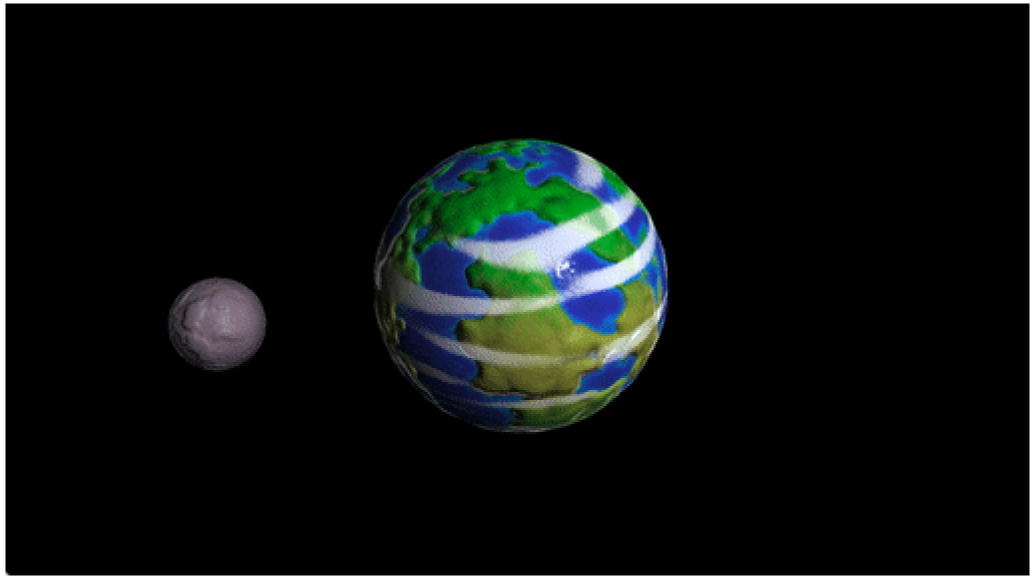
Transformations & Rasterization



Some Slides/Images adapted from Marschner and Shirley and David Levin

Today: Transformations and Shaders

Transforms and Shaders

Today:

Reminder – Rasterization

Introduction to the Graphics Pipeline

Transformations

Wednesday:

Normal and Bump Mapping

Perlin Noise

Midterm

Announcements

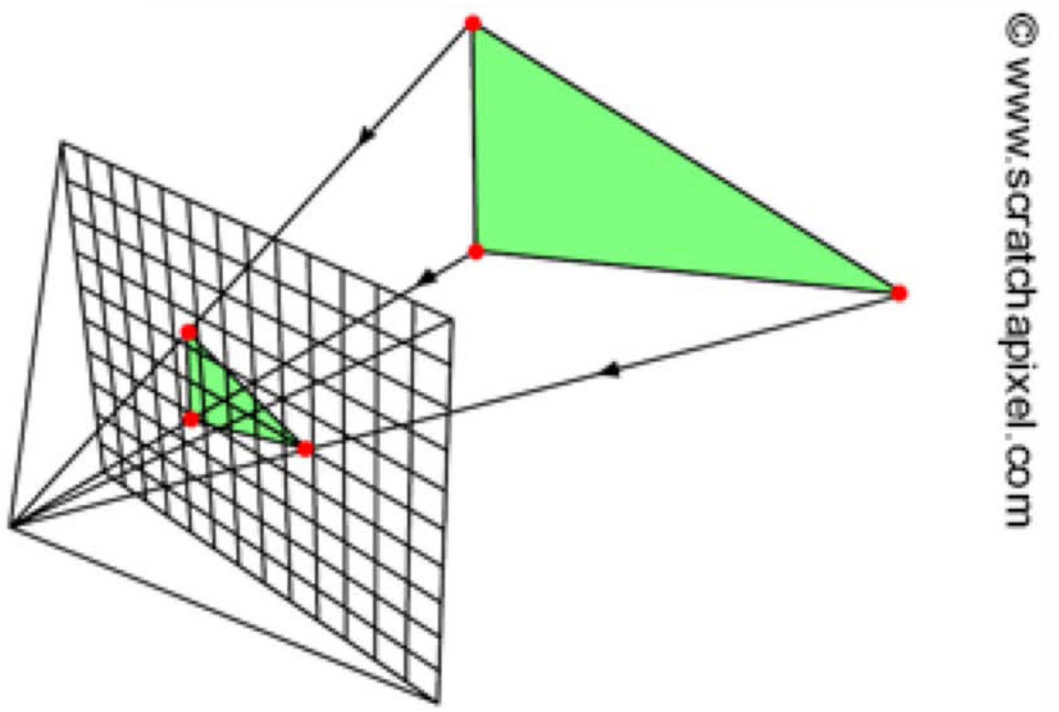
Midterm marks out Wednesday

A3 marks out by 20 July (drop date)

