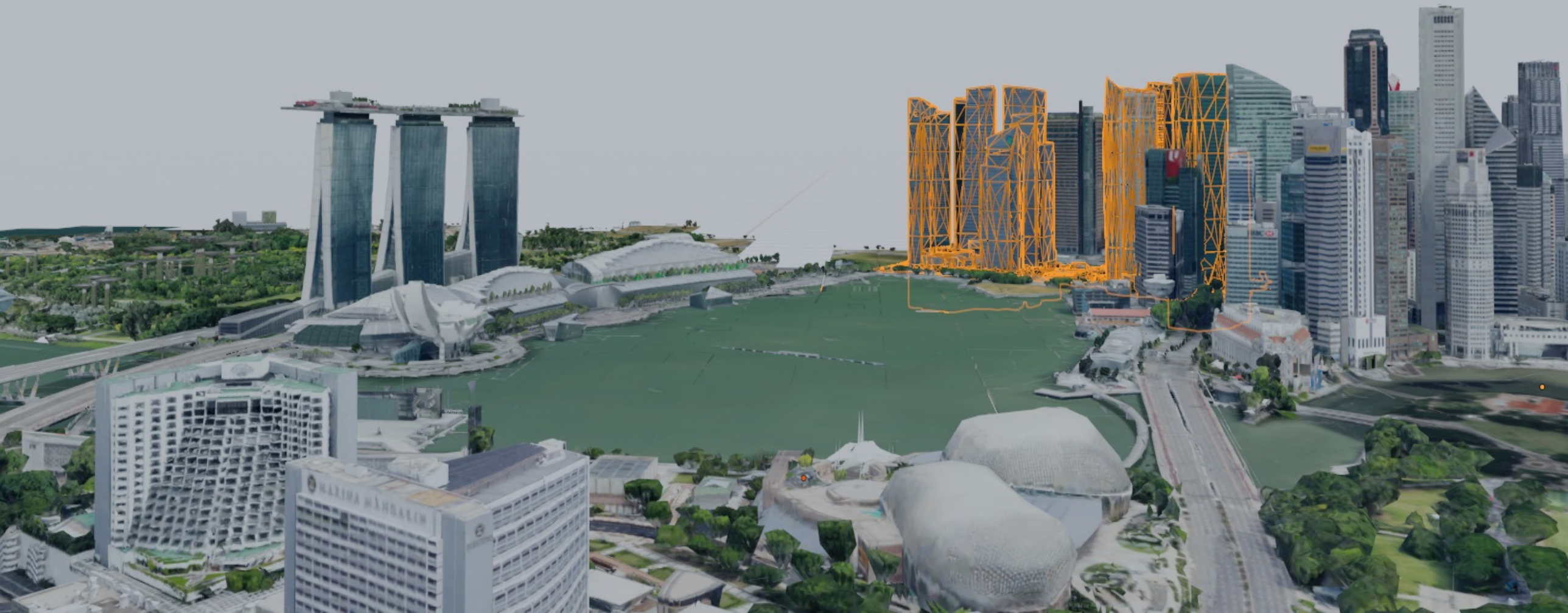
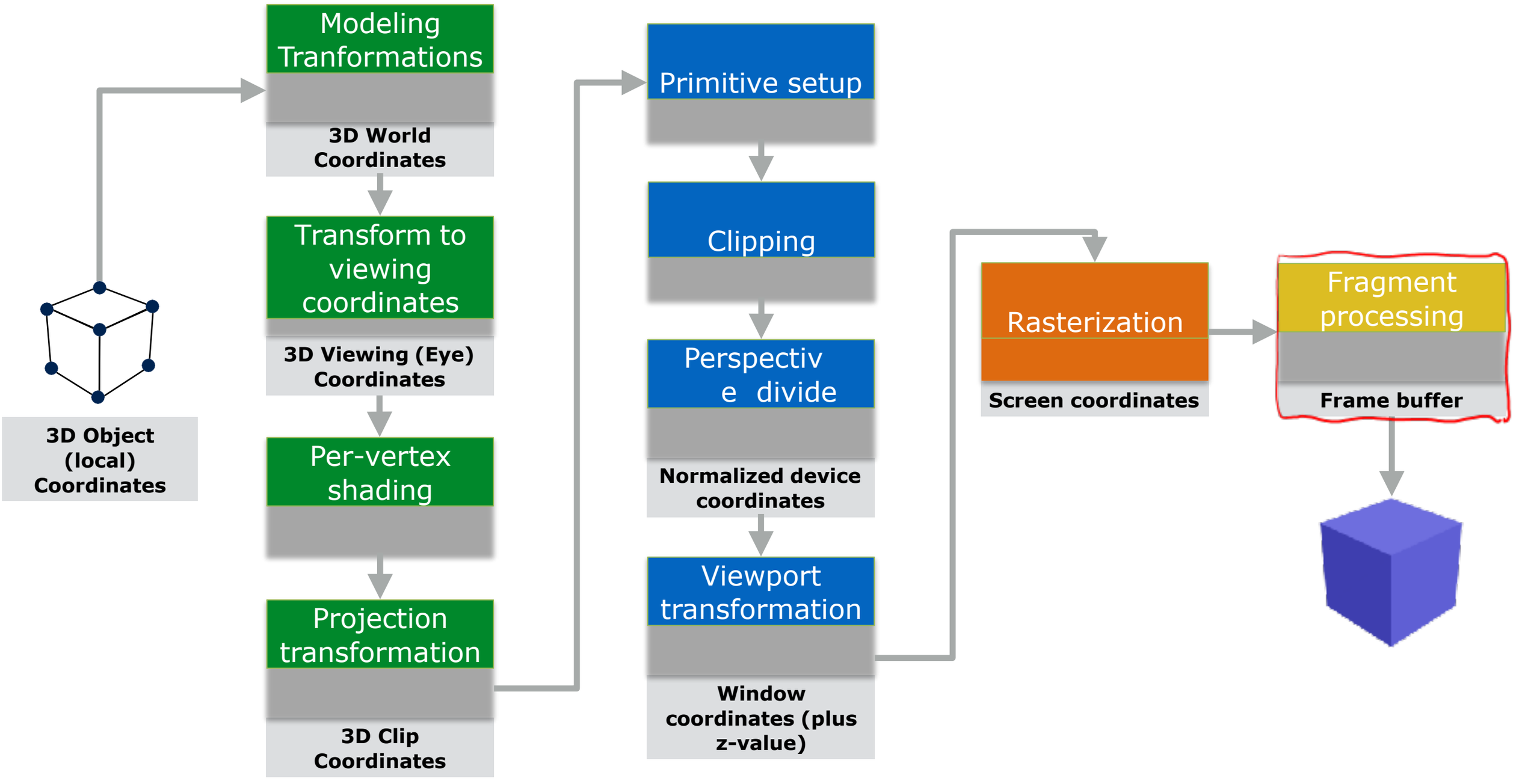
Fragment processing

