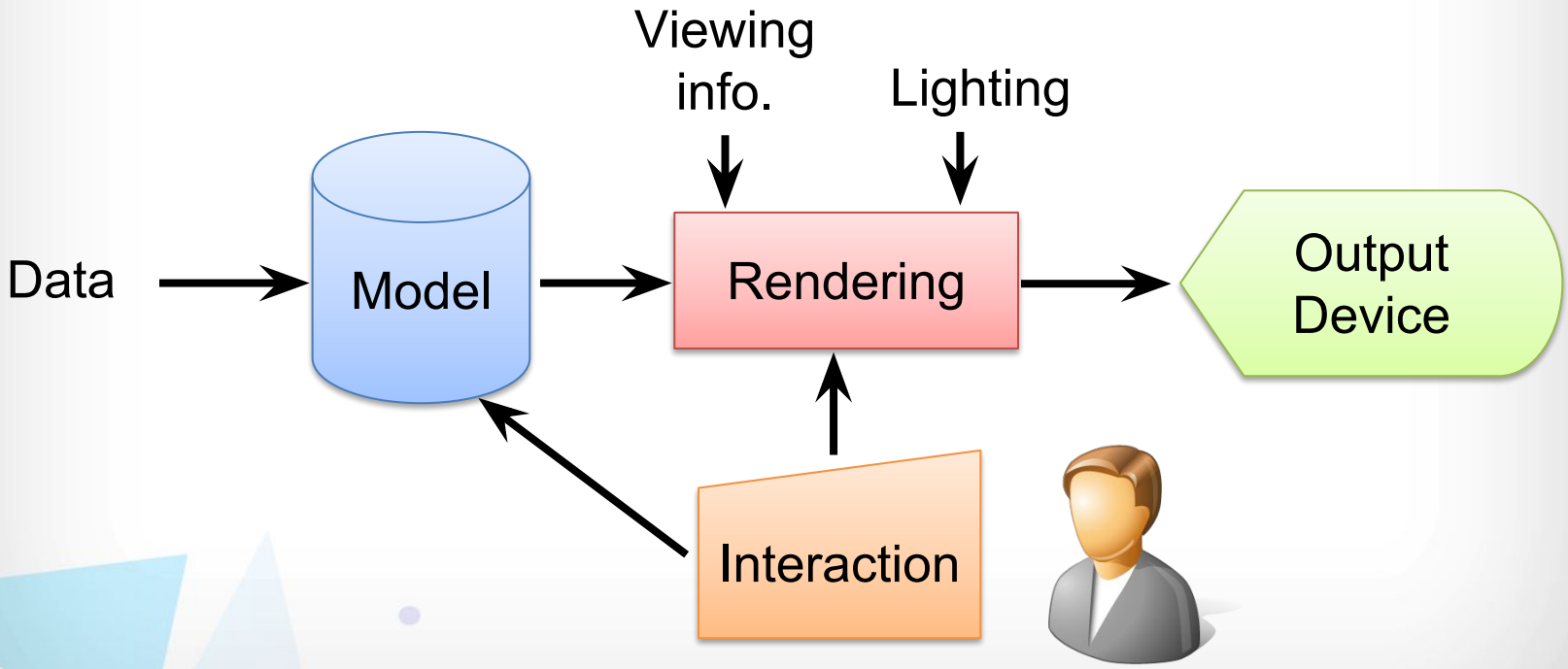
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

CG Basics I

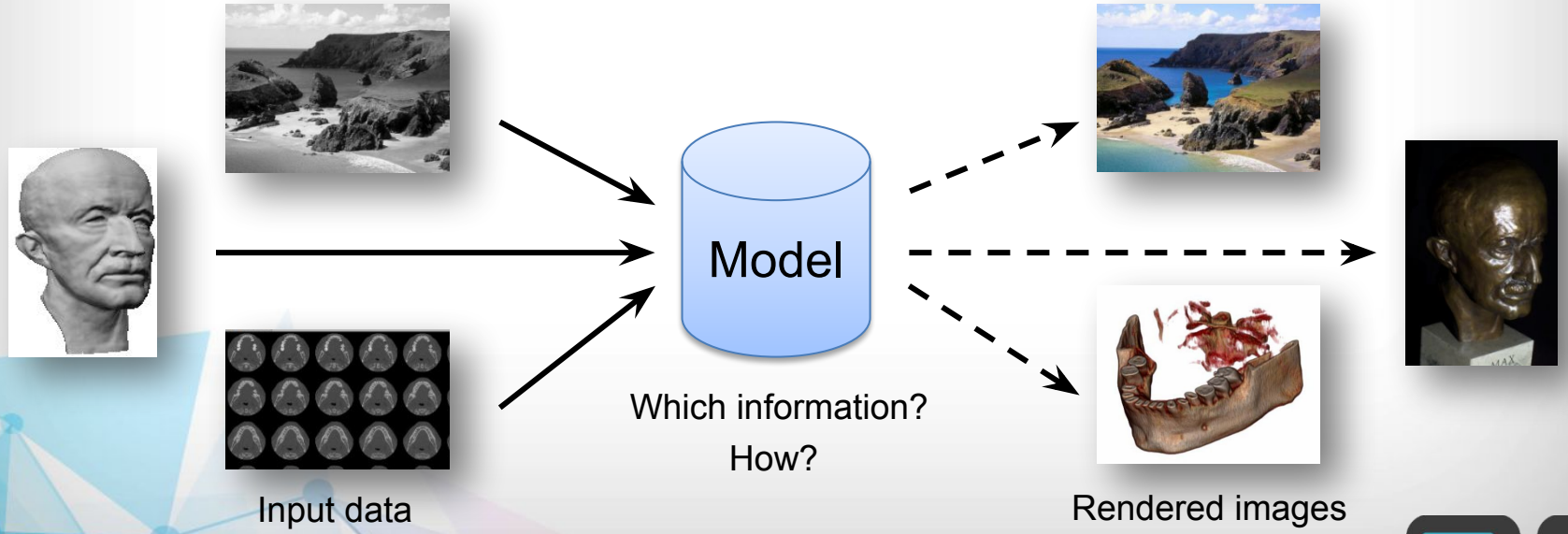
Image Synthesis

- **Functional diagram**



Models

- "Virtual" representation of the input data
  - Geometric information: vertices, edges, polygons…
  - Colors, materials, textures…
  - Intensity or density values…

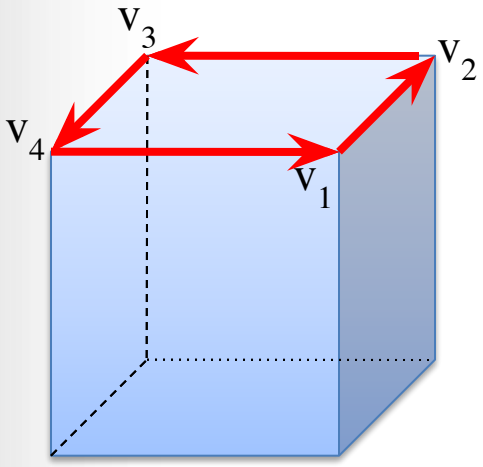


Models

- In this course we will work with Polygonal Models
  - We should store geometric information
    - Faces, edges and vertices
  - Also topological information
    - Adjacency information between geometric element
  - They must be valid:
    - Oriented faces
    - Each edge limits 2 faces
    - Closed and simple models
    - Vertices sorted according to the faces' orientation
  - They have associated a Coordinate System
  - To render them: flat faces → triangulation

Models

- **Example 1**



Faces

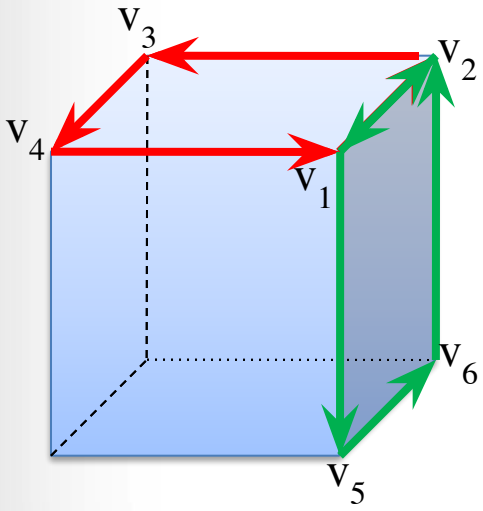
Normal	Nº V	Id 1st V
(a1, b1, c1)	4	1

Vertices

X	Y	Z
	…	

Models

- **Example 1**



Faces

Normal	Nº V	Id 1st V
(a1, b1, c1)	4	1
(a2, b2, c2)	4	5

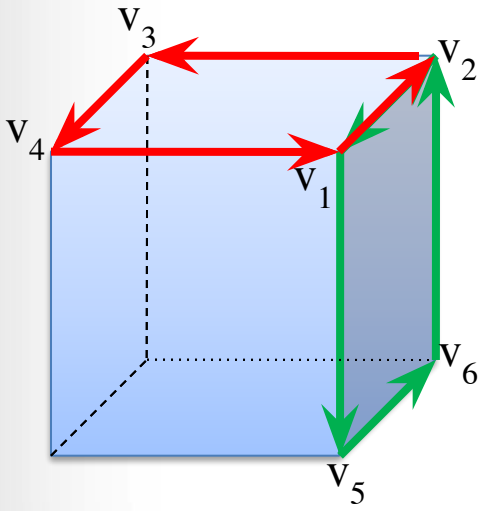
Vertices

X	Y	Z
	…	

Repeated vertices!!

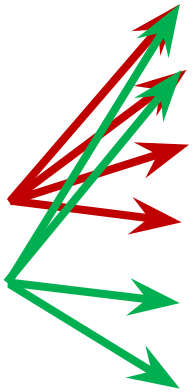
Models

- **Example 2**



Faces

Normal	Id Vertices
(a1, b1, c1)	1, 2, 3, 4
(a2, b2, c2)	1, 5, 6, 2



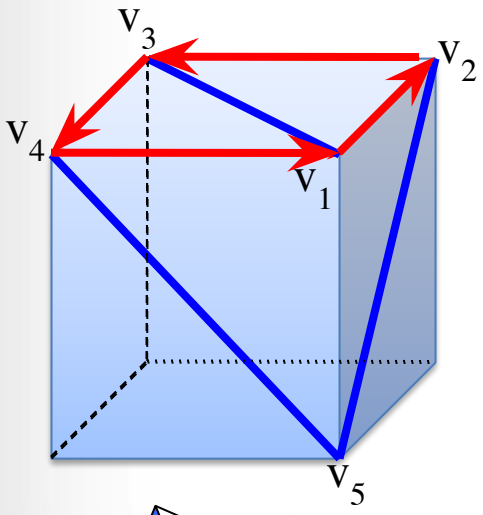
Vertices

X	Y	Z
	…	

Better representation!