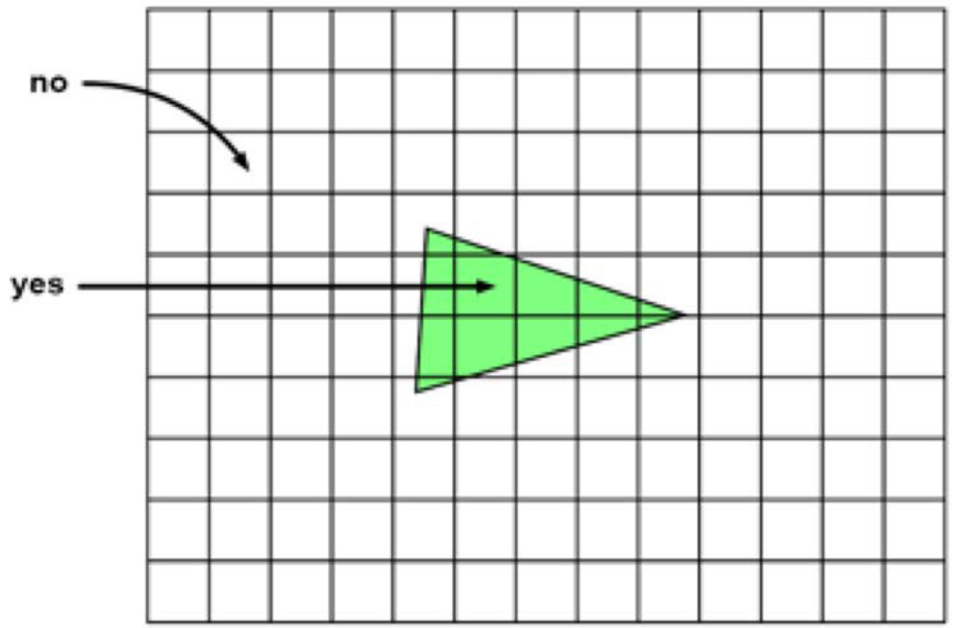
A6 due 23 July – please try to get running asap

Any Questions ?