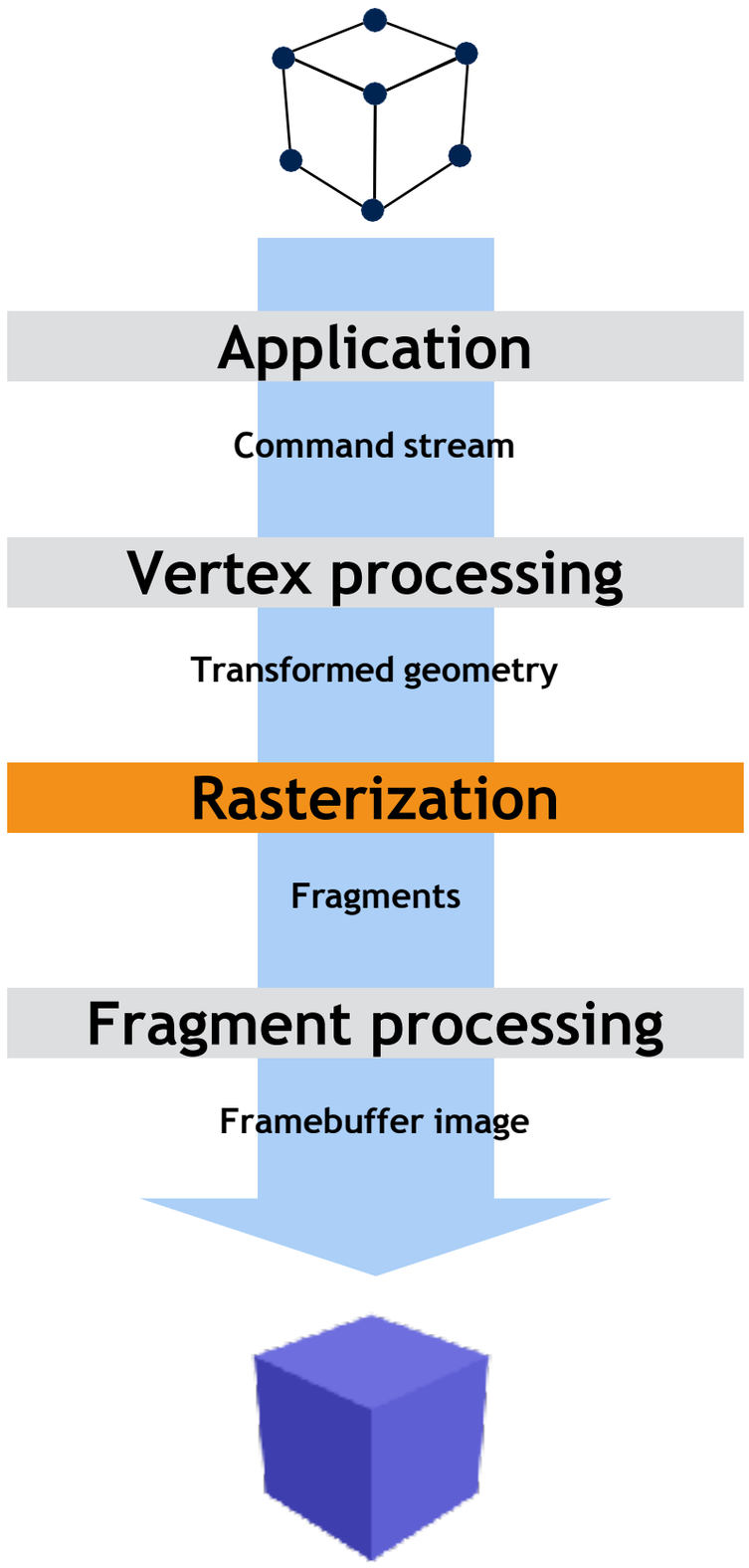


Coordinate systems



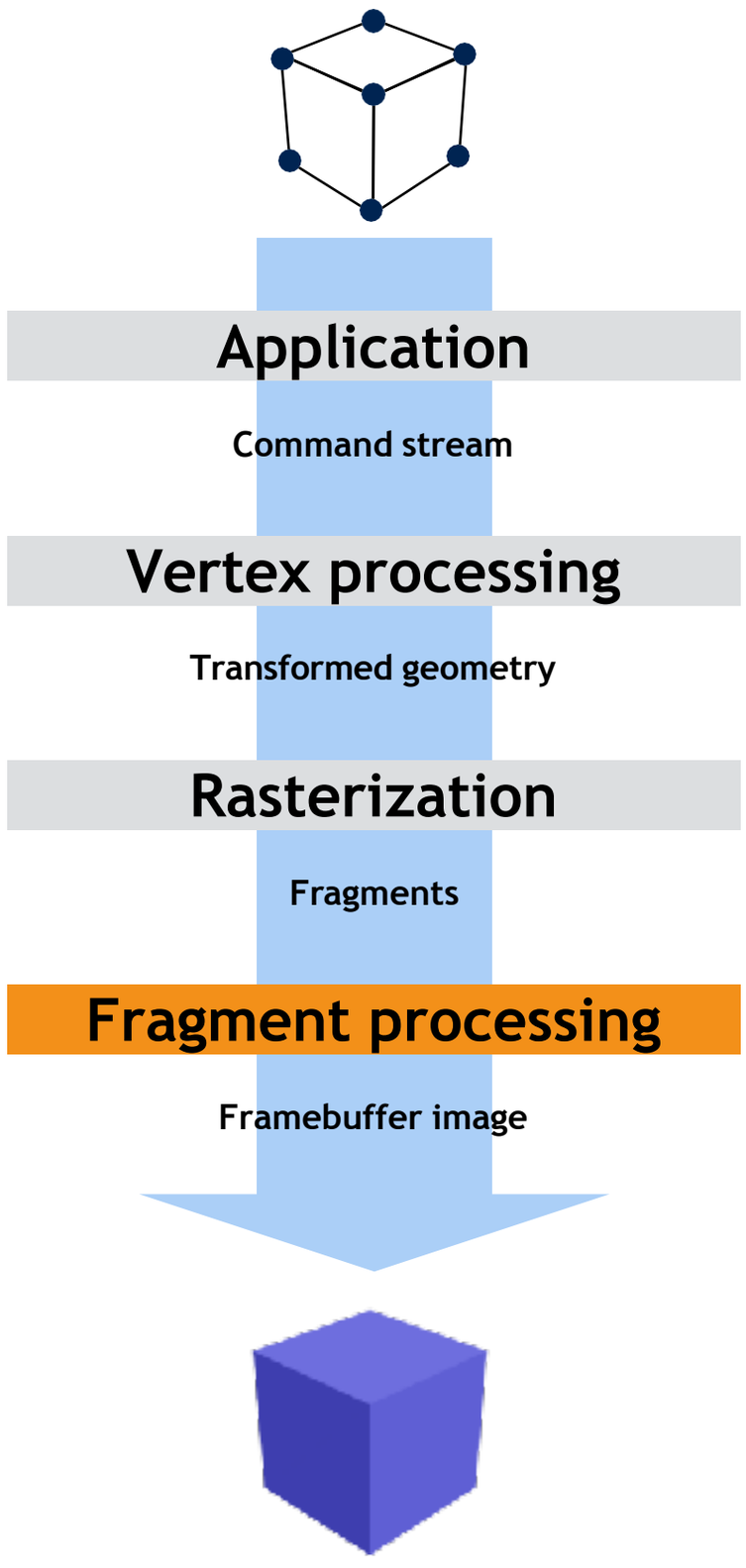
Real-time rendering pipeline

- **Rasterization**
  - **enumerates** the pixels that are covered by the primitive
  - **interpolates** values, called attributes, across the primitive
- The output of the rasterizer stage is a set of **fragments**, one for each pixel covered by the primitive
- Fragment data
  - raster position
  - depth (z-value)
  - interpolated attributes (color, texture coordinates, etc.)
  - stencil
  - alpha

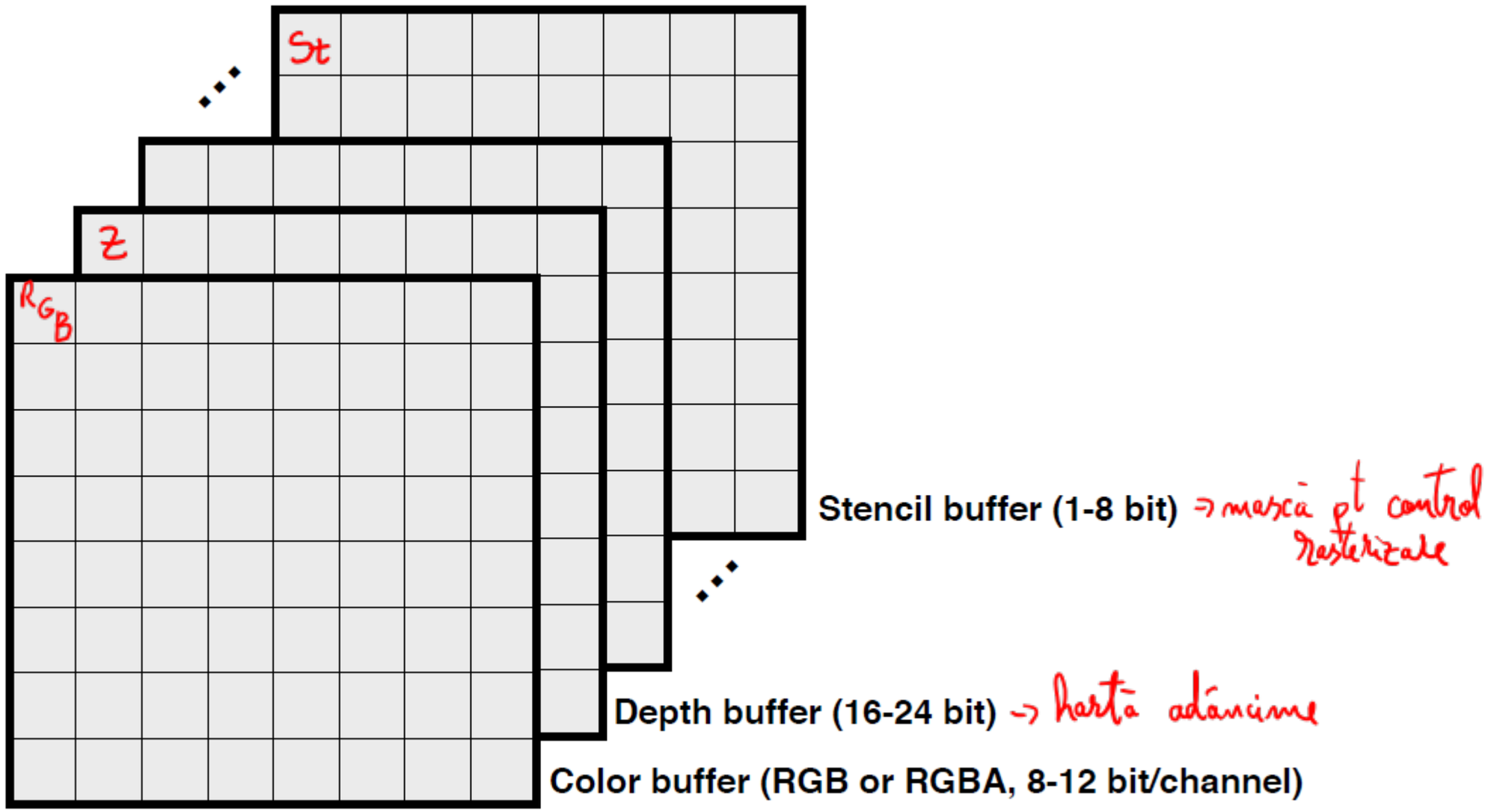


Fragment processing

- Process each fragment from the rasterization process into a set of colors and a single depth value
- **Scissor test** - discard fragments outside of a certain rectangular portion of the screen
- **Stencil test** - test the fragment's stencil value against the value in the current stencil buffer; if the test fails, the fragment is culled
- **Depth test** - test the depth of the current fragment with the existing depth; if the test passes update the depth buffer
- **Blending** - combine the fragment's color with the existing color in the frame buffer



