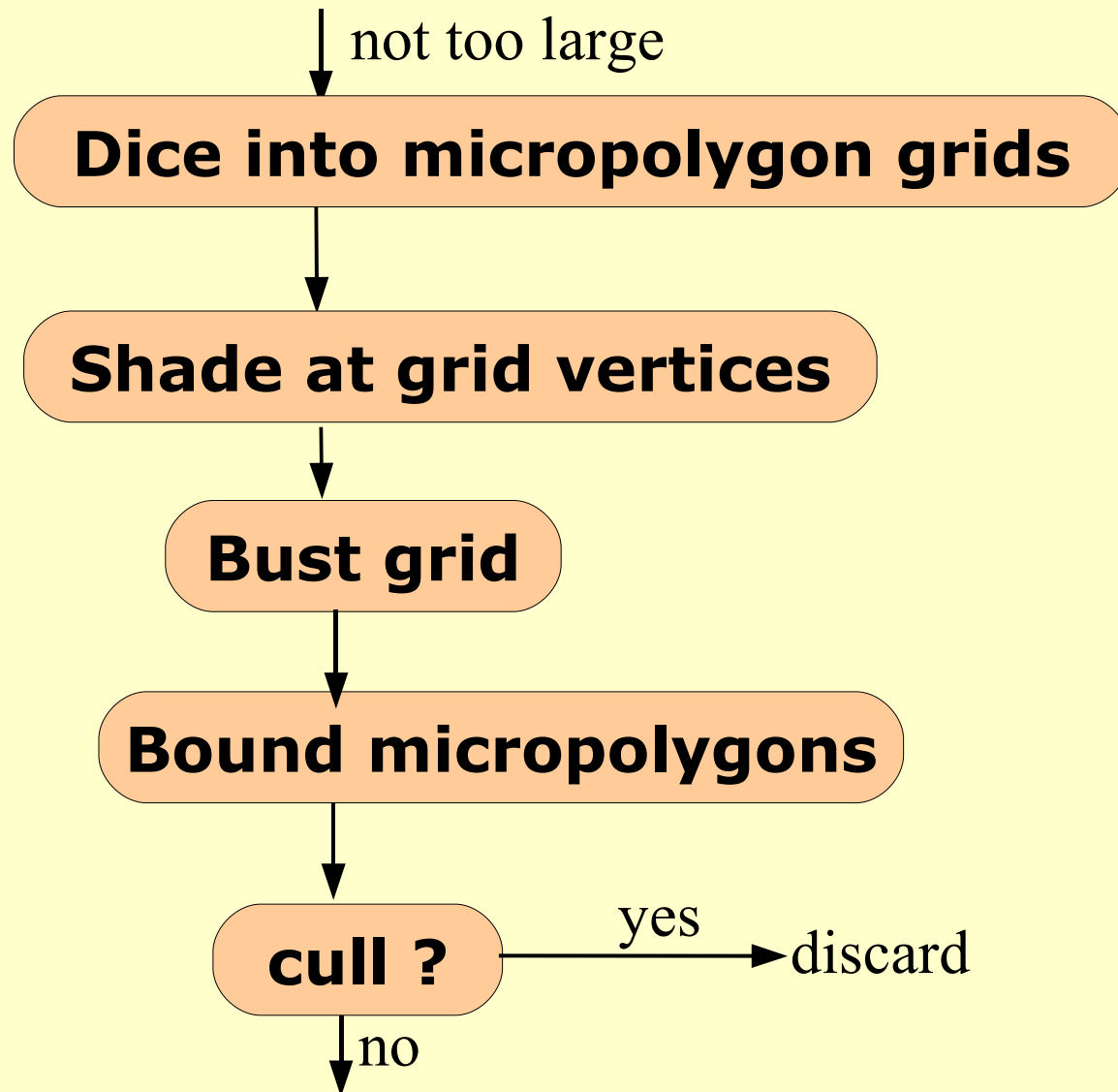
