

## Rendering Pipelines



# A Rendering Pipeline

- Rendering or Graphics Pipeline is the sequence of steps that we use to create the final image
- Many graphics/rendering pipelines have been proposed although the most historically/practically dominant ones are:
  - The Rasterization Pipeline
  - The Micropolygon (Reyes) Architecture
  - Ray Tracing (Path Tracing and other image driven methods can be also classified here)

# General Principles

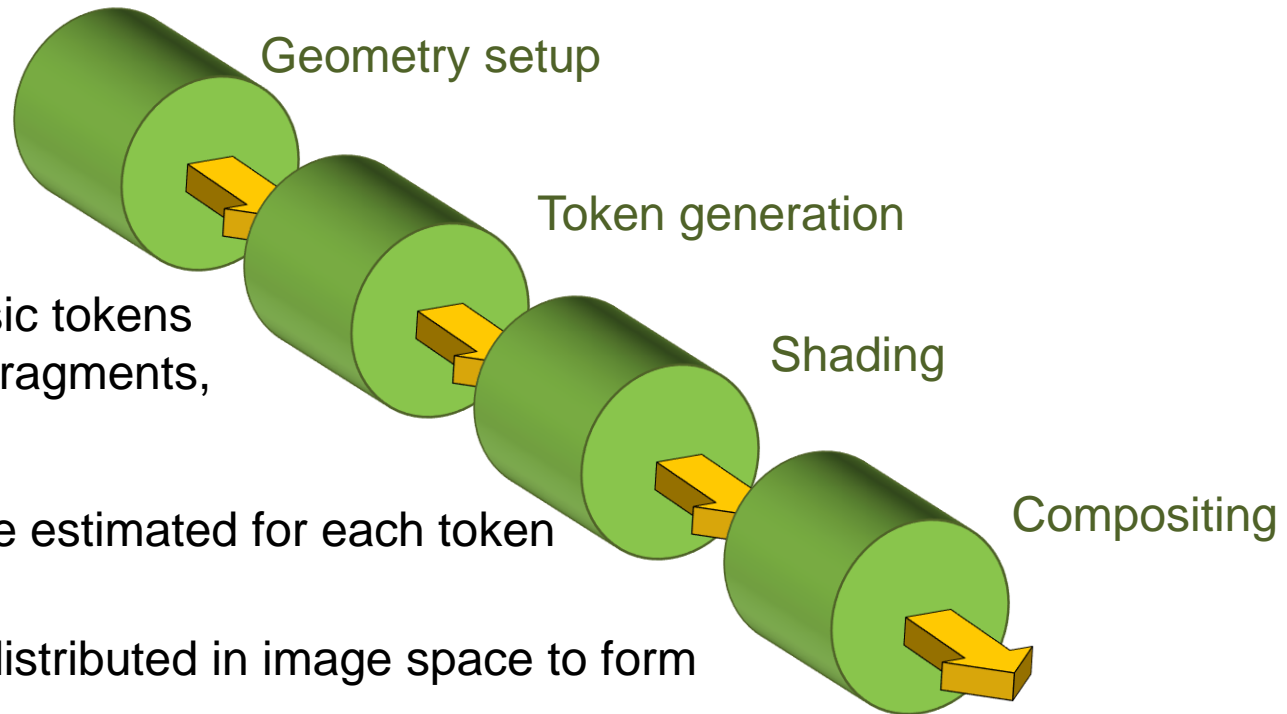
- All systems share some broad common stages and features

The geometric data are “arranged” in space, replicated and cleaned up

The system issues the basic tokens for the image generation (fragments, paths, micropolygons)