Framebuffer



Illumination

- Compute light intensity on each object point (fragment)
- **Reflection model**
  - **Local** – direct illumination
  - **Global** – direct illumination + multiple reflections

  Influenced by
  - **Light model**
    - Ambient illumination
    - Diffusion illumination
    - Specular illumination
  - **Materials**

Local vs Global illumination

[GPS Project – Simina Alexandru]

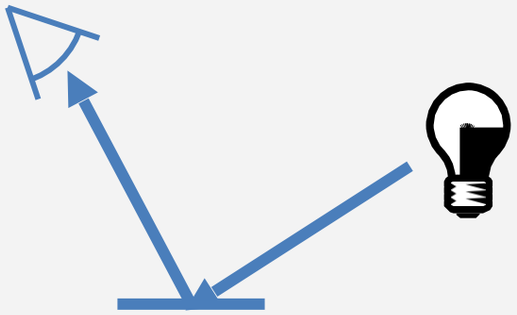
Local vs Global illumination

[GPS Project – Somfelean Iulia]

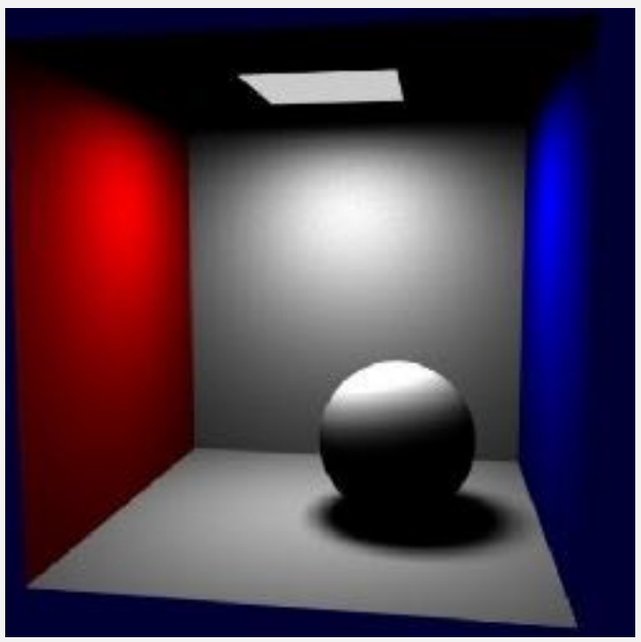
Local vs Global illumination

[http://simonmajar.com/storage/originals/e6/d3/screenshot0000.jpg]

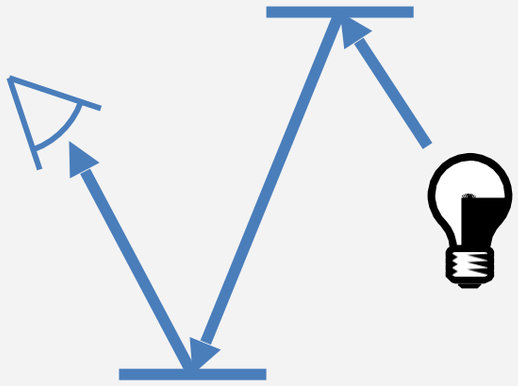
Global illumination



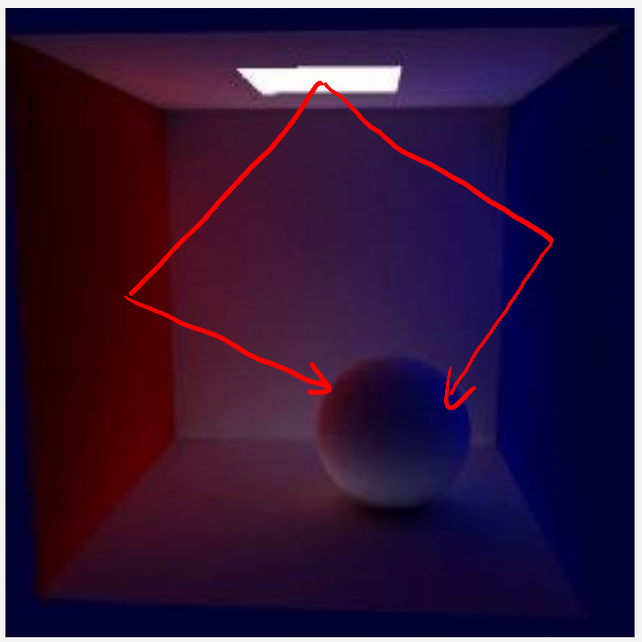
Direct illumination



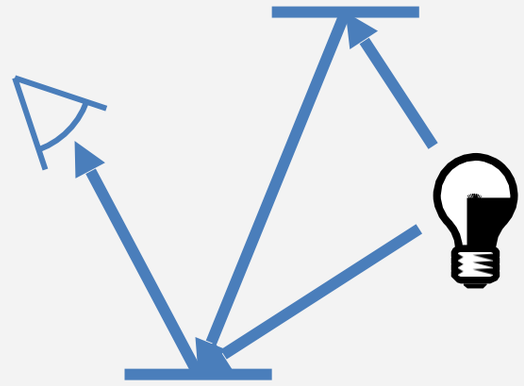
+



Indirect illumination



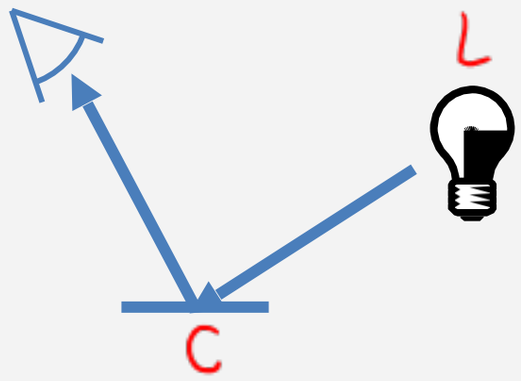
=



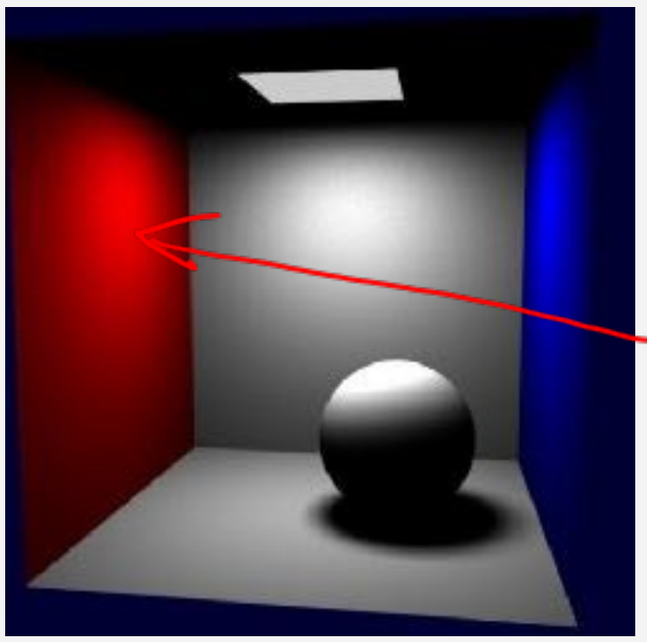
Total illumination

[Andries van Dam]

Global illumination



Direct illumination



Exemplu pt. calcul iluminare:

L (255, 255, 255) -> lumină "albă"

C (100, 0, 0) -> culoare obiect
R G B

=> obiectul "reflectă" lumina roșie în proporție de 100/255. Reflectă 0% verde, 0% albastru

Exemplu de calcul: Adaugă $RGB_{\text{lumină reflectată}}$ la RGB_{obiect}

$C = C + C/L \bullet L$

$C = (100, 0, 0) + (100/255, 0, 0) \bullet (255, 255, 255)$

Pentru variație în iluminare pe suprafață, folosim o variabilă "y"

$C = (100, 0, 0) + (100, 0, 0) \bullet y$, $y \in [0, 1]$ pe suprafața obiectului

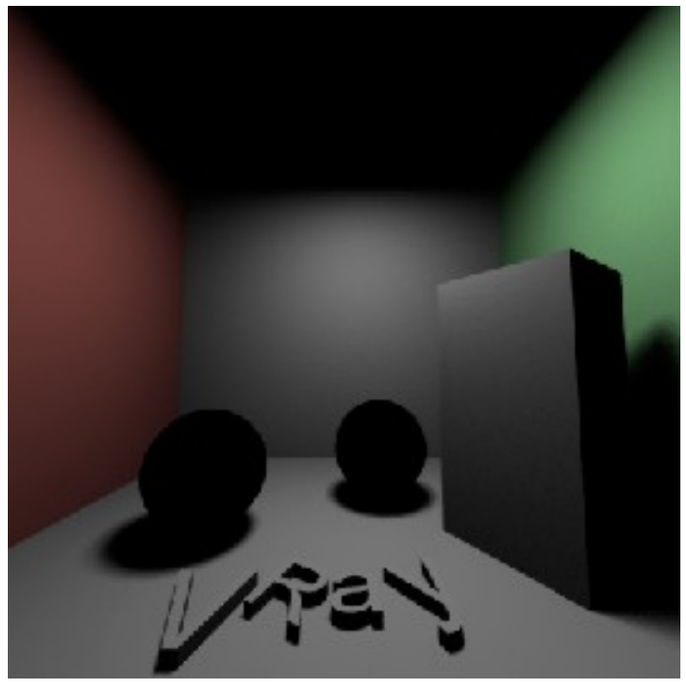
⇓

C o să varieze între (100,0,0) și (200,0,0)