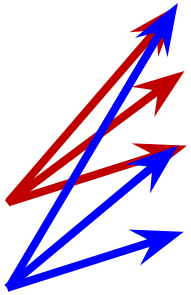
Models

- **Example 3**



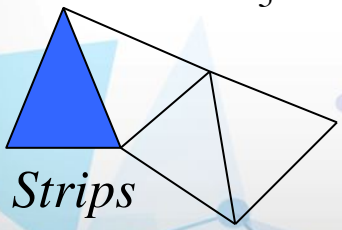
Faces

Normal	Id Successive Vertices
(a1, b1, c1)	2, 3, 1
(a2, b2, c2)	4



Vertices

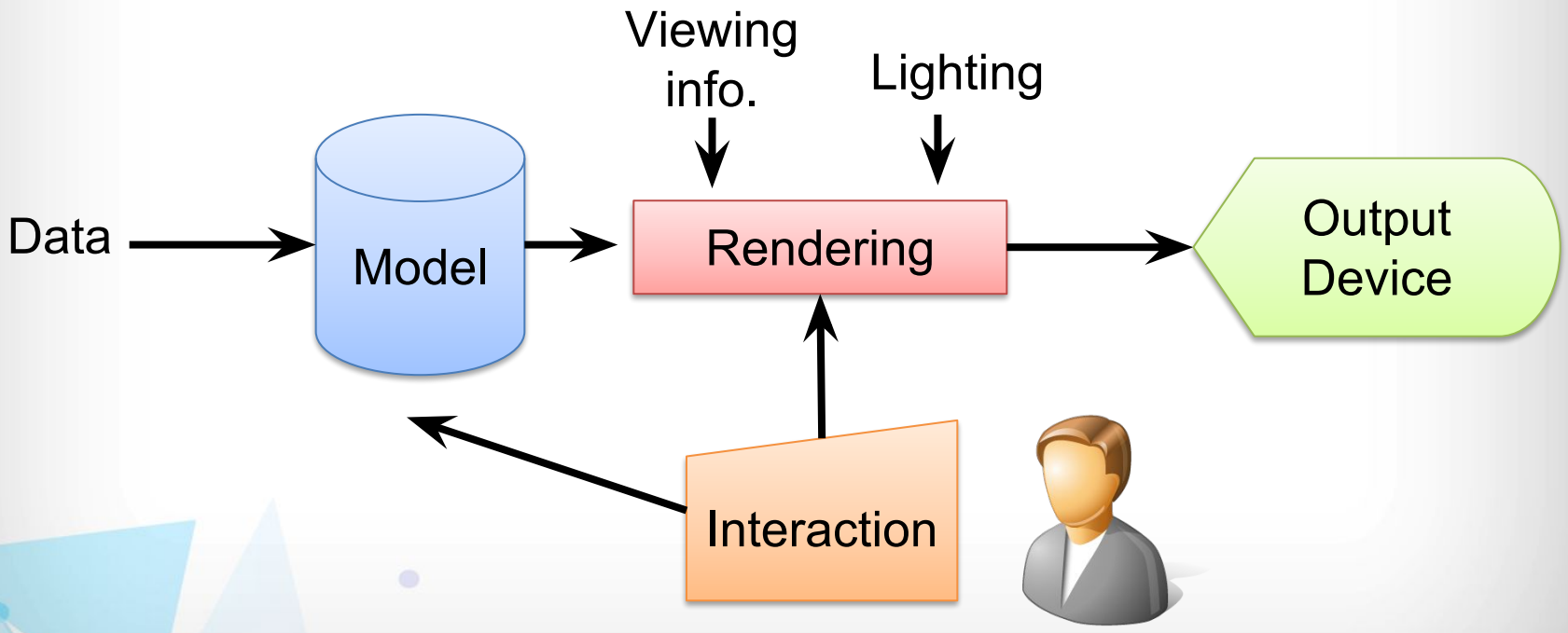
X	Y	Z
	…	



Even better representation!

Image Synthesis

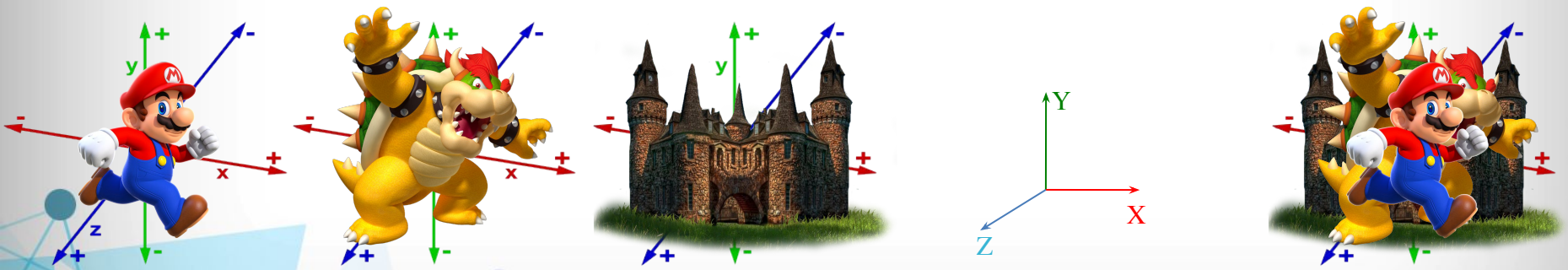
- **Functional diagram**



Transformations

- **Coordinate systems**
  - The elements of each model (vertices, edges, …) are defined according to a local coordinate system
  - In order to render a model, its elements must be transformed to the global coordinate system of the application

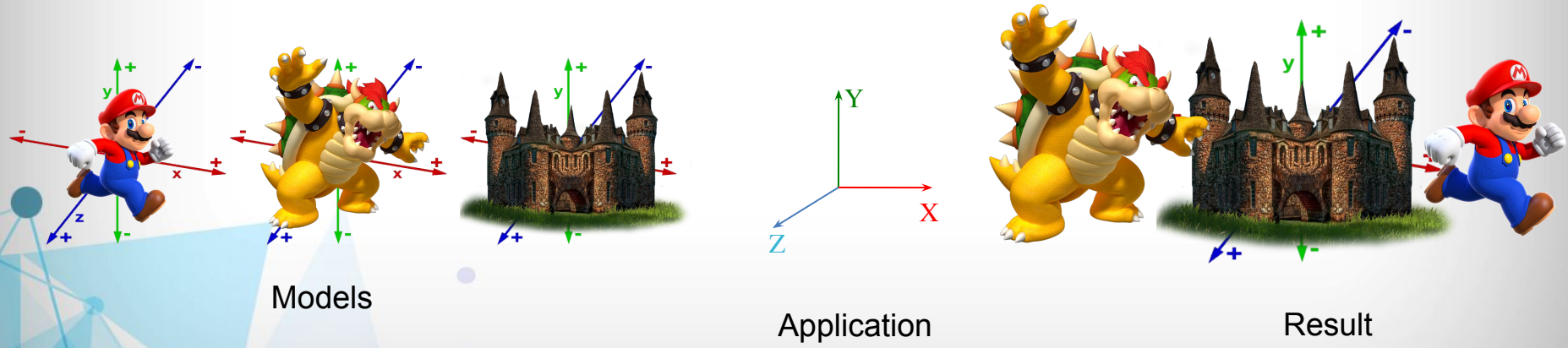
Problem! All CS with the same origin!!



Models Application Result

Viewing info

- **Coordinate systems**
  - Furthermore, they must be placed in the correct position!
  - To do this, we use geometric transformations
    - Translations, rotations and scaling
    - The combination of them can be represented as a 4x4 matrix!



Viewing info

- **Geometric transformations**

$$Translation = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Scaling = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