Shading and other data are estimated for each token

Results are composited / distributed in image space to form the rendered image



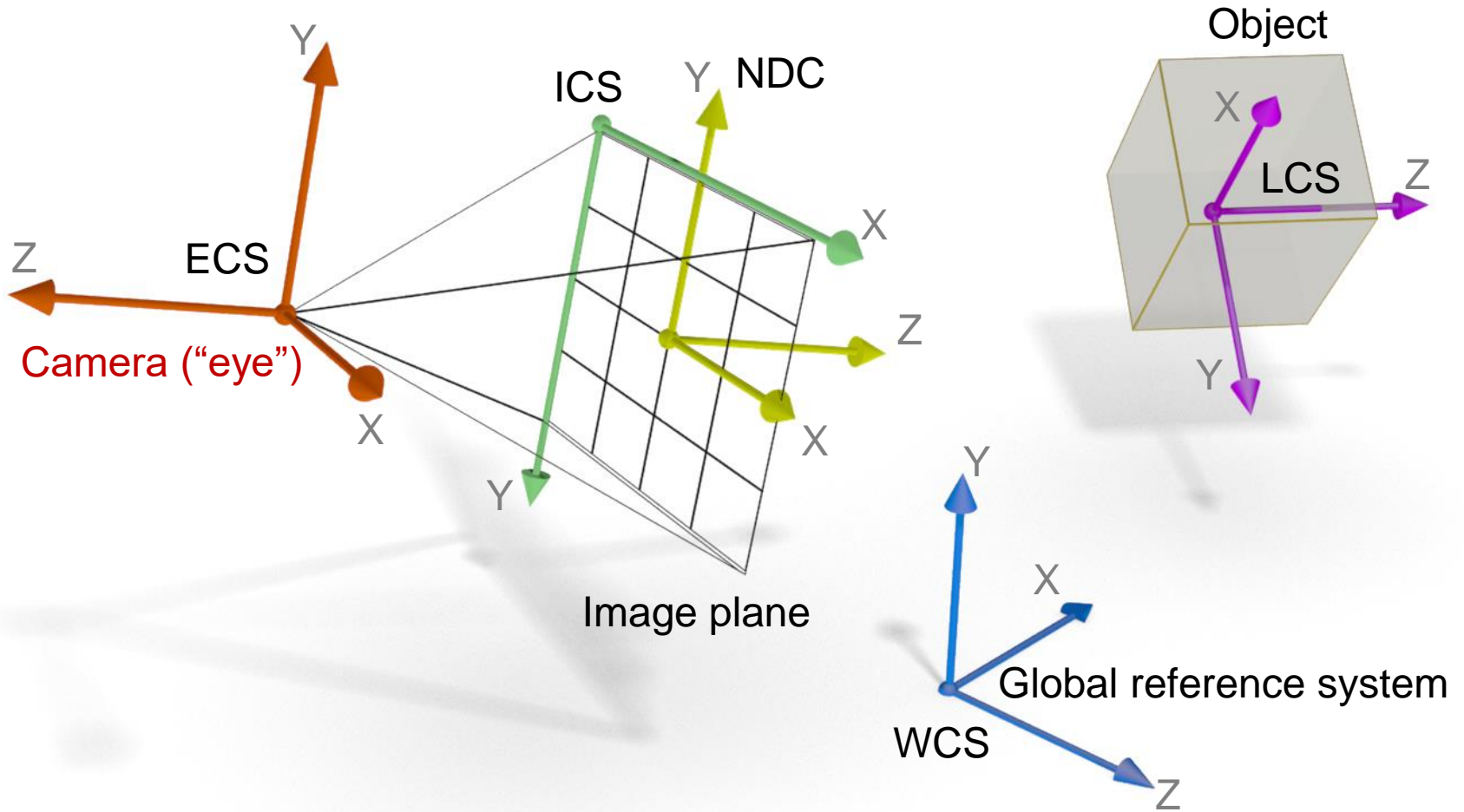
Possibly re-entrant

- It is easy to see, even with no knowledge of a particular rendering pipeline or system that rendering architectures are:
  - Inherently parallelizable
  - Easy to model and implement as highly-efficient pipelines
- These features are extensively exploited both in software renderers and hardware implementations

# Reference Frames (1)

- All rendering architectures treat geometric primitives and other mathematical functions and constructs relative to a reference coordinate system, which usually changes among the various stages
- We typically encounter the following reference frames:
  - Local- “Object”-Space Coordinate System (LCS)
  - World-Space Coordinate System (WCS)
  - Eye-Space Coordinate System (ECS)
  - Normalized Device Coordinates (NDC)
  - Image Space Coordinates (IS)