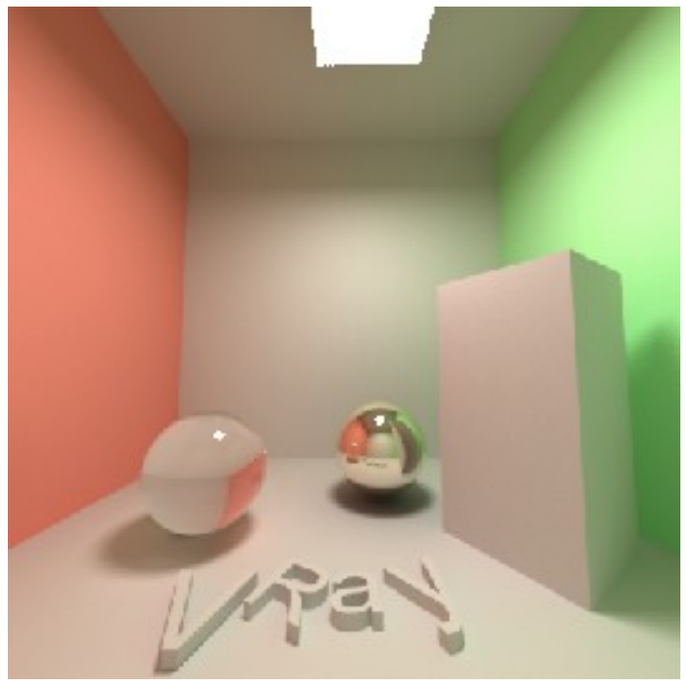
*Limitare valori R, G, B în intervalul [0, 255]

Illumination models

- **Local models**
  - light from direct sources
  - advantages: fast rendering
  - disadvantages: low photorealism

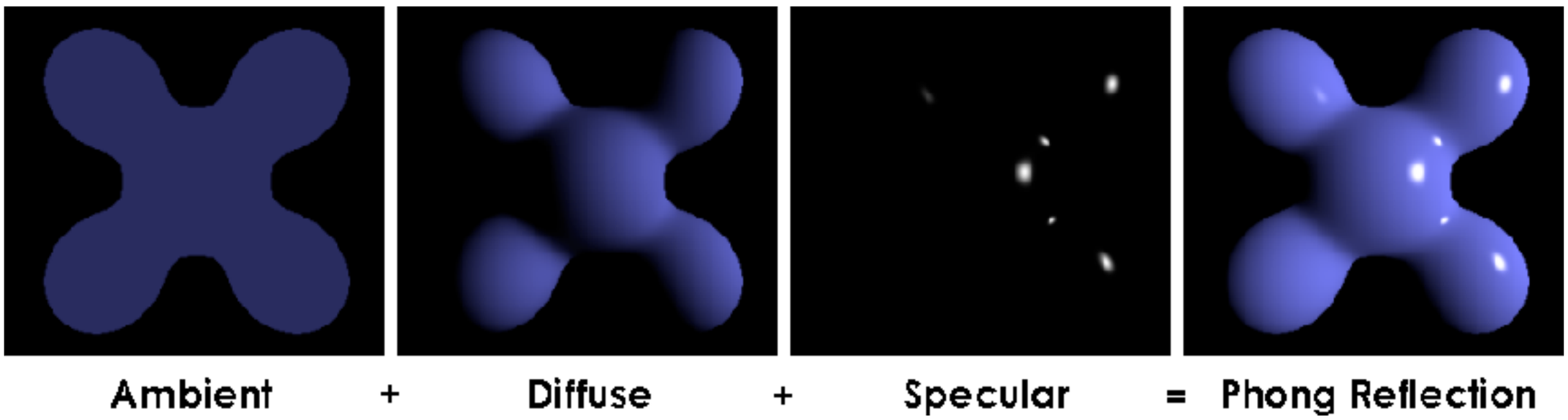


- **Global models**
  - capture all illumination information
  - advantages: shadows, inter-object reflection, refraction, caustics, etc.
  - disadvantages: slow rendering



Phong illumination model

- *Approximate* color as a sum of the following components:
  - **ambient** (approximates the indirect illumination)
  - **diffuse** (color of the object under normal conditions)
  - **specular** (highlights on shiny objects)



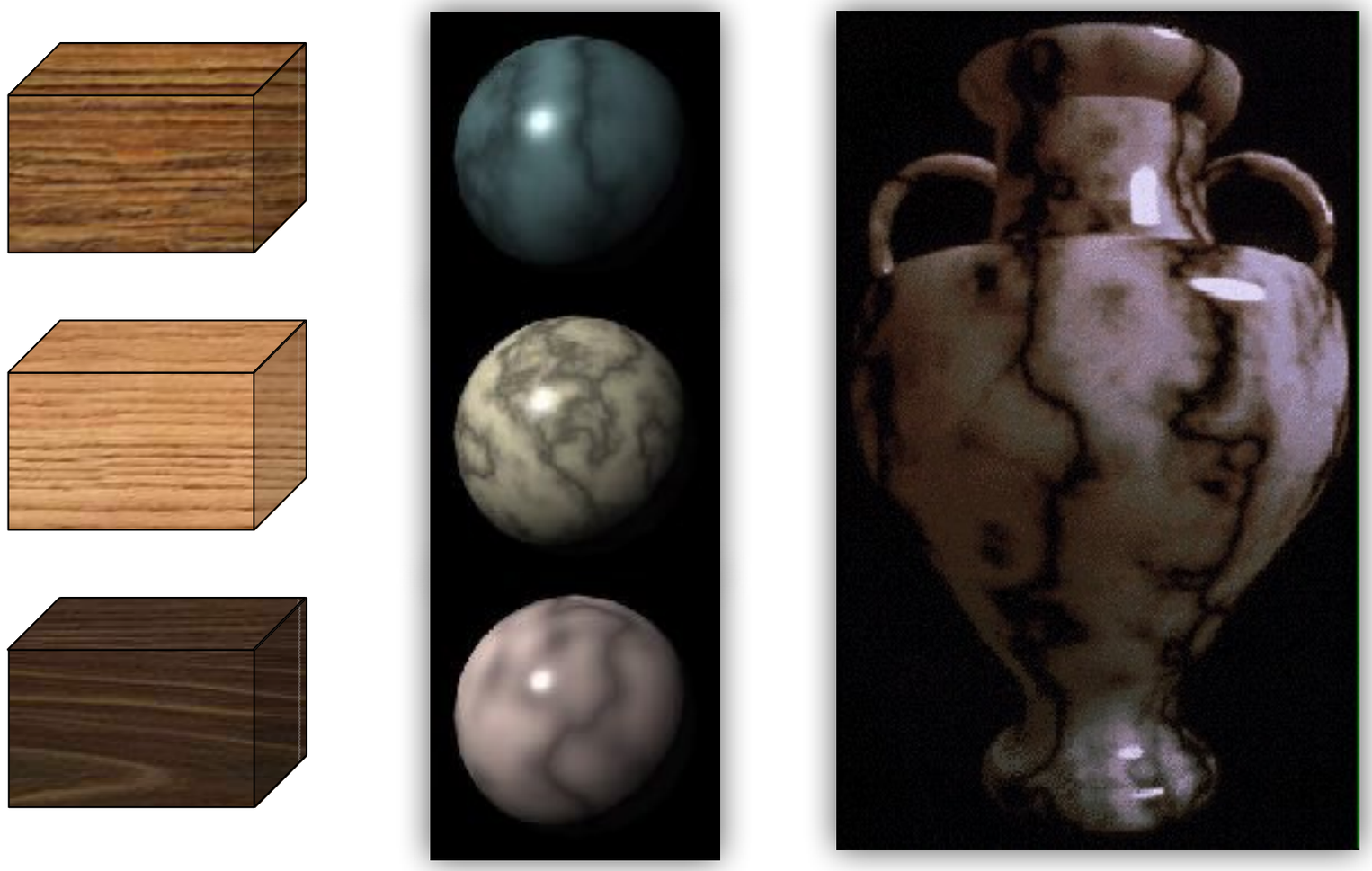
[http://upload.wikimedia.org/wikipedia/commons/6/6b/Phong_components_version_4.png]