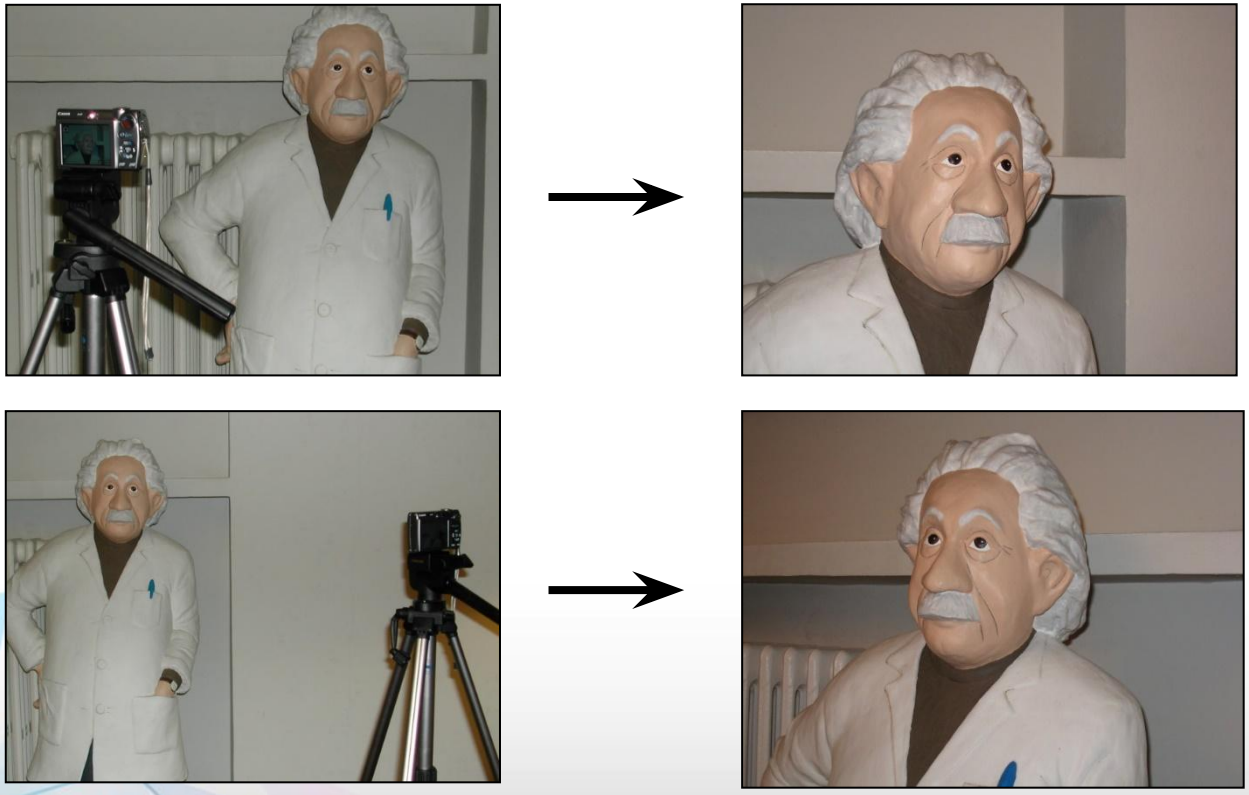
Viewing info

So, to place objects all over the scene…

```
var mvMatrix = mat4.create();
mat4.identity(mvMatrix);
mat4.translate(mvMatrix, [0.0, -1.0, -10.0]);
mat4.rotate(mvMatrix, radians, [0.0, 0.0, 1.0]);
mat4.scale(mvMatrix, [1.0, -1.0, 1.0]);
```

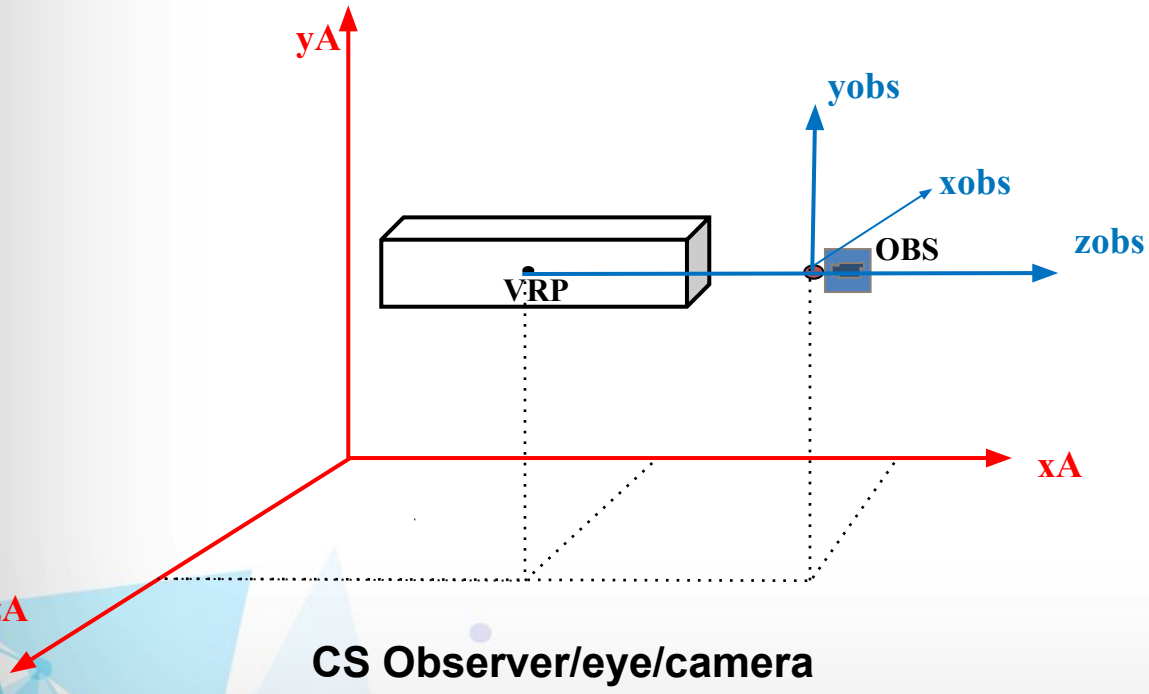
Viewing info

- Viewing information



Viewing info

- OpenGL/WebGL camera



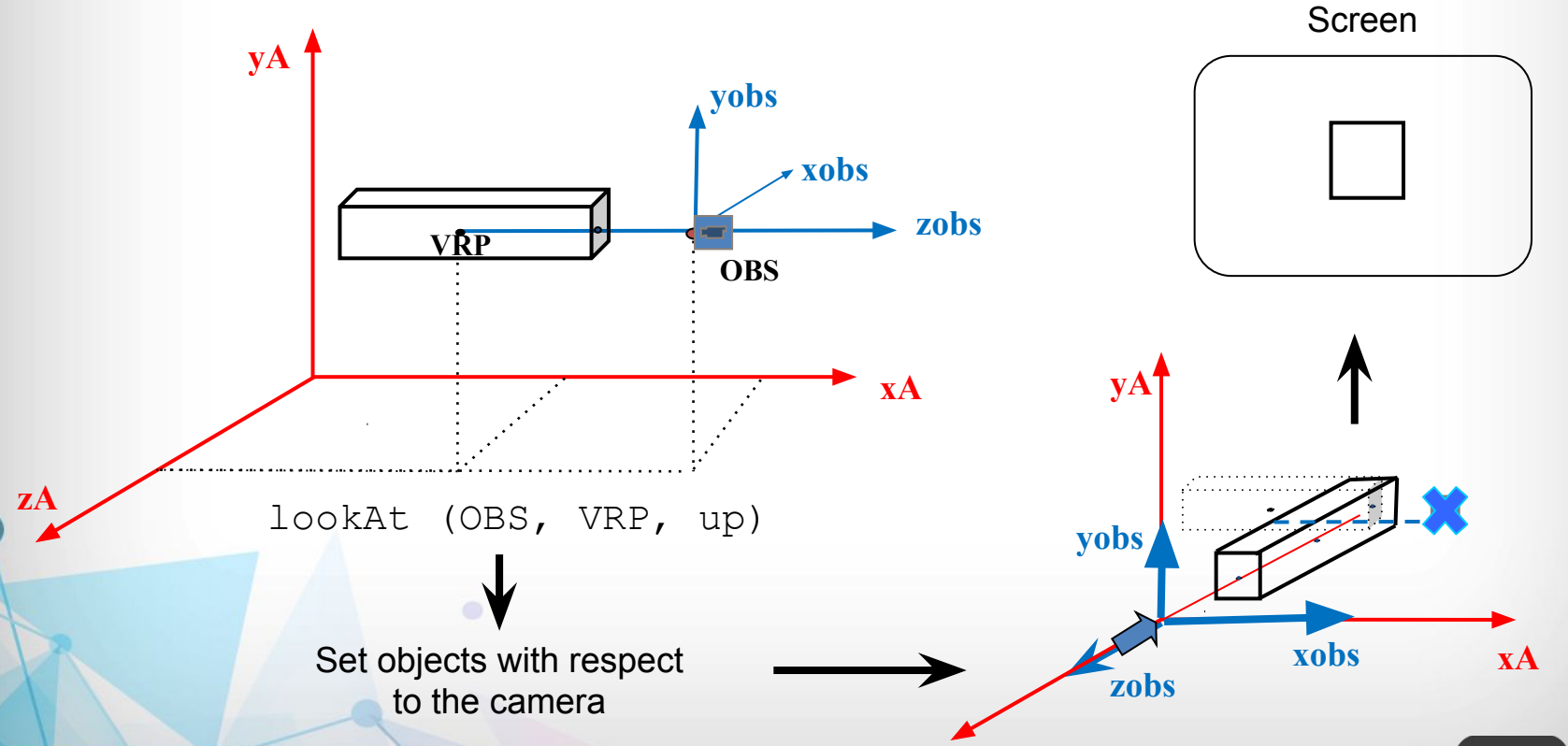
CS Observer/eye/camera

OBS = Observer
VRP = View Reference Point
up = View Up Vector

```
lookAt (OBS, VRP, up)
```

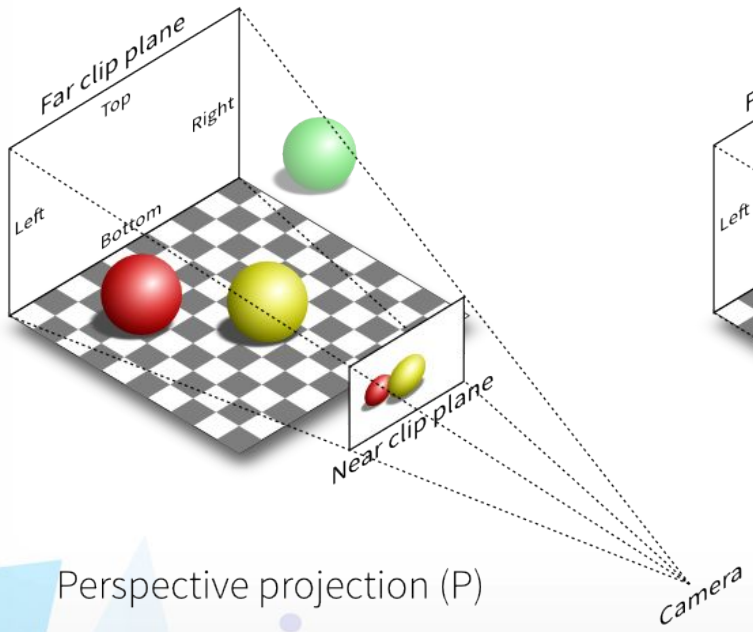
Viewing info

- **View transformation**

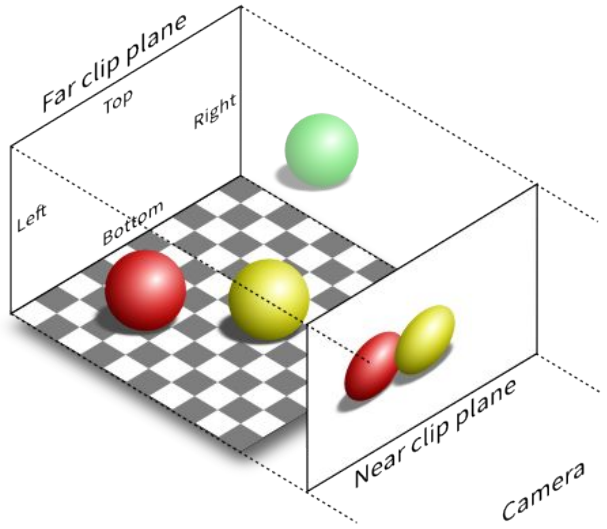


Viewing info

- **Projection transformation**



Perspective projection (P)