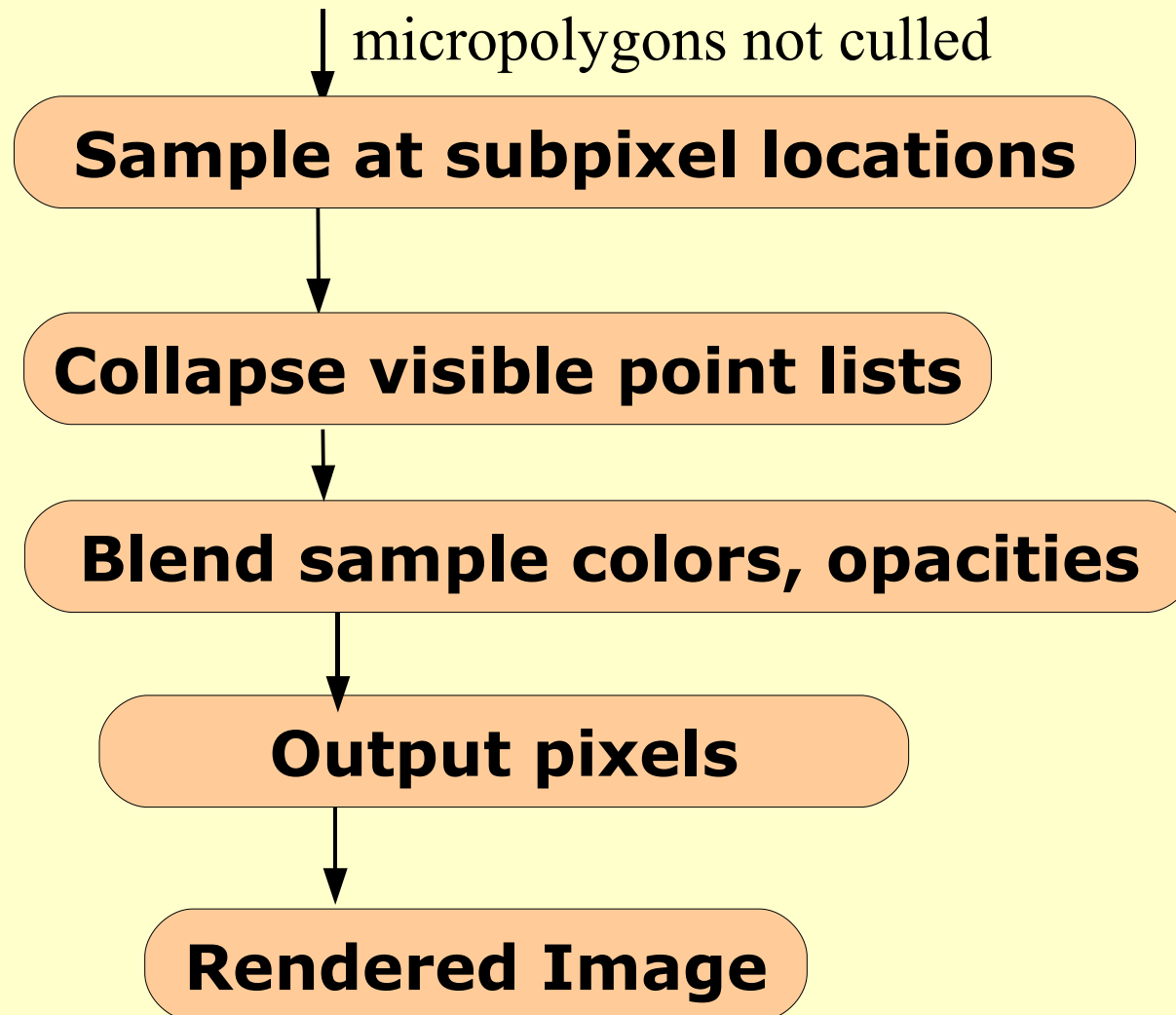