Texture mapping

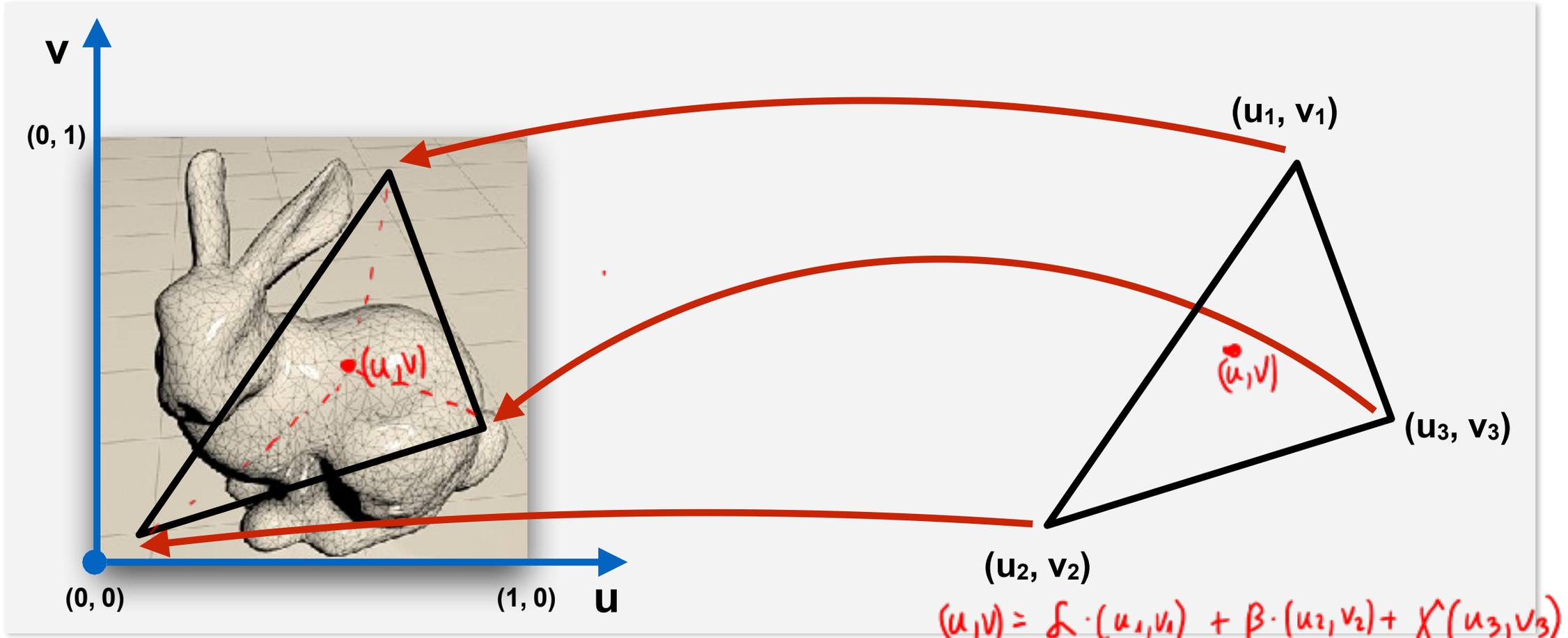
- Used to add detail to the appearance of an object (by changing reflectance)
- 2D Texture Mapping - define a mapping between a surface point and the associated texture map
- Texture coordinates
  - specified for every vertex
  - interpolated during the rasterization stage

Texture mapping

- Texture
  - 2D – graphics pattern based on matrix of pixel
  - 3D – spatial definition of the graphics pattern



2D Texture mapping



640 x 480

1920 x 1080

Ex:

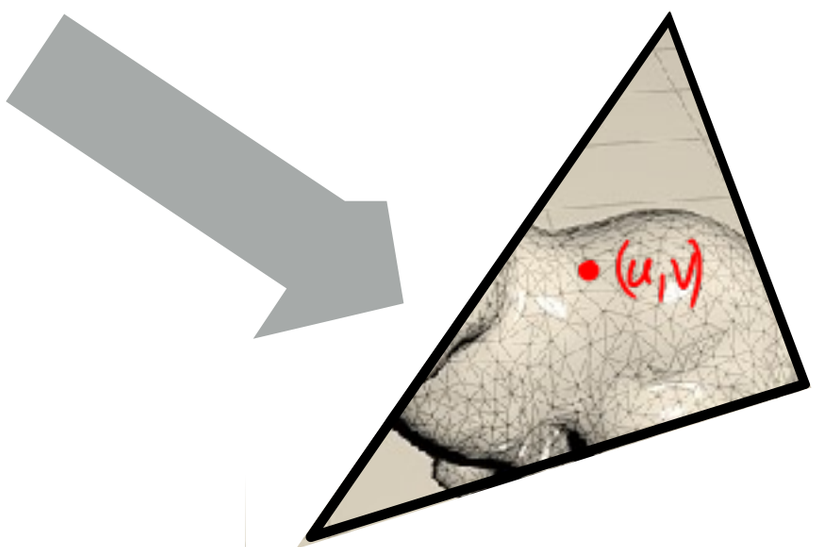
(u,v) = (0.5, 0.5)

⇓

(320, 240)

hay

(960, 540)

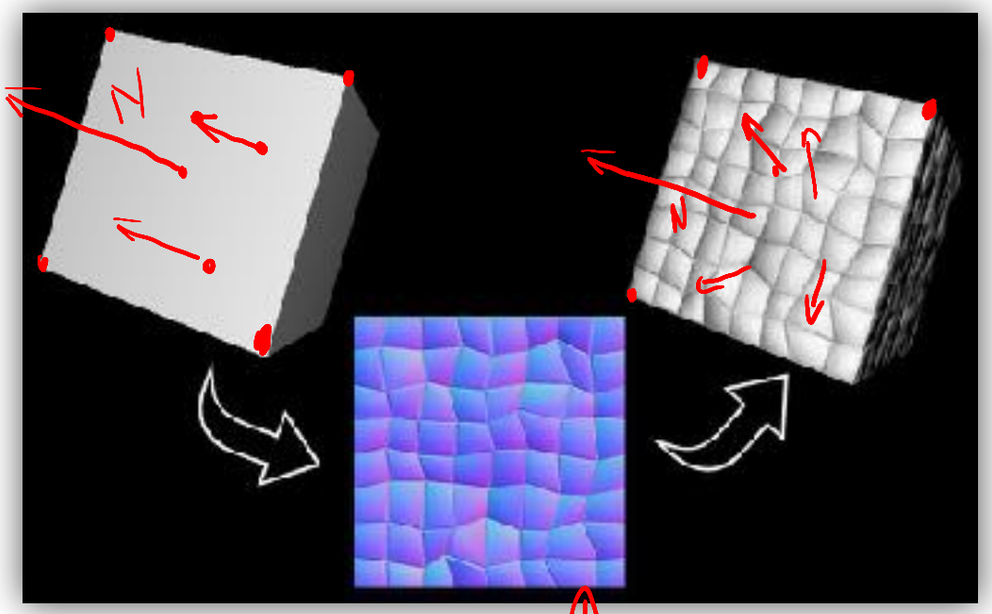


Texture mapping

[http://andysciro.com/wp-content/uploads/2010/03/1268515673184.jpg]

Normal mapping

- Encodes normals as RGB values (stored as a texture)
- Replace normal at pixel with the texture value: (R, G, B) = (x, y, z) of normal
- Preserve the geometry



http://www.independentdeveloper.com/images/normalmapper.jpg

Environment Maps

- Simulates reflections
- Uses a pre-computed texture map of the environment
- Less expensive than raytracing

http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/Theory/mappingimages/V2-cubemap.jpg

Environment Maps

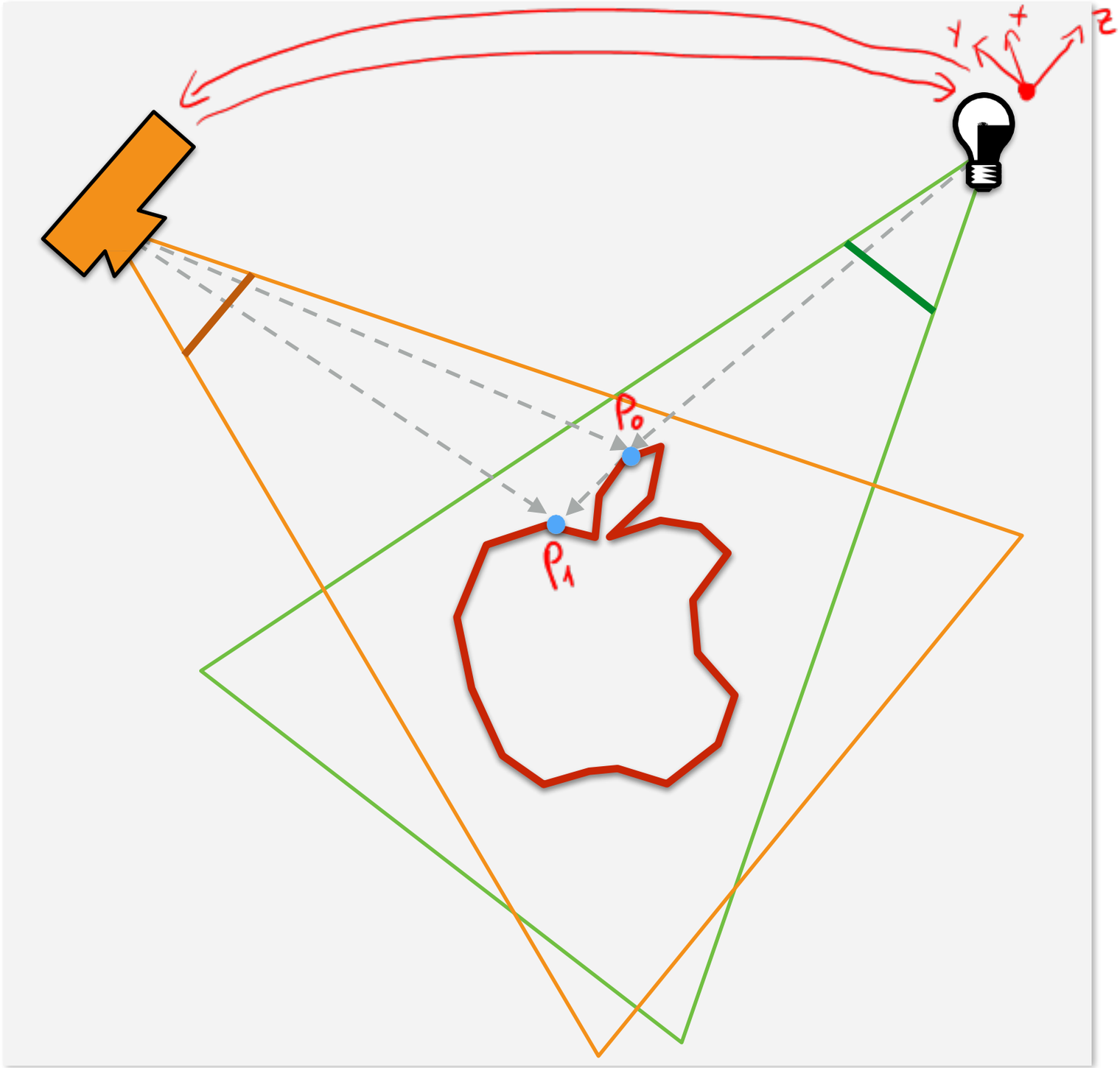
http://1.bp.blogspot.com/-224u1k6KpvM/Upa-1PhC_mI/AAAAAAAASY8/WZ2e6LPdwVg/s1600/Need-for-Speed-Rivals-18-10-2013-1.jpg