# Reference Frames (2)



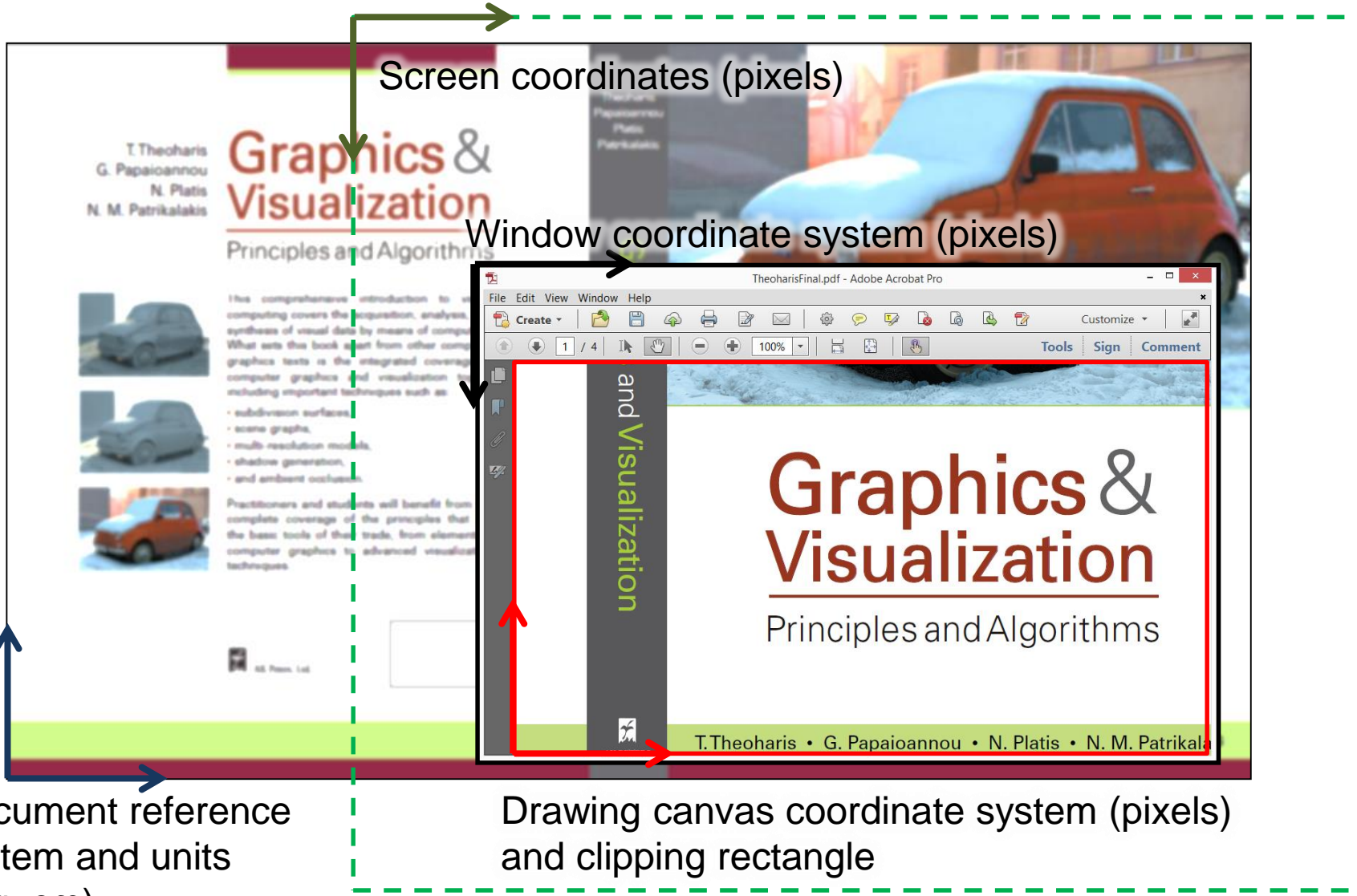
# Coordinate Systems – Windows

Screen coordinates (pixels)