Figure 2.9 Grid "busting"  
from Rendering for Beginners, S. Raghavachary

# *The Reyes Rendering Pipeline*



# *The Reyes Rendering Pipeline*



# *RenderMan Geometric Pipeline*

- **Convert bounds and micropolygon into screen space**
- Both the bounding box and the micropolygon vertices are converted into “screen space” (possibly into Normalized device coords)

# RenderMan Geometric Pipeline

- **Sample at subpixel locations**
  - The screen space bound determines which pixels might be covered by the micropolygon. For each of these pixels a number of stochastically jittered pixel sample locations are predetermined.
  - Number and exact location of samples are specified in GSM settings.

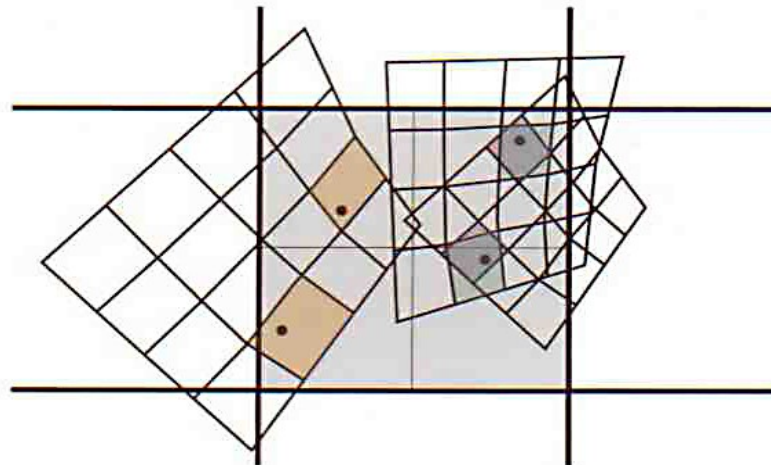


Figure 2.10 Sampling micropolygons at subpixel locations

from Rendering for Beginners, S. Raghavachary

# *RenderMan Geometric Pipeline*

- **Sample at subpixel locations**
  - The color of the micropolygon at the sampled point is determined by one of several interpolation methods considering the colors at all four corners. (e.g. Gouraud interpolation)
  - The color, opacity and depth of the micropolygon at the sample point are recorded in the sample location's visible point list.
  - The visible point list for any sampled point is kept in depth sorted order. The front-most opaque point in the list is the last one that will need to be considered. This algorithm is often referred to as "Hiding".

# *RenderMan Geometric Pipeline*

- **Collapse visible point lists**
  - Involves determining a single color for each sample by considering depth order and opacities of each sample

# *RenderMan Geometric Pipeline*

- **Blend sample colors**
  - After collapsing visible point lists all the samples within a single pixel are blended together using a specified reconstruction filter (essentially an image processing operation). This provides a single color value for the pixel that can be displayed on the screen or added to an image file.