Orthographic projection (O)

`perspective (FOV, ra`$_w$`, zN, zF)` `ortho (left, right, bottom, top, zN,zF)`

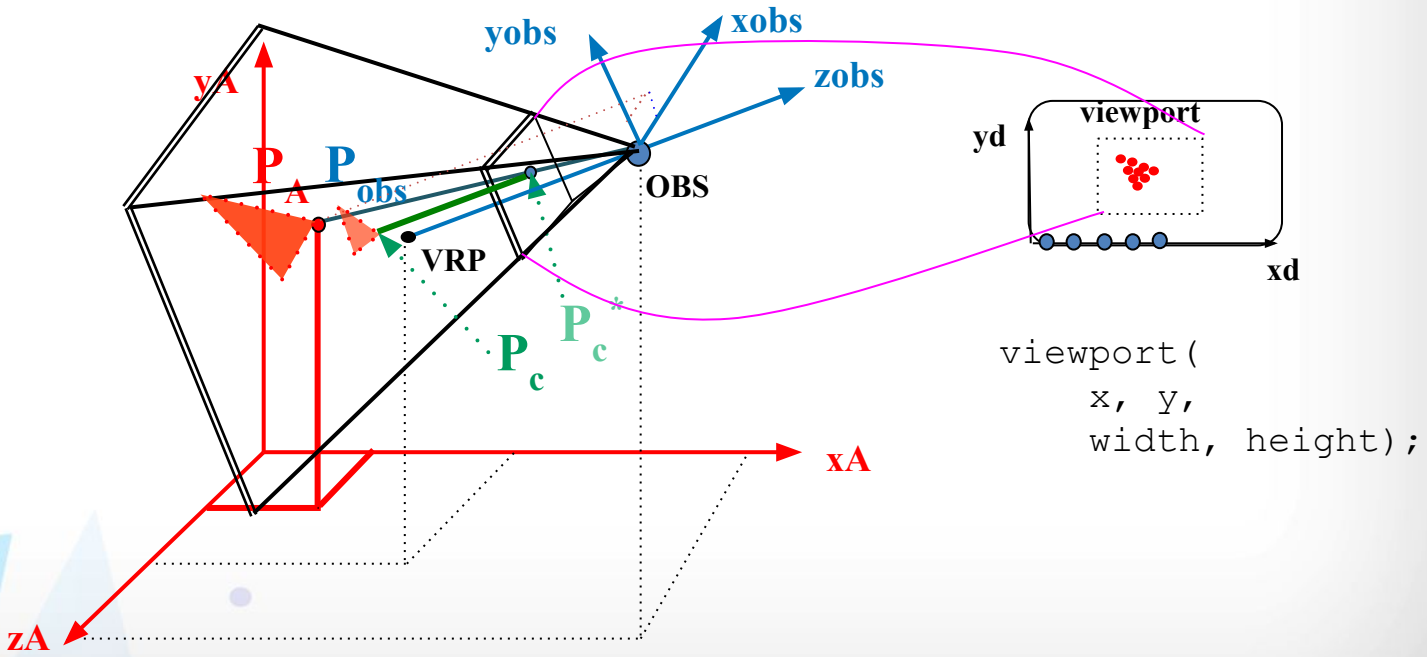
Viewing info

So, to configure the camera position and projection...

- `var vMatrix = mat4.create();`
- `mat4.lookAt(vMatrix, [5.0, 5.0, 5.0], [0.0, 0.0, 0.0], [0.0, 1.0, 0.0]);`
- `var pMatrix = mat4.create();`
- `mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);`

Viewing info

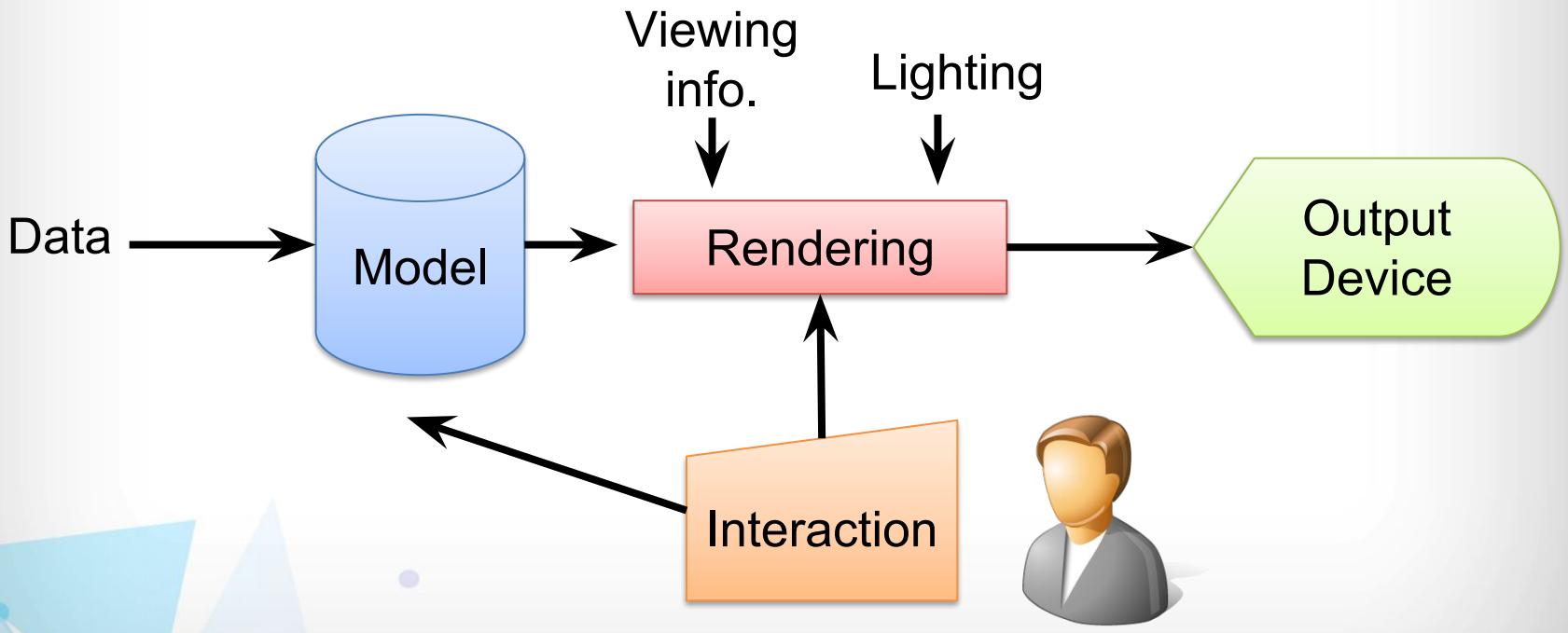
- OpenGL/WebGL pipeline




```
viewport(
    x, y,
    width, height);
```

Image Synthesis

- **Functional diagram**



Rendering

- **Rendering process in WebGL**

1. Clear screen → `clear(gl.COLOR_BUFFER_BIT |…);`

2. Viewing information
2a Set view/viewport → `viewport(…);`

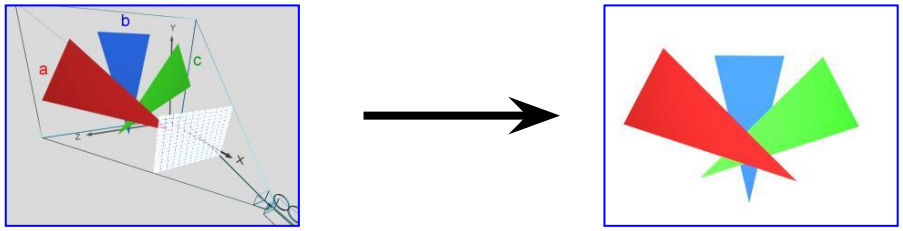
2b Modelview transform (position/orientation) → `lookAt (...)` or geom transforms.

2c Projection Transformation (lens optical properties) → `perspective(…)` or `ortho(…)`

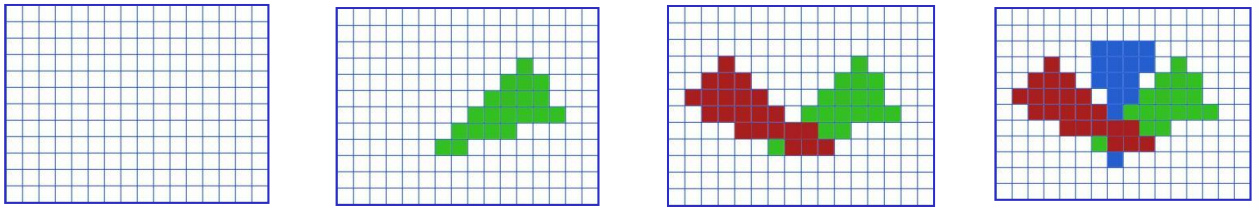
3. Draw the scene → `drawScene(…)`

Rendering

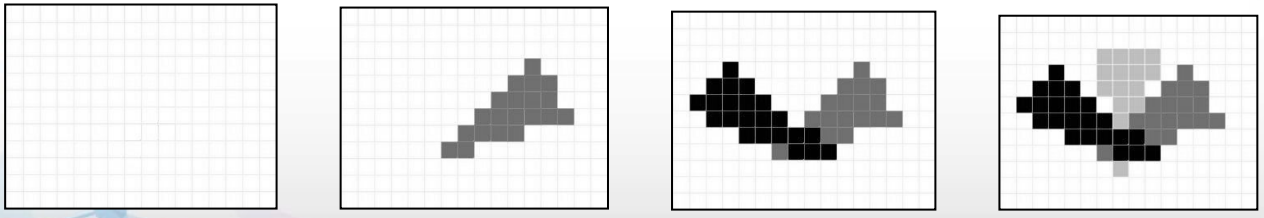
- **Color and depth of the scene**



Color



Depth



Rendering

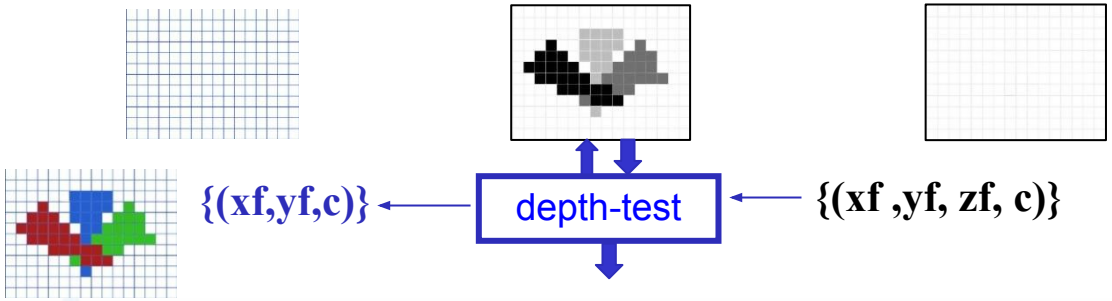
- **Color and depth of the scene**
  - **Stored in 2 buffers with the screen's resolution**