Window coordinate system (pixels)

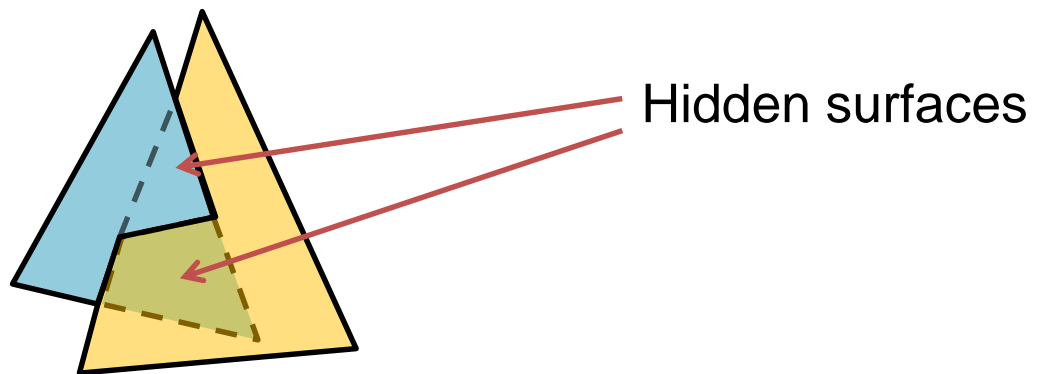
Document reference system and units (e.g. cm)

Drawing canvas coordinate system (pixels) and clipping rectangle



# Hidden Surface Elimination (Sorting)

- A common task to all pipelines is the proper ordering of the parts of surfaces to correctly display the visible ones in front of the hidden parts
- The implementation mechanism for HSE varies significantly from one architecture to another





# The Reyes Rendering Architecture (1)



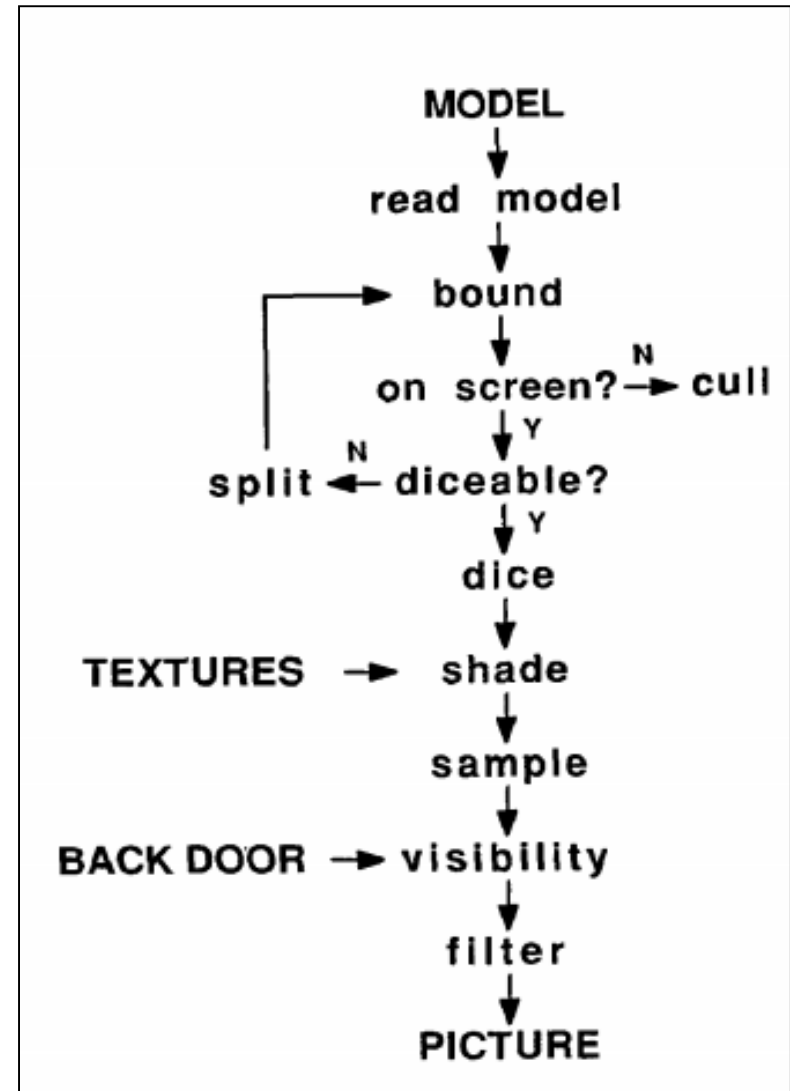
- Production-grade architecture used by Pixar (Photorealistic Renderman) in many films
- The inspiration behind modern GPU (rasterization-based) shading pipelines