Shadow map

- Two step algorithm:

1. **Create shadow map**
   - Render scene from light source
   - Store depth buffer
2. **Render scene from camera**
   - Project fragment to depth buffer/light source
   - If occluder in front → dark
   - Otherwise → bright

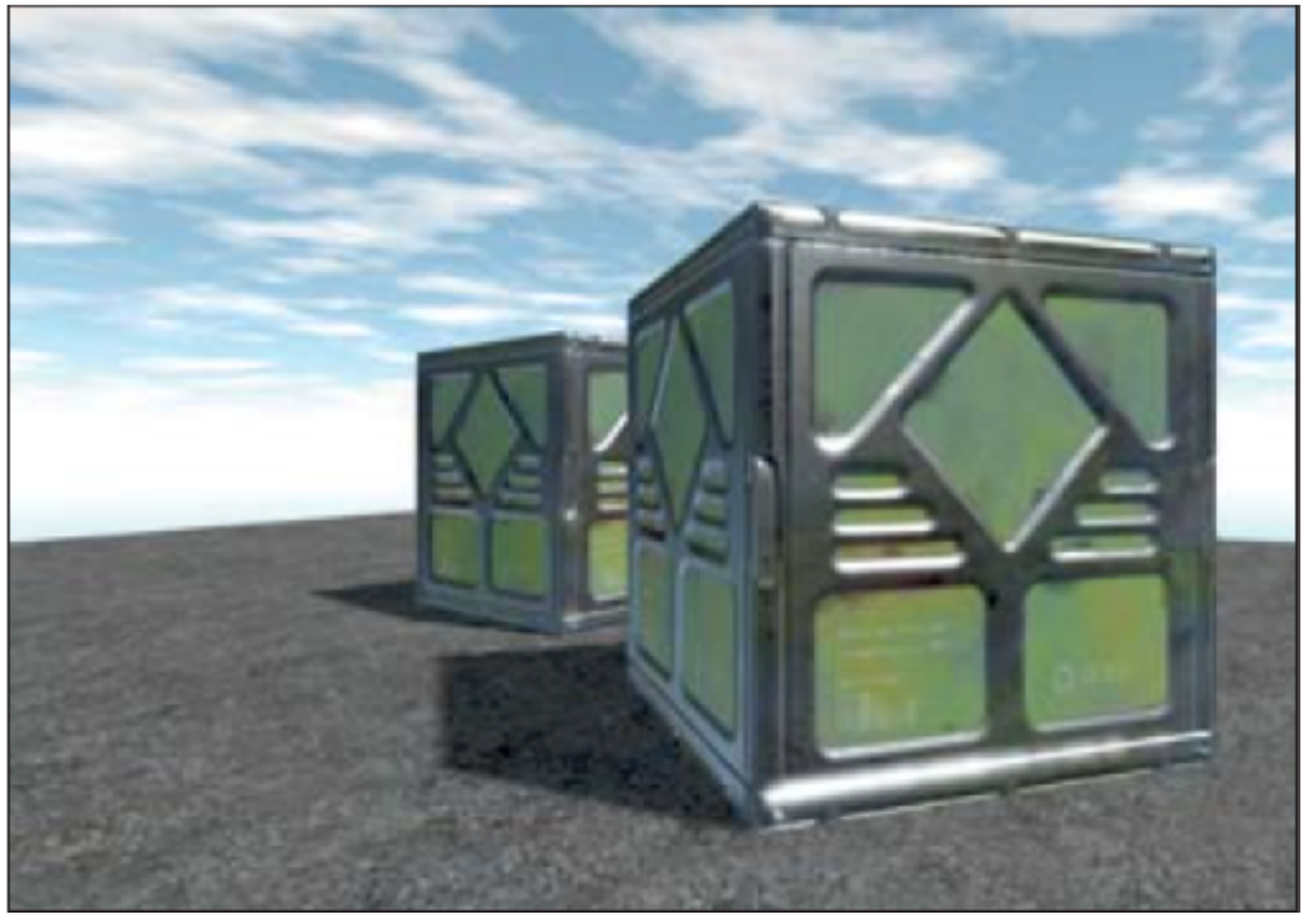
Shadow map



if $z_{P_1} < z_{P_0}$

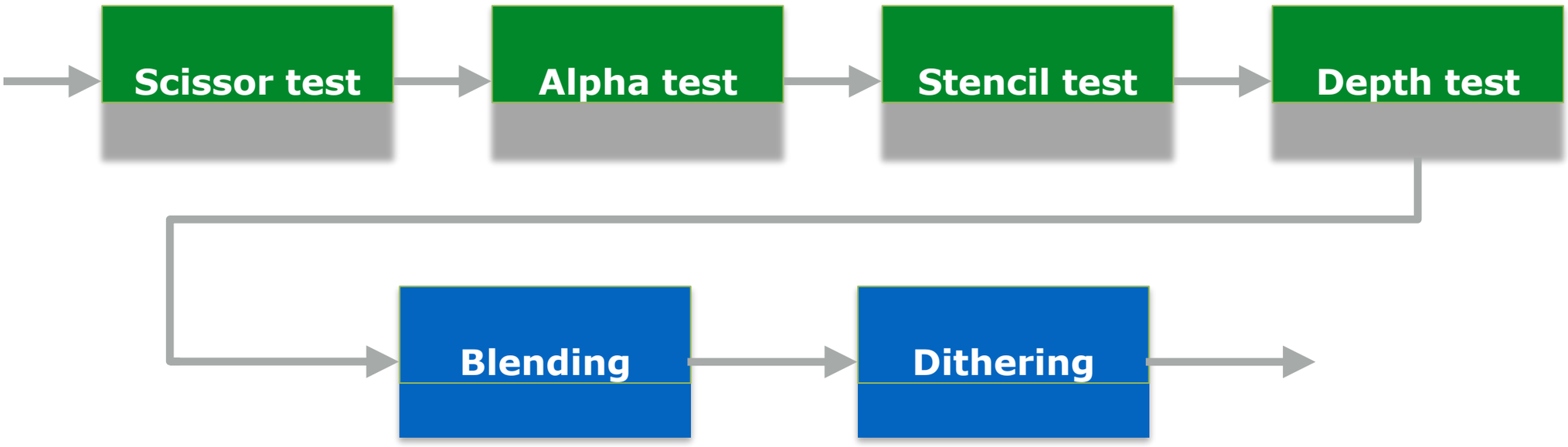
P_1 in umbra

Shadow map

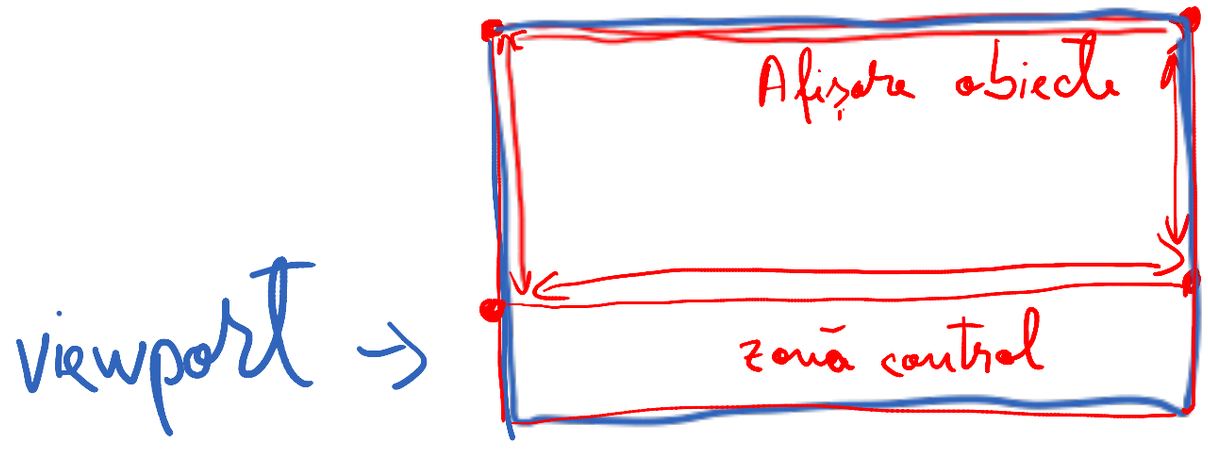


[Source: Computer Graphics: Principles and Practice (3rd Edition), John F. Hughes et al.]