Frame Buffer: $(r, g, b) \in [0, 2^{n-1}]$

1. Init with the background color

Depth Buffer: $z \in [0, 2^{n_z-1}]$

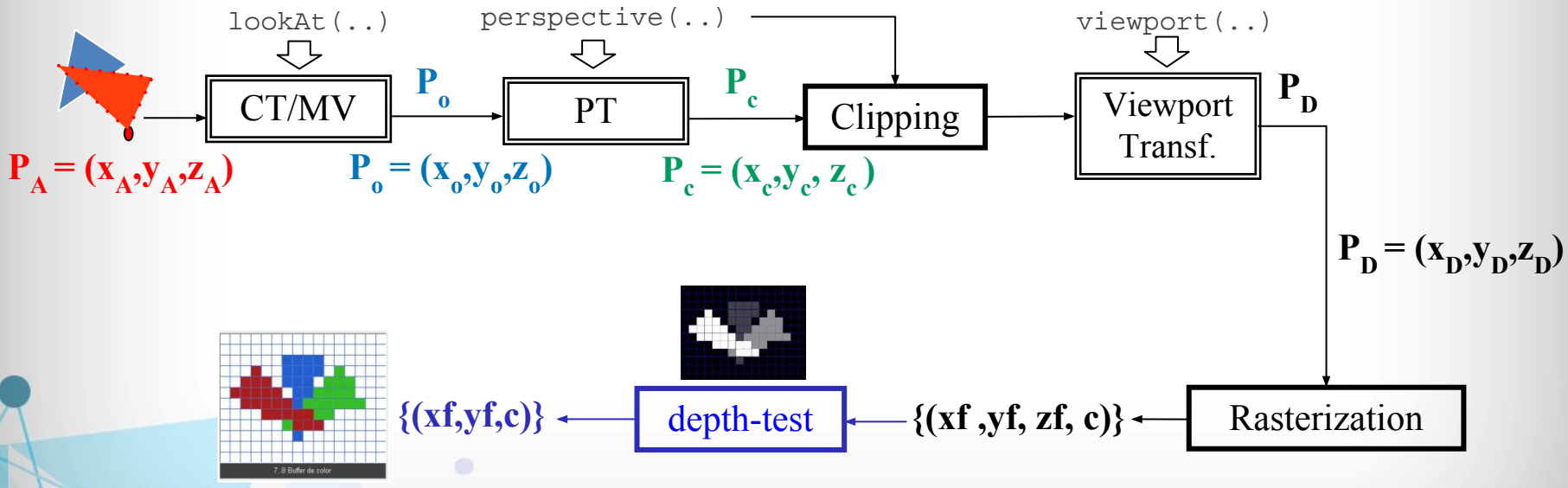
1. Init with the furthest value




```
if (zf < depth_buffer(xf,yf)){
        depth_buffer(xf,yf) = zf;
        frame_buffer(xf,yf) = c;
 }
```

Rendering

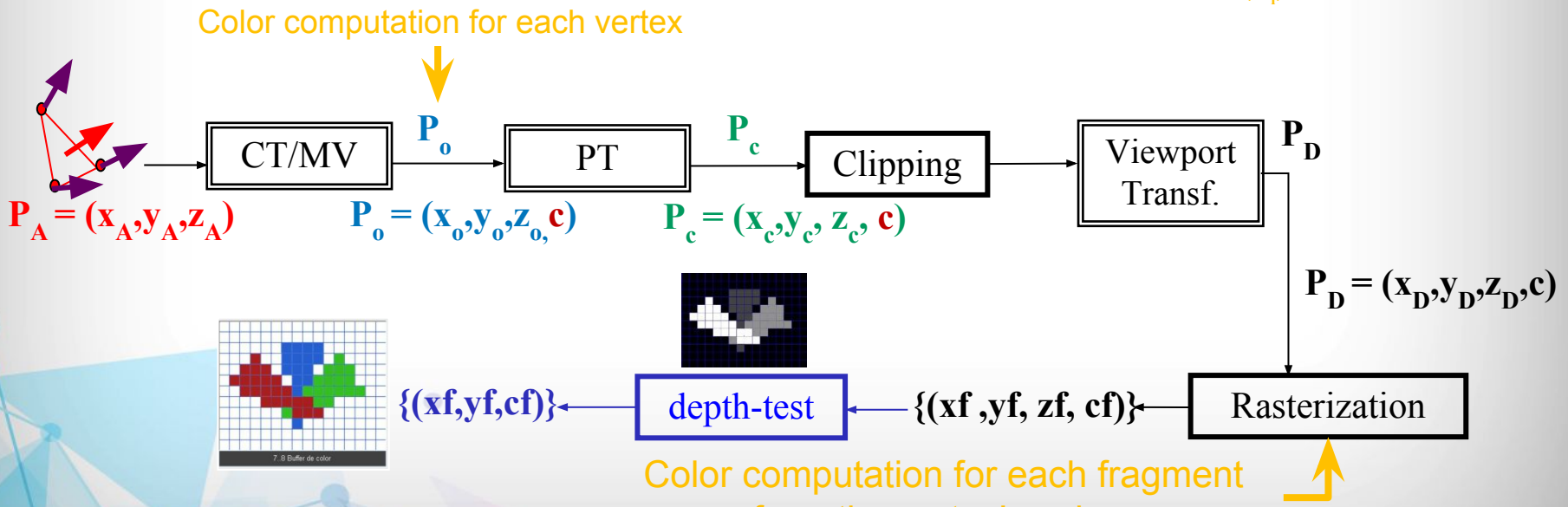
- OpenGL/WebGL pipeline



Rendering

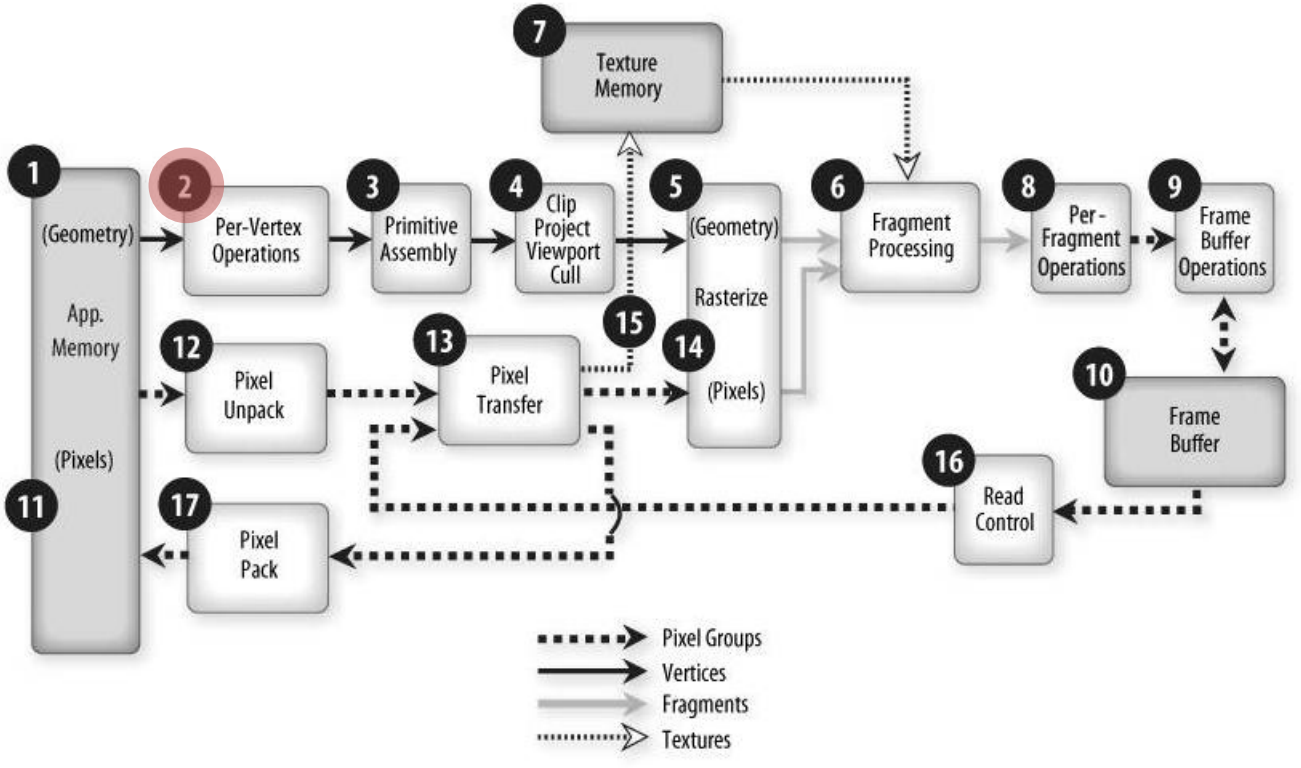
- **Lighting in OpenGL/WebGL**
  - Active light sources (position, direction, color)
  - Ambient light
  - Object material

$$I_\lambda(P) = I_{a\lambda}\, k_a + \Sigma_i\, (I_{fi\lambda}\, k_d \cos(\Phi_i)) + \Sigma_i\, (I_{fi\lambda}\, k_s \cos^n(\alpha_i))$$
$$|\Phi_i| < 90^o$$