# The Reyes Rendering Architecture (2)

- Efficiency through:
  - Controllable breakdown of rendering to many lightweight tasks
  - Localized calculations
- Principle:
  - Break up geometry into tiny micro-polygons
  - Shade each micro-polygon
  - Test for visibility
  - Sample micro-polygons to contribute to pixels

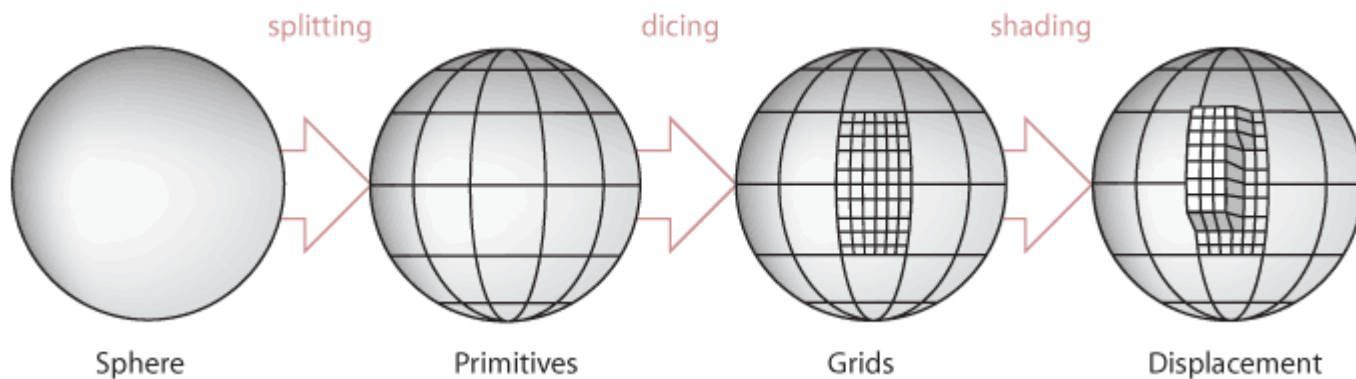
# The Reyes Pipeline

- Geometry is either:
  - iteratively split into simpler models or
  - “diced” into a grid of micro-polygons
- Each grid is shaded independently
- Micro-polygons are tested for visibility and
- Sampled and filtered in the image domain



# Reyes Micro-polygons

- Micro-polygon size is roughly  $\frac{1}{2}$  the pixel width and height (see Nyquist criterion)
- Splitting limit is determined by the above
- Shading results in determining the position (after possible displacement) and color of the micro-polygon