Fragment tests



Scissor test



- Controls fragment visibility and restricts all drawing to a rectangular region
- If a fragment lies inside the rectangle, it passes the scissor test

Alpha test

Color = (R, G, B, α)

α → opacitate

- **Alpha channel**
  - additional color channel (beside the red, green and blue)
  - not a visible color
  - used for defining transparency
- **Operations**
  - compare the alpha value of an incoming fragment to a constant
  - cull all fragments that do not pass the test
  - comparison operators: <, <=, = , >= , >, ≠

Alpha test

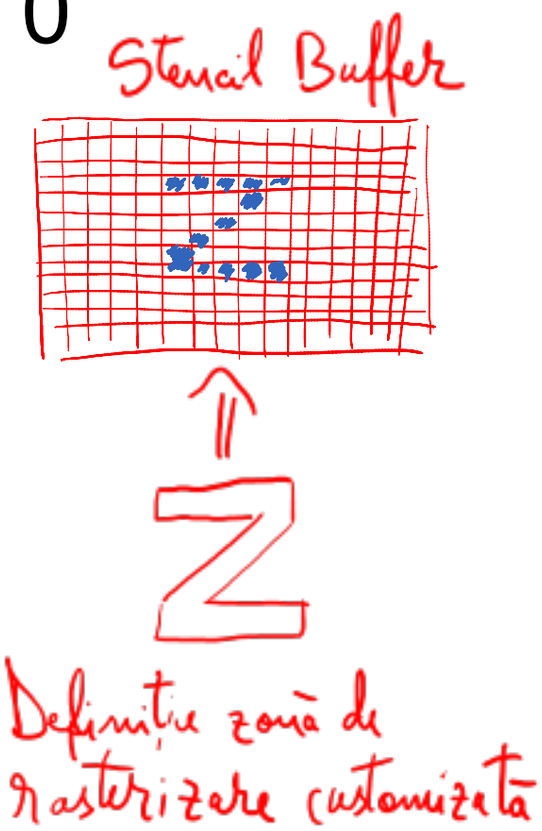
if α < prag => elimină

Stencil test

- **Stencil Buffer**
  - additional buffer
  - provides one channel (typically1..8bit) for every pixel
- **Operations**
  - Counting value for one pixel up or down for every fragment drawn
  - Set stencil value to constant if fragment is drawn
  - Clear/initialize value for all pixels
  - Perform same tests as for alpha (<,<=,…)

Example of stencil test

- Rendering of Decals (e.g.: street demarcations)
  - Clear stencil buffer to 0
  - Draw decal polygons and set stencil to 1 for all pixels drawn
  - Draw street polygon only where stencil = 0



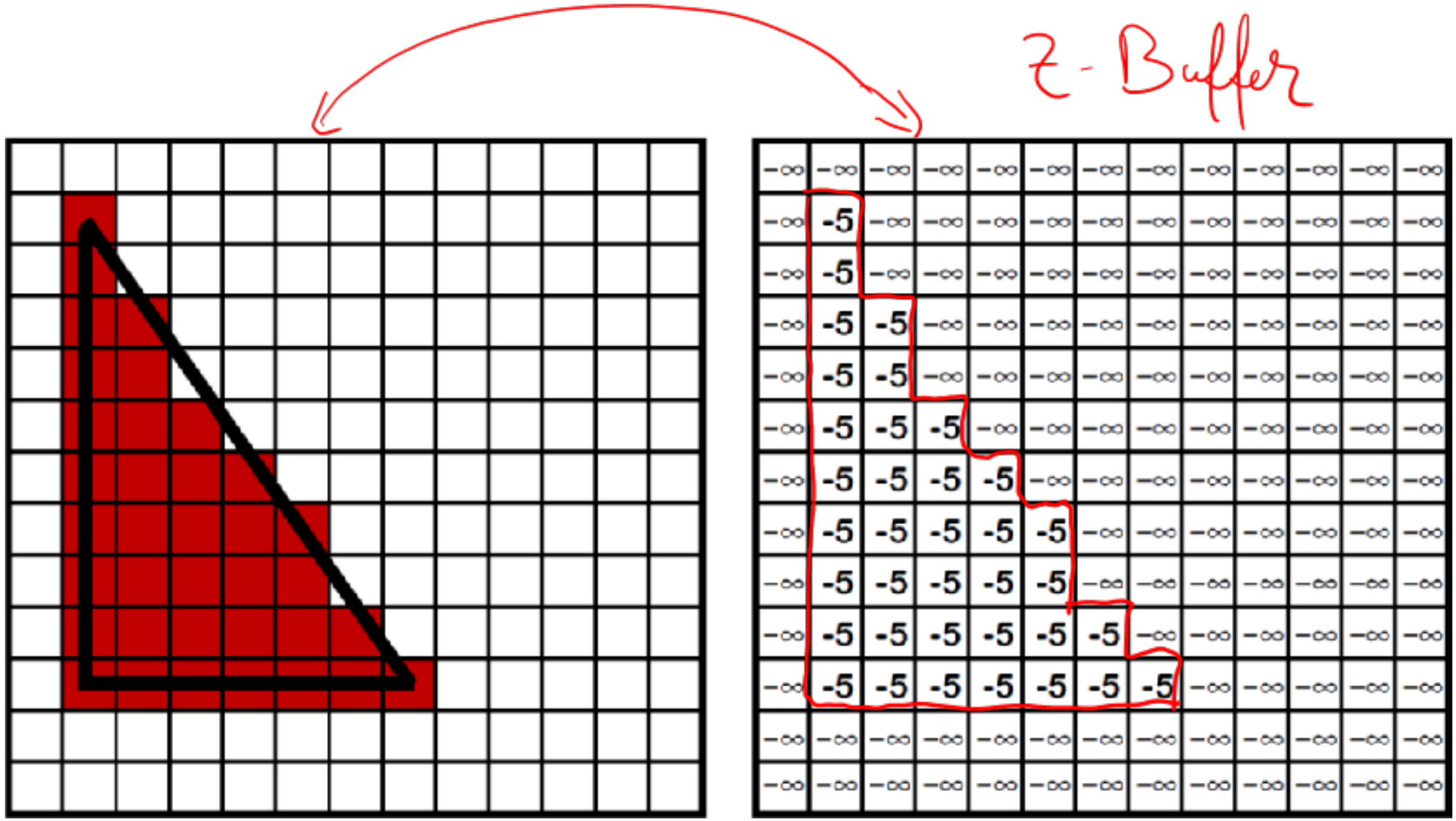
Depth test

- The depth buffer keeps track of the distance between the viewpoint and the object occupying a particular pixel
- If the depth test passes, the incoming depth value replaces the value already in the depth buffer
- The depth buffer is used for hidden-surface elimination