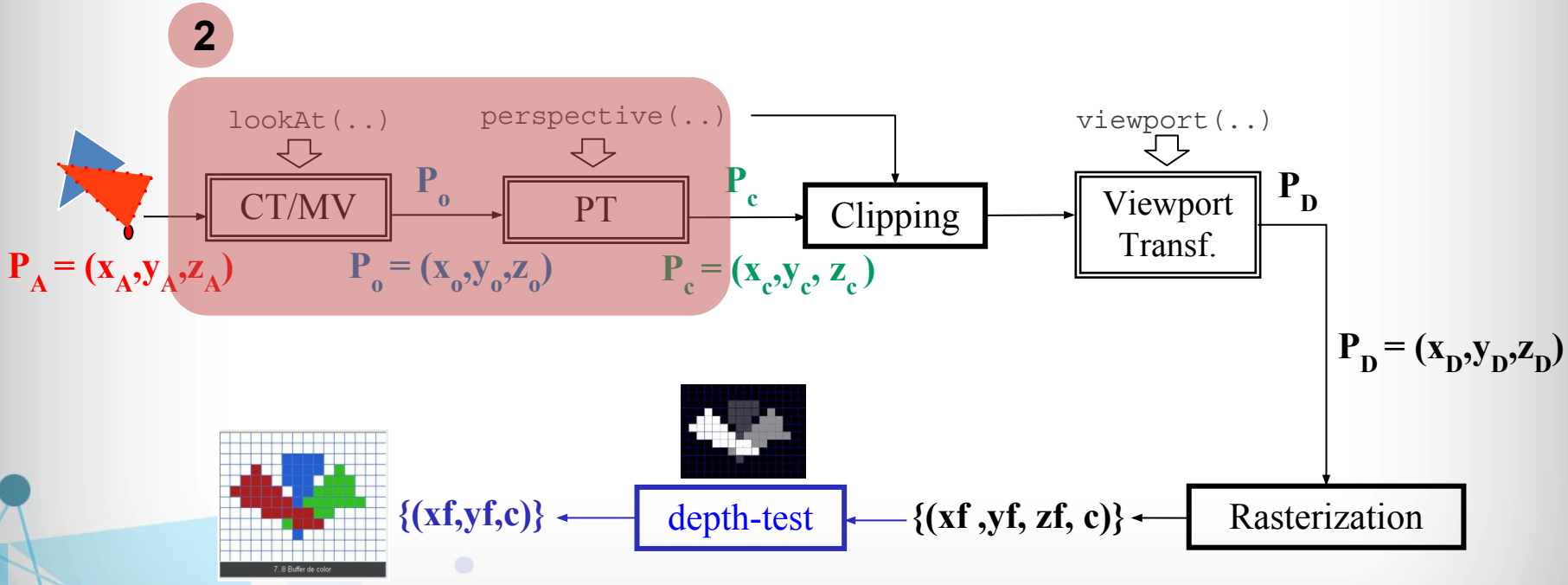
Rendering

- OpenGL/WebGL fixed pipeline



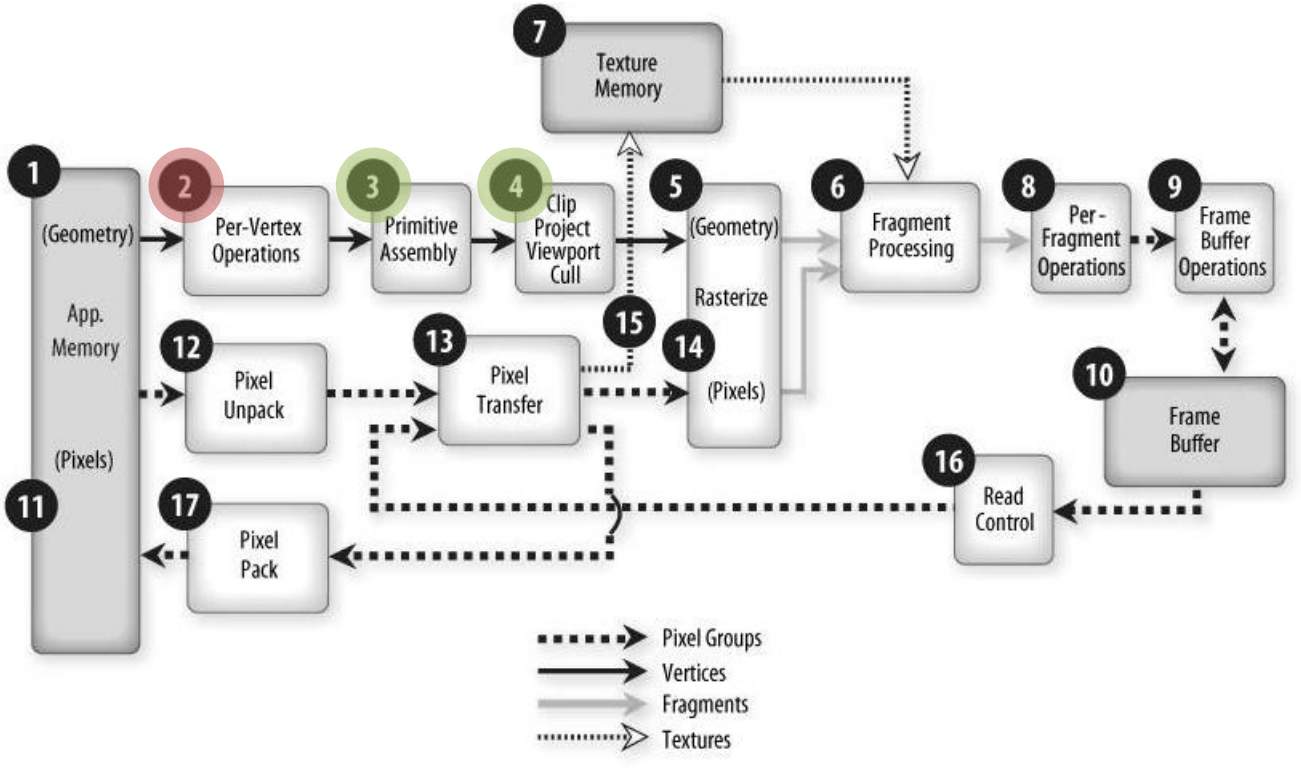
Rendering

- OpenGL/WebGL fixed pipeline



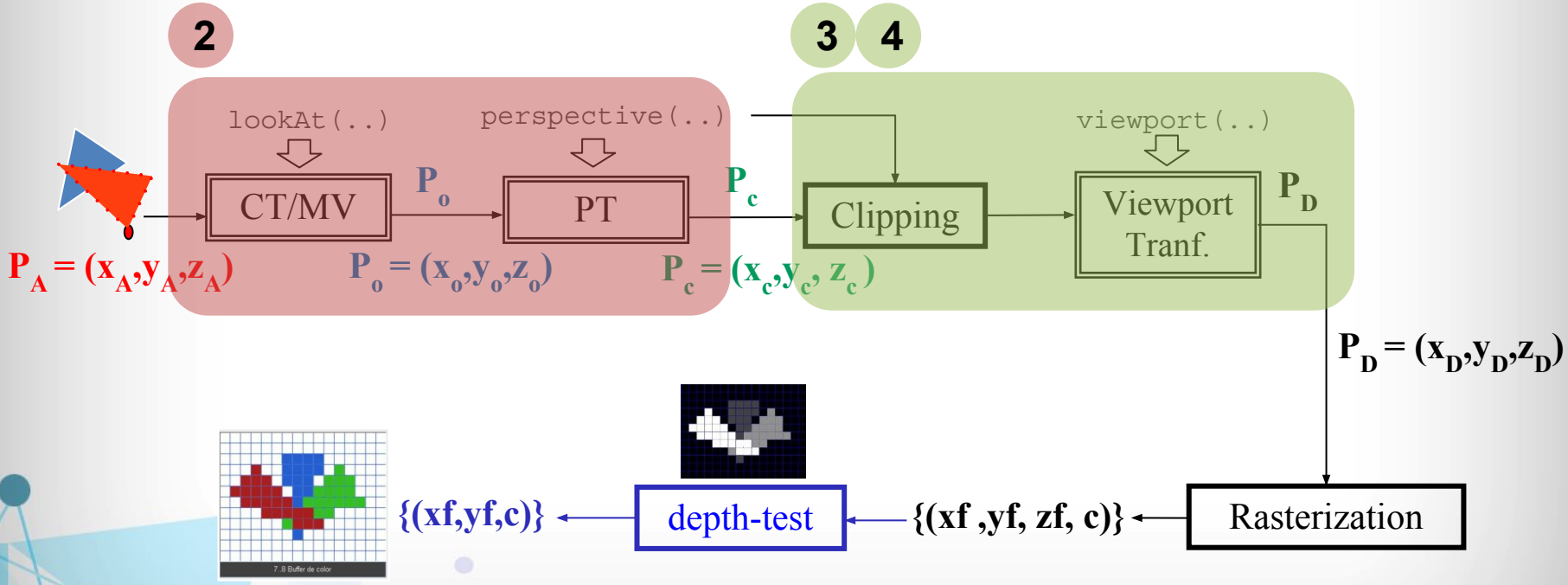
Rendering

- OpenGL/WebGL fixed pipeline



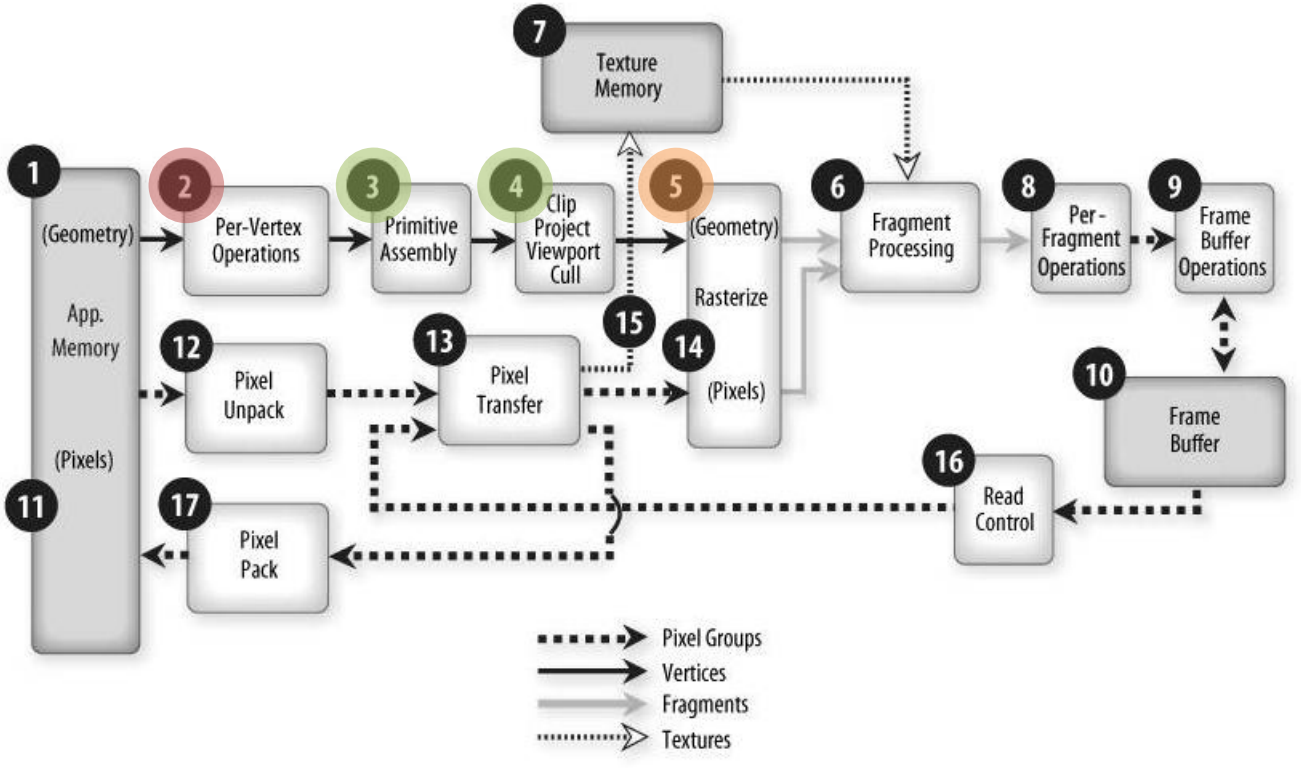