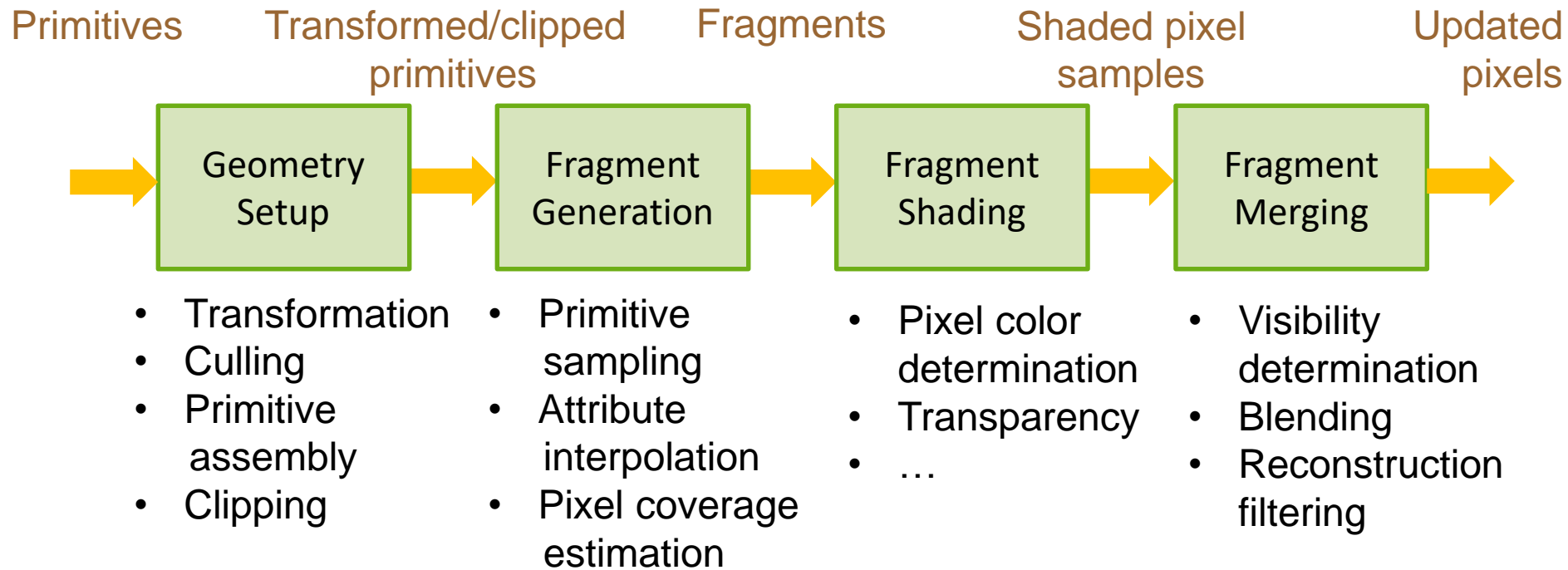


# Reyes - Advantages

- Single geometric type: everything resolves to micro-polygons
- Parallelism
- Linear scalability w.r.t scene complexity and image size
- Controllable image quality / rendering time
- Bounded memory footprint
- Locality of operations / memory access
- Efficient displacement mapping (see texturing)

- The heart of most software-based primitive drawing algorithms
- The architecture of all real-time hardware graphics pipelines (Graphics Processing Units - **GPUs**)
- They implement strategies for sampling screen-space primitives on a regular grid (raster) at a pixel or sub-pixel level
- Shading occurs after the primitive samples have been determined (often called **fragments**)

# A High Level Rasterization Pipeline



- This is a general 2D/3D overview of the task a rasterization pipeline involves
- The GPU graphics pipeline is discussed separately