# RenderMan Geometric Pipeline

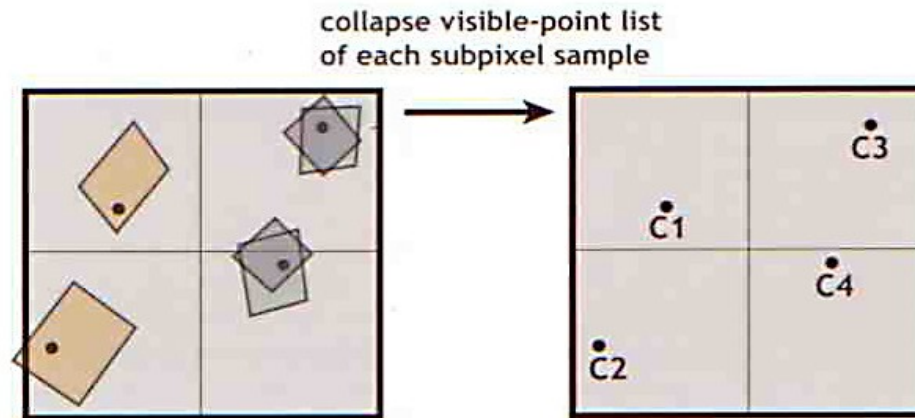


Figure 2.11 Collapsing visible-point sample lists

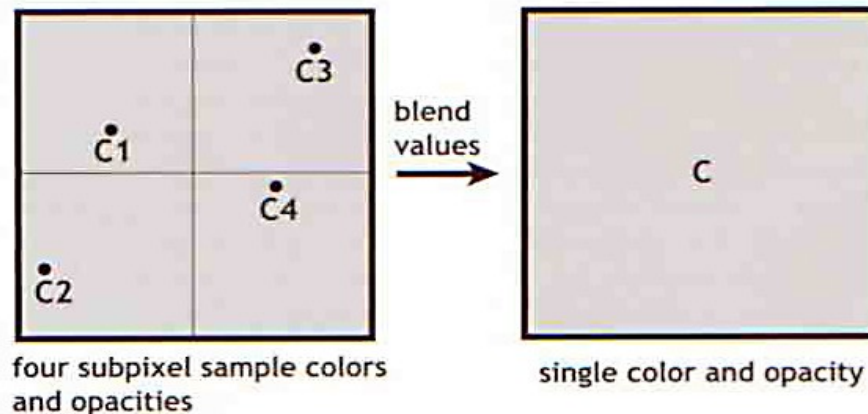
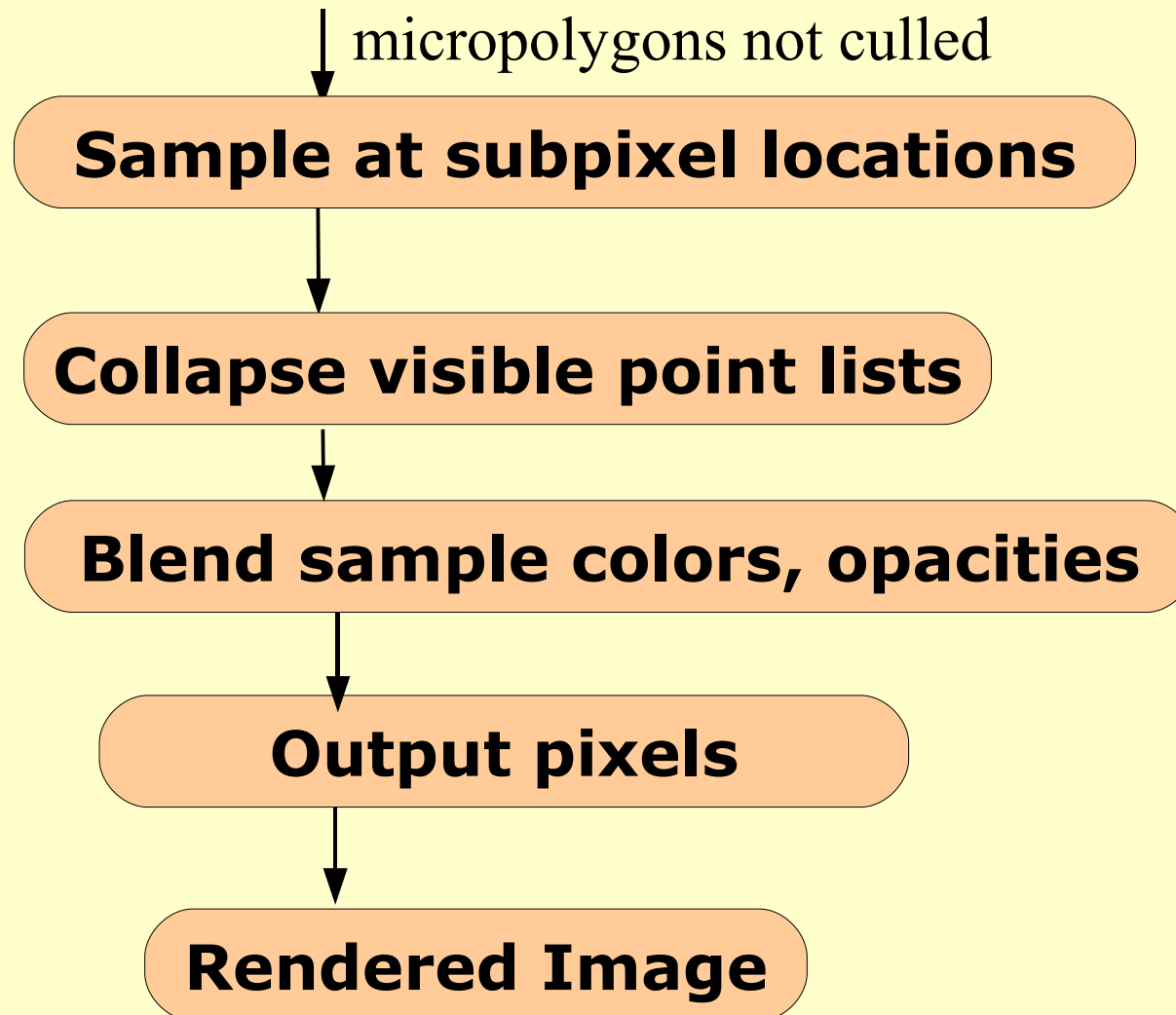


Figure 2.12 Blending (filtering) subpixel values to produce final pixel result  
from Rendering for Beginners, S. Raghavachary

# *The Reyes Rendering Pipeline*



# *RenderMan Geometric Pipeline*

- **Bucketing**

- To reduce the number of visible point lists that must be maintained, the image is rendered in pieces or “blocks”.
- Divides the image into rectangular shaped pieces called buckets. All sample locations and pixels can then be completely finished and freed before moving onto another part of the image. Keeps memory for visible point lists to a minimum.
- Primitives must be bucket sorted before splitting – using bounds all primitives are placed onto a particular bucket list.

# *RenderMan Geometric Pipeline*

- **Bucketing**

- Buckets are processed one at a time – during splitting some subprimitives will be removed from one bucket and placed into another bucket's list.
- The entire scene (group of visible primitives) will have to be in memory for each bucket but visible point lists are maintained only for that bucket and are gone when the next bucket is processed.