Rendering

- OpenGL/WebGL fixed pipeline



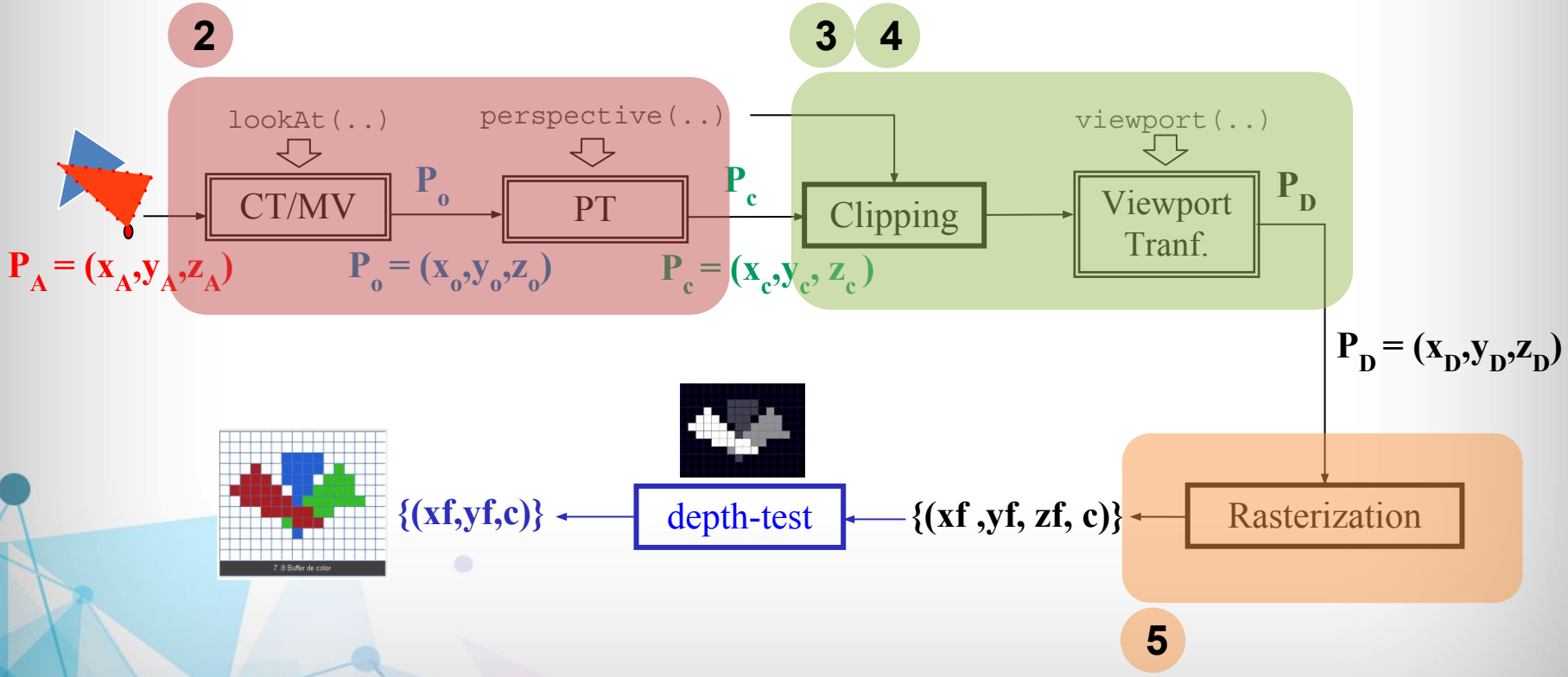
Rendering

- OpenGL/WebGL fixed pipeline



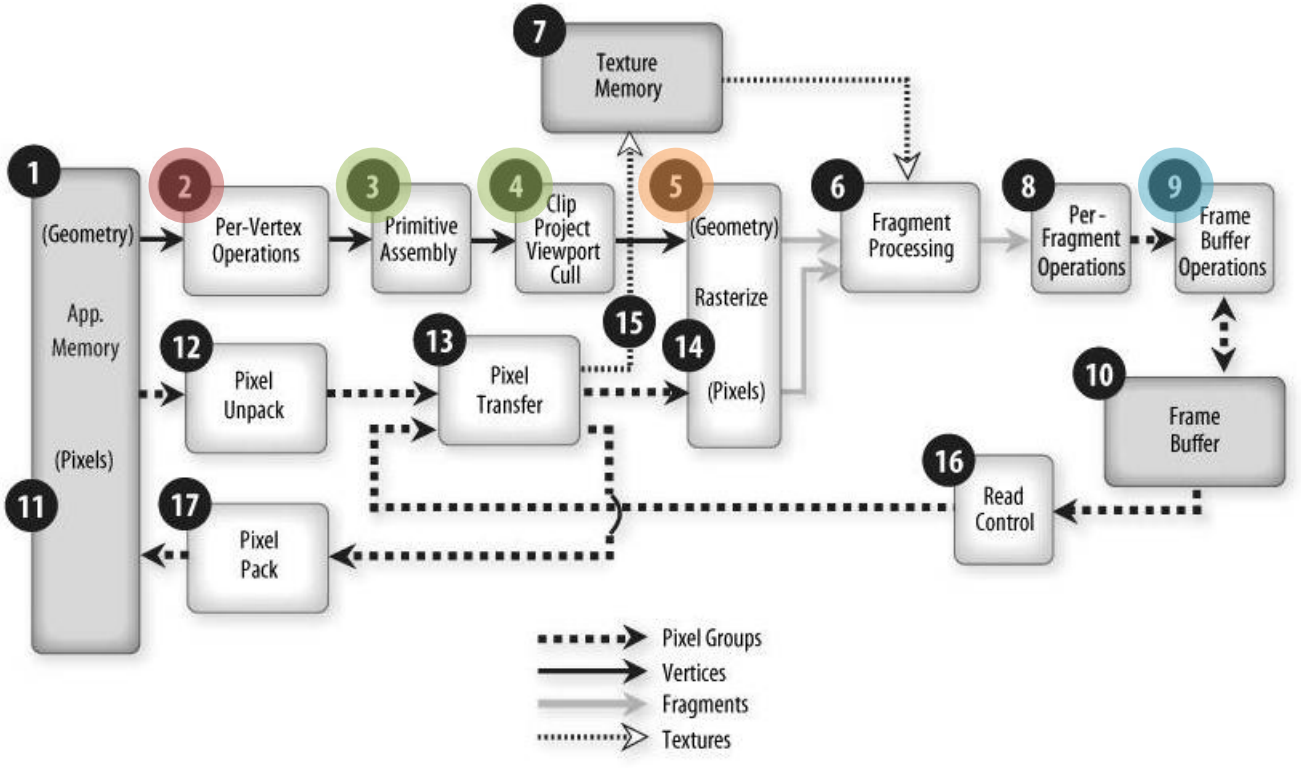
Rendering

- OpenGL/WebGL fixed pipeline



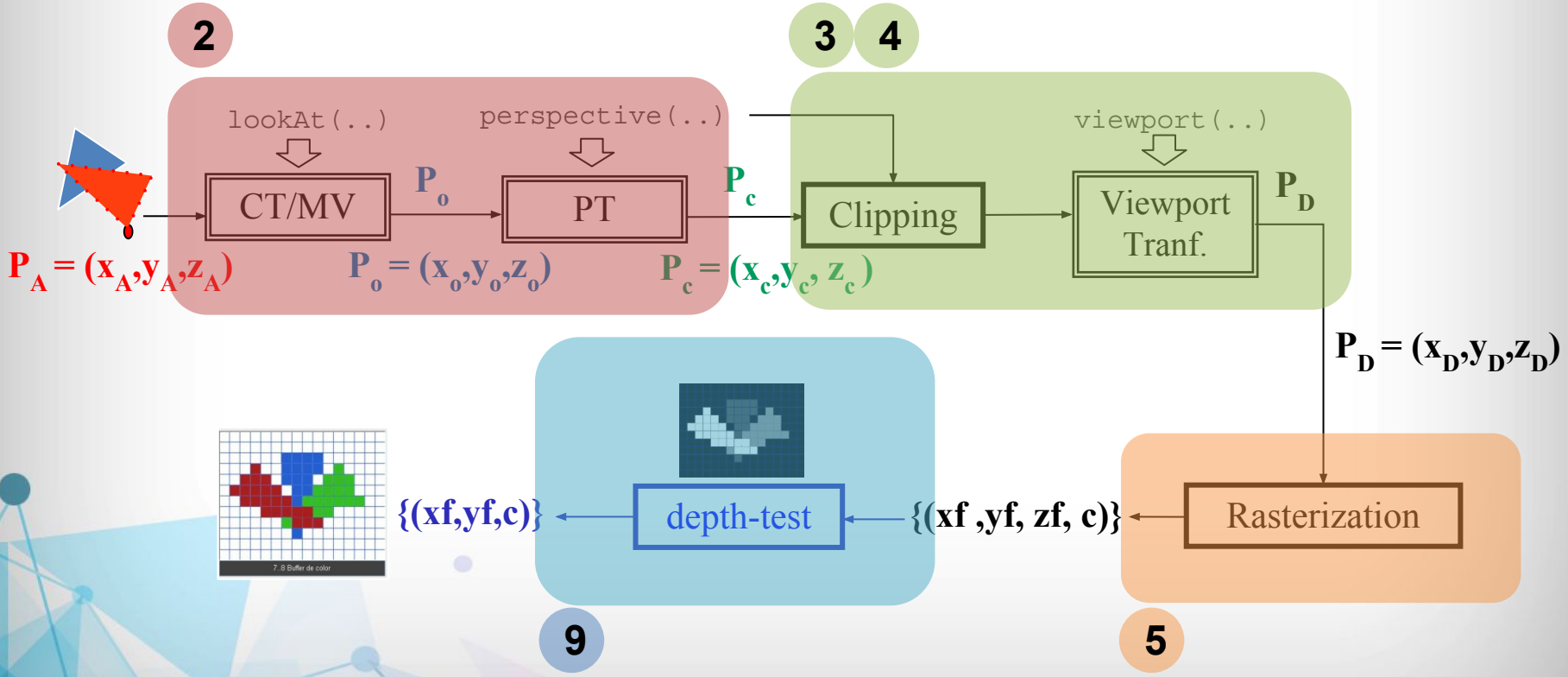
Rendering