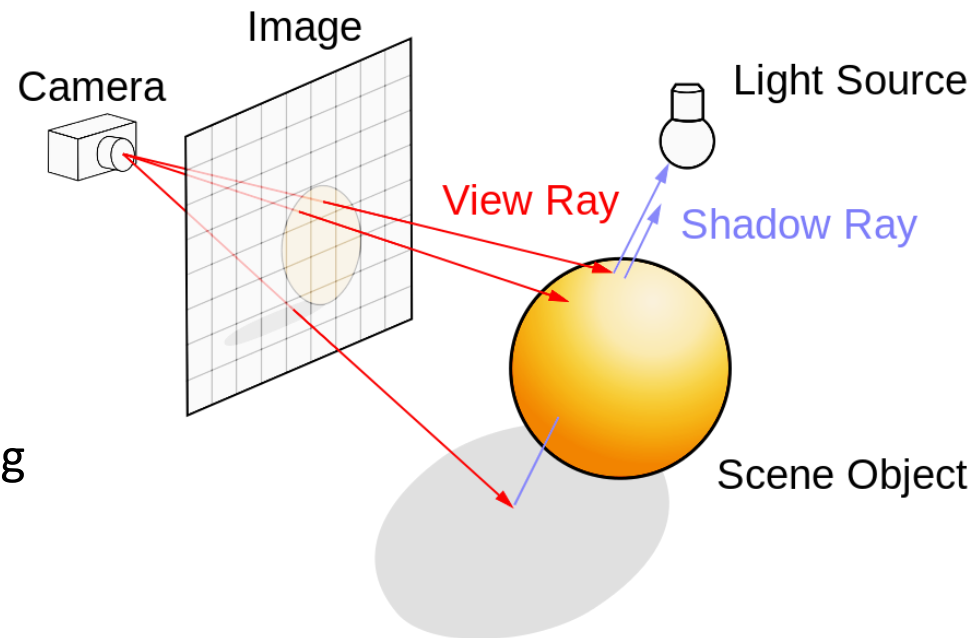
# Ray Tracing Pipelines

- In RT, instead of the primitives, the path space is sampled:
  - Rays are generated and “traced” through the 3D environment
  - Intersection of rays with the nearest geometric primitives (implicit HSE) triggers shading and spawning of new rays
- HSE:
  - The Reyes and rasterization pipelines perform HSE in image space
  - Ray tracing methods do the sorting 3D space (ray space)



# Simple Ray Casting

- In its most simple form RT:
  - Generates a number of rays from the “eye” through the pixel locations on the image plane
  - Computes the nearest intersection of rays with the primitives
  - Shades the closest points
- This simple process is called ray casting
  - Shading also typically involves sending rays towards the light source(s) to check for shadowing



- There are many interesting photorealistic image generation algorithms that are based on this simple ray tracing idea (see Path Tracing etc.)
- The power and elegance of ray tracing comes from:
  - The fact that rays can be recursively traced through the environment (Whitted-style ray tracing)
  - The ability of rays to interact with many mathematical constructs, beyond simple primitives or even surfaces! (see volume rendering)

# Recursive Ray Tracing Example



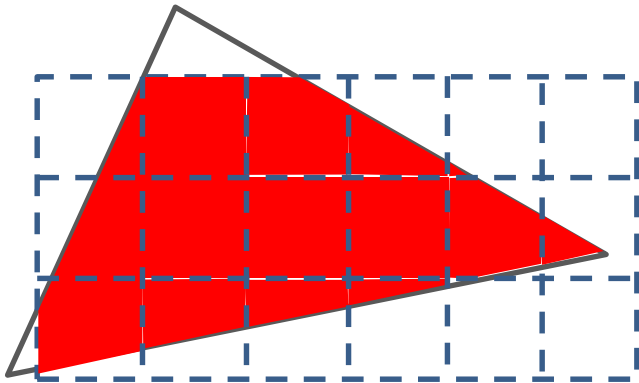
Ray casting



Recursive ray tracing  
(stochastic path tracing)

# Pixel Coverage

- The impact a primitive has to a single pixel it occupies is generally proportional to the pixel coverage
- In the worst case, the coverage is binary, determined by the generation or not of a fragment for this pixel



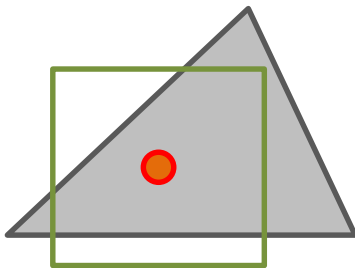
# Why Pixel Coverage is Useful?

- Can help blend fragments more accurately
- Can be used in antialiasing filters to “smooth” out and properly render the contribution of
  - thin structures
  - sharp transitions

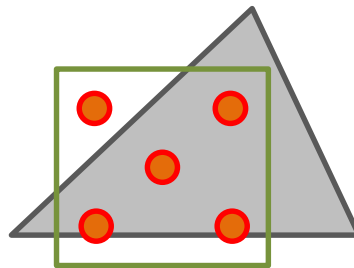
antialiasing

# Determining Pixel Coverage

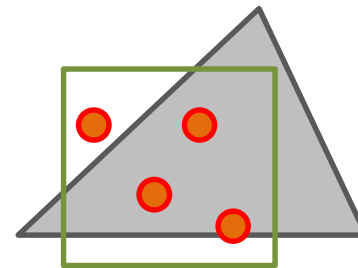
- Pixel area covered by a polygon can be computed analytically if we assume a rectangular pixel (which is not), or better:
- If we sample the coverage with an arbitrary pattern of coverage taps:



Pixel center



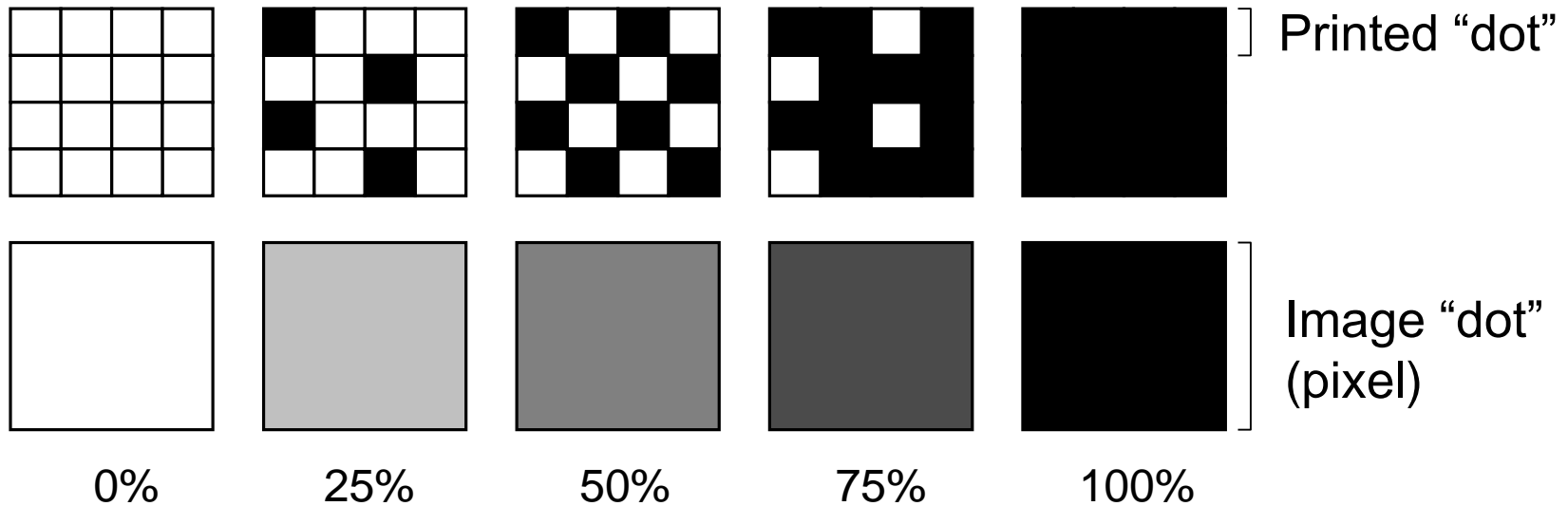
Fixed pattern



Random / rotating pattern

# Half-toning

- The pixel coverage idea was also used in the printing industry (and old display systems) to effectively approximate a wide range of intensity variations with only a few (usually 2) available tones:



# Half-toning in Print





- Conventional printing also involves a raster generation phase, so we treat it accordingly
- Vectorized output: Plotters
  - Use shape outlines to control the trajectory of a plotting head
- 3D printing
  - Build surfaces in space layer by layer
  - Extract contours (outlines) to express layer slice boundaries
  - Use contours to drive a material-depositing head

- Georgios Papaioannou
- Sources:
  - [REYES]: R. L. Cook, L. Carpenter, E. Catmull, The Reyes image rendering architecture. *SIGGRAPH Comput. Graph.* 21, 4 (August 1987), pp. 95-102, 1987.
  - [RTR]: T. Akenine-Möller, E. Haines, N. Hoffman, Read-time Rendering (3<sup>rd</sup> Ed.), AK Peters, 2008
  - T. Theoharis, G. Papaioannou, N. Platis, N. M. Patrikalakis, Graphics & Visualization: Principles and Algorithms, CRC Press