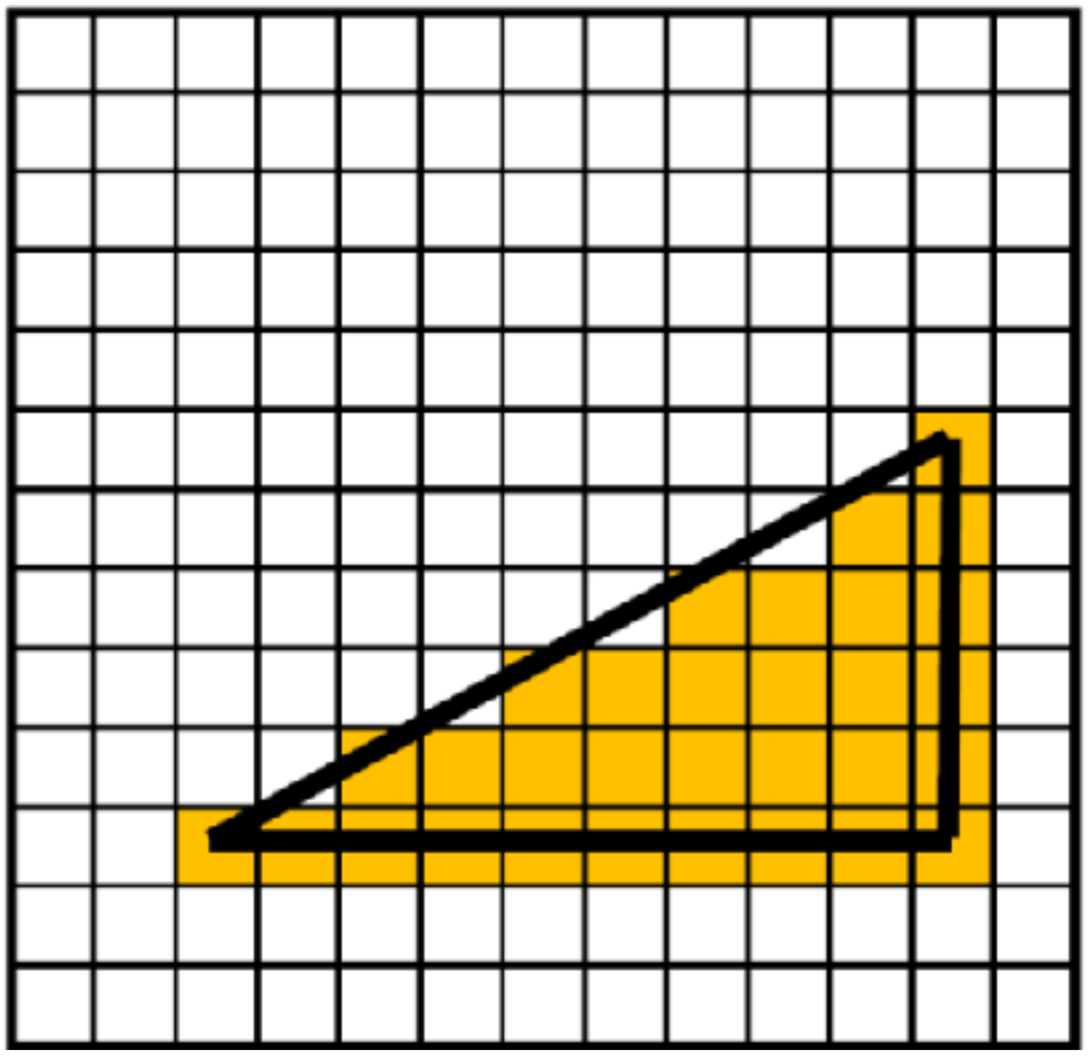
Z-Buffer

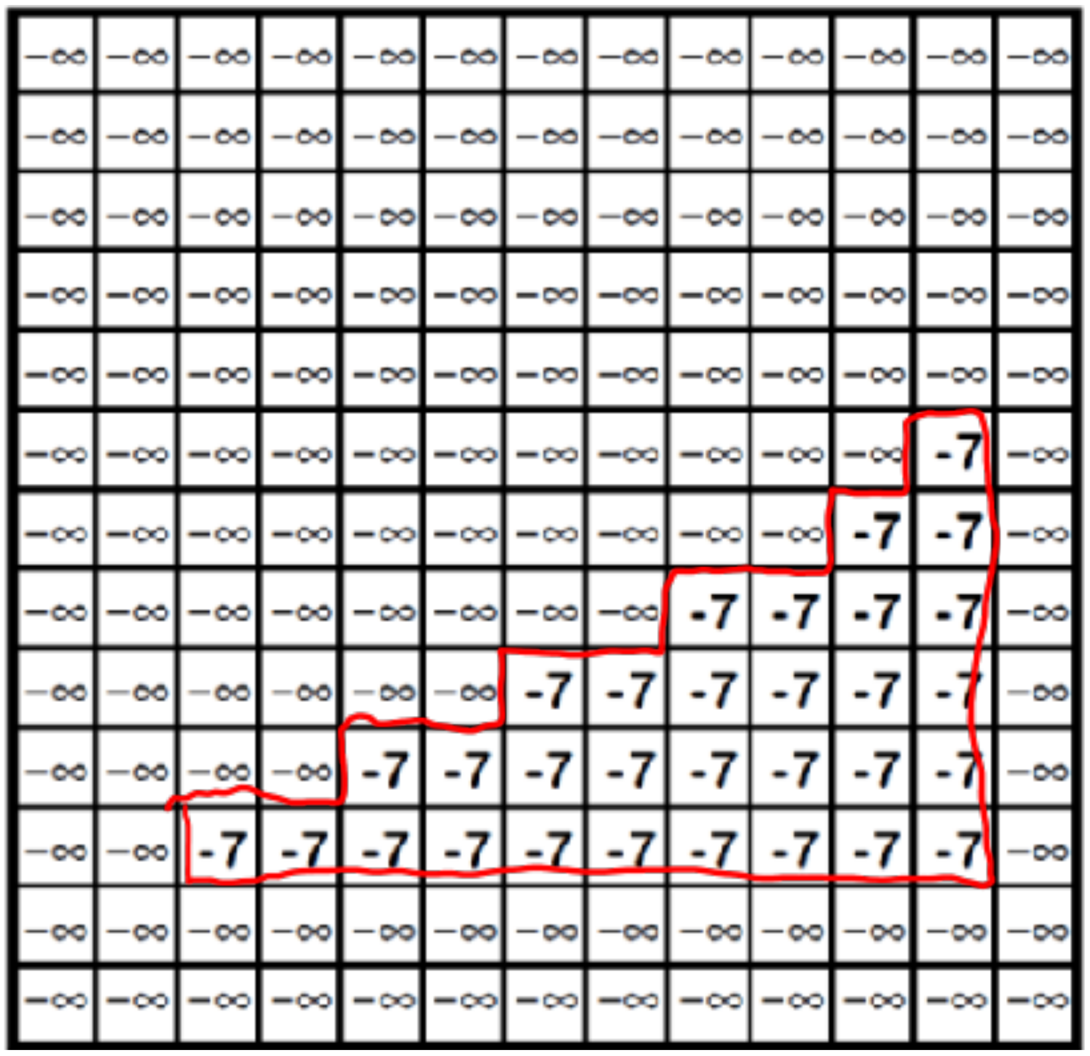
```
foreach pixel(i, j)
    zbuffer[i, j] = FAR
    color_buffer[i, j] = background_color

foreach triangle T
    foreach pixel(i, j) covered by T
        z = compute depth
        if(z > zbuffer(i, j))
            zbuffer[i, j] = z;
            color_buffer[i, j] = compute color
```

Depth test

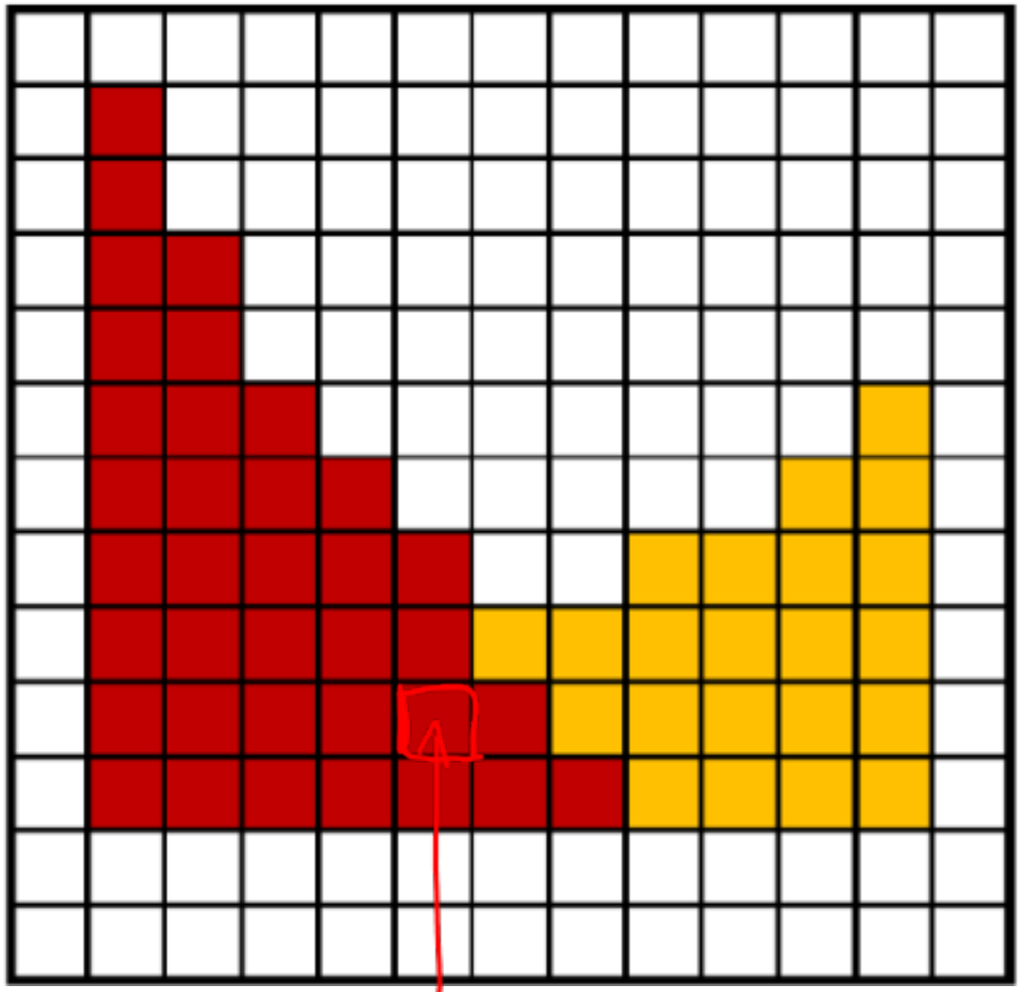


Depth test




$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ -7 $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ -7 -7 $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ -7 -7 -7 -7 $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ -7 -7 -7 -7 -7 -7 $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ -7 -7 -7 -7 -7 -7 -7 -7 $-\infty$
$-\infty$ $-\infty$ -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$
$-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$ $-\infty$

Depth test



$z = -5 > z = -7$

−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
−∞	-5	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
−∞	-5	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
−∞	-5	-5	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
−∞	-5	-5	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
−∞	-5	-5	-5	−∞	−∞	−∞	−∞	−∞	−∞	−∞	-7	−∞
−∞	-5	-5	-5	-5	−∞	−∞	−∞	−∞	−∞	-7	-7	−∞
−∞	-5	-5	-5	-5	-5	−∞	−∞	-7	-7	-7	-7	−∞
−∞	-5	-5	-5	-5	-5	-7	-7	-7	-7	-7	-7	−∞
−∞	-5	-5	-5	-5	-5	-5	-7	-7	-7	-7	-7	−∞
−∞	-5	-5	-5	-5	-5	-5	-5	-7	-7	-7	-7	−∞
−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞
−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞

Blending

- Combine the fragment color (source color) and the framebuffer color (destination color):
  - **Color = a · SourceColor + b · DestinationColor**
  - Partially transparent surfaces
    - Use alpha channel (alpha= 0.1 means we see what is behind the object with 10% of the intensity, and the object itself with 90% intensity)
    - a = 1 - Source Alpha and b = SourceAlpha

Blending

- All objects behind the transparent object must have been already drawn when the transparent object is rendered
- Draw all opaque objects first, then draw transparent objects in back-to-front order – is required a depth sort of polygons