- OpenGL/WebGL fixed pipeline



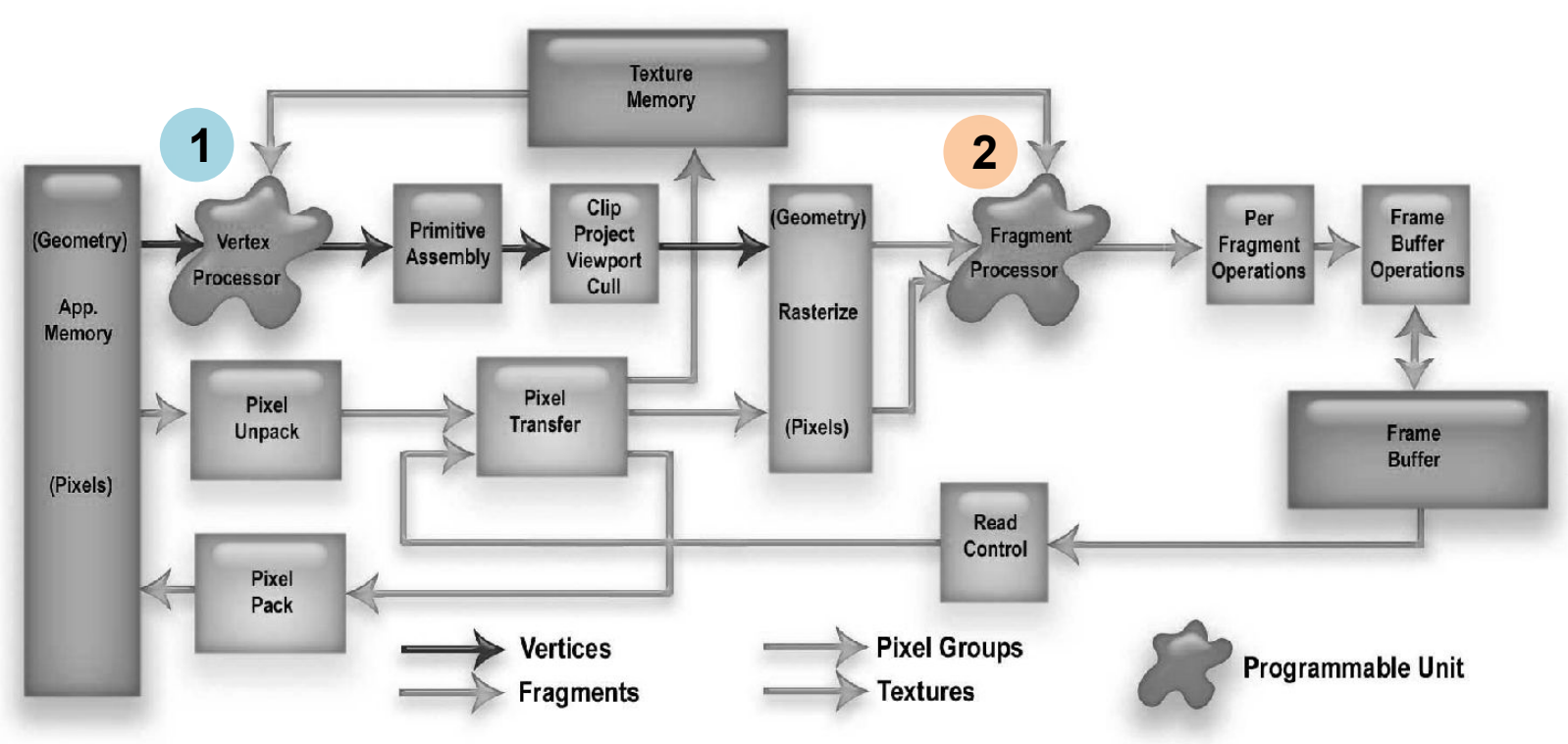
Rendering

- OpenGL/WebGL fixed pipeline



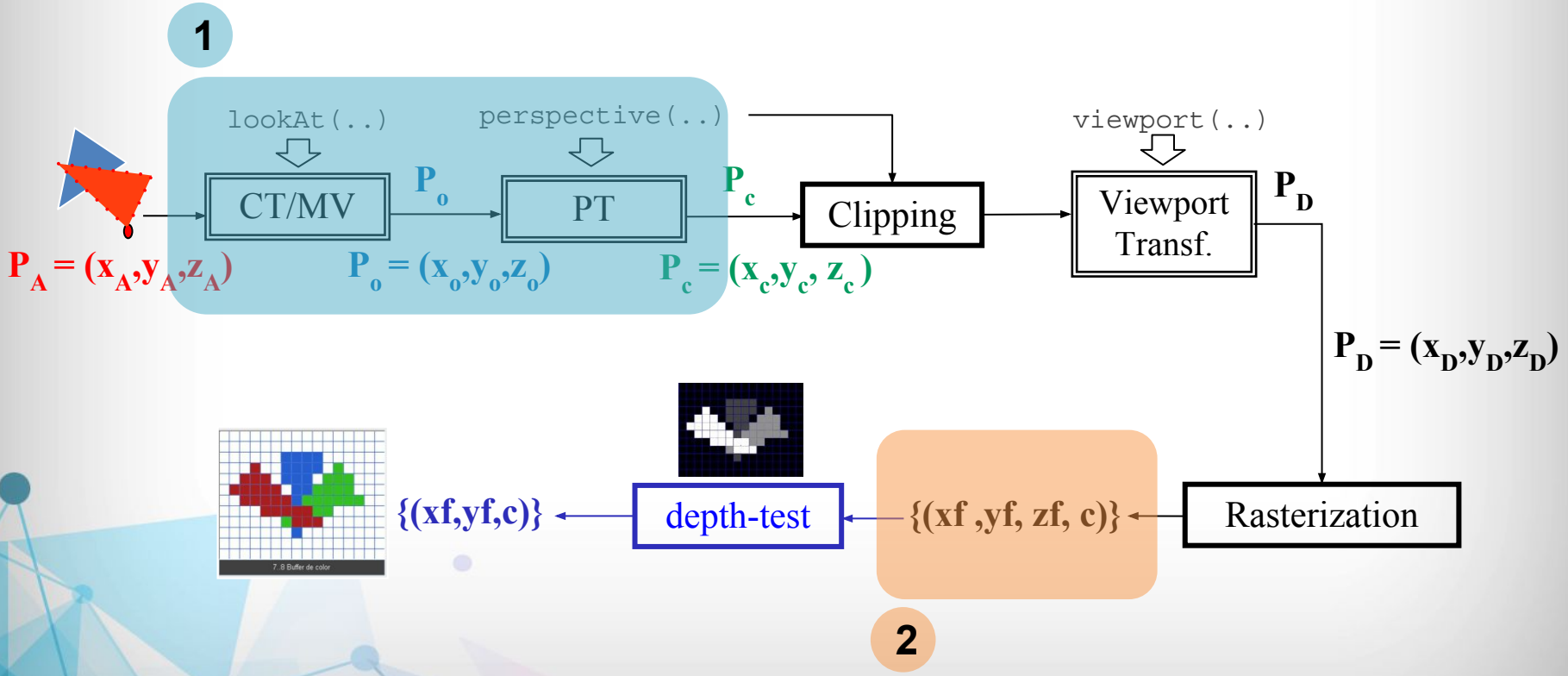
Rendering

- OpenGL/WebGL programmable pipeline



Rendering

- OpenGL/WebGL fixed pipeline



Questions?

www.citm.upc.edu