Rasterization



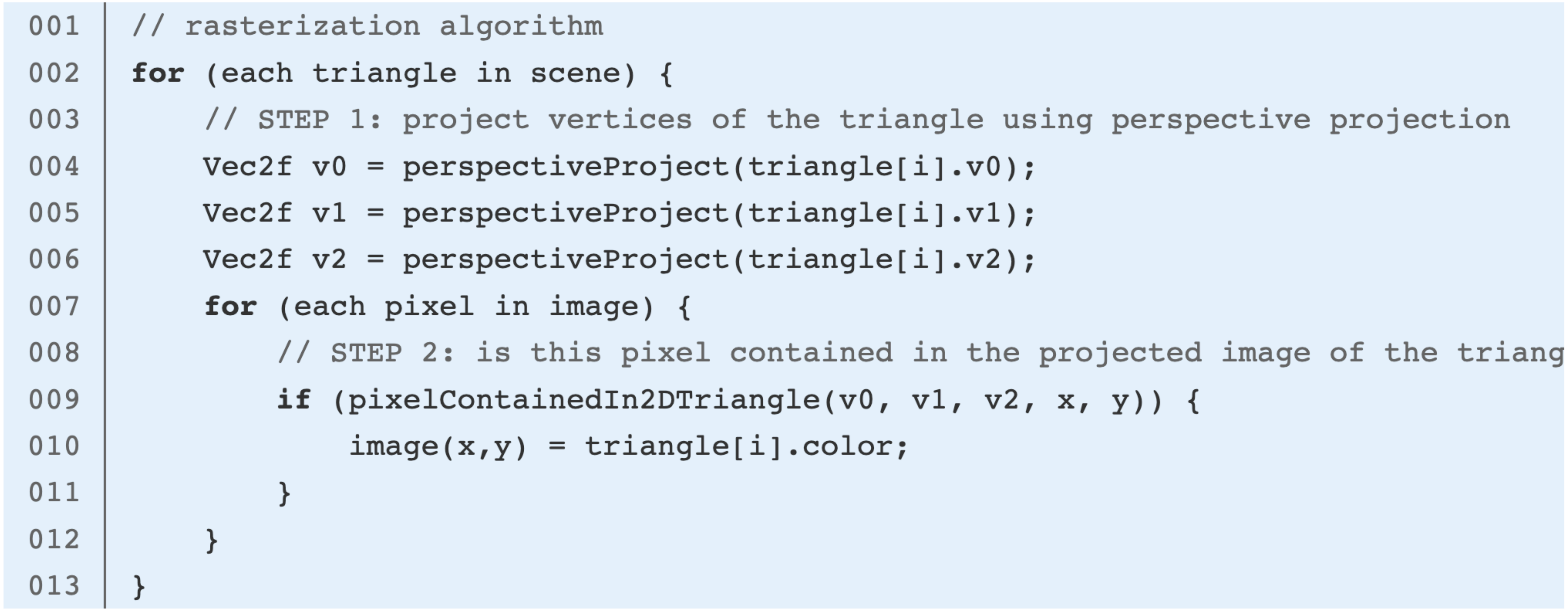
1. Project Vertices to Image Plane



2. Turn on pixels inside triangle

https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

Rasterization

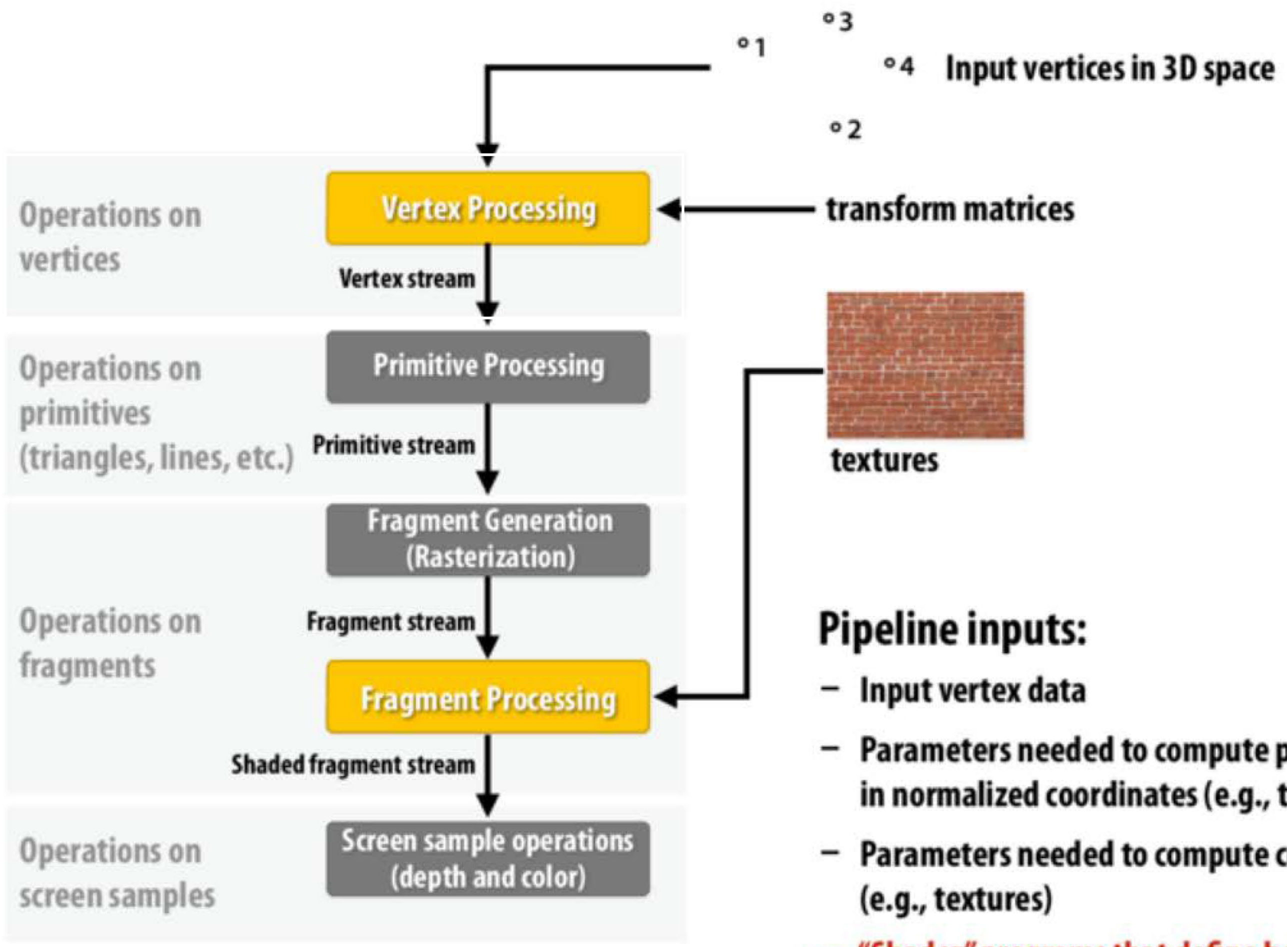


```
// rasterization algorithm
for (each triangle in scene) {
    // STEP 1: project vertices of the triangle using perspective projection
    Vec2f v0 = perspectiveProject(triangle[i].v0);
    Vec2f v1 = perspectiveProject(triangle[i].v1);
    Vec2f v2 = perspectiveProject(triangle[i].v2);
    for (each pixel in image) {
        // STEP 2: is this pixel contained in the projected image of the triang
        if (pixelContainedIn2DTriangle(v0, v1, v2, x, y)) {
            image(x,y) = triangle[i].color;
        }
    }
}
```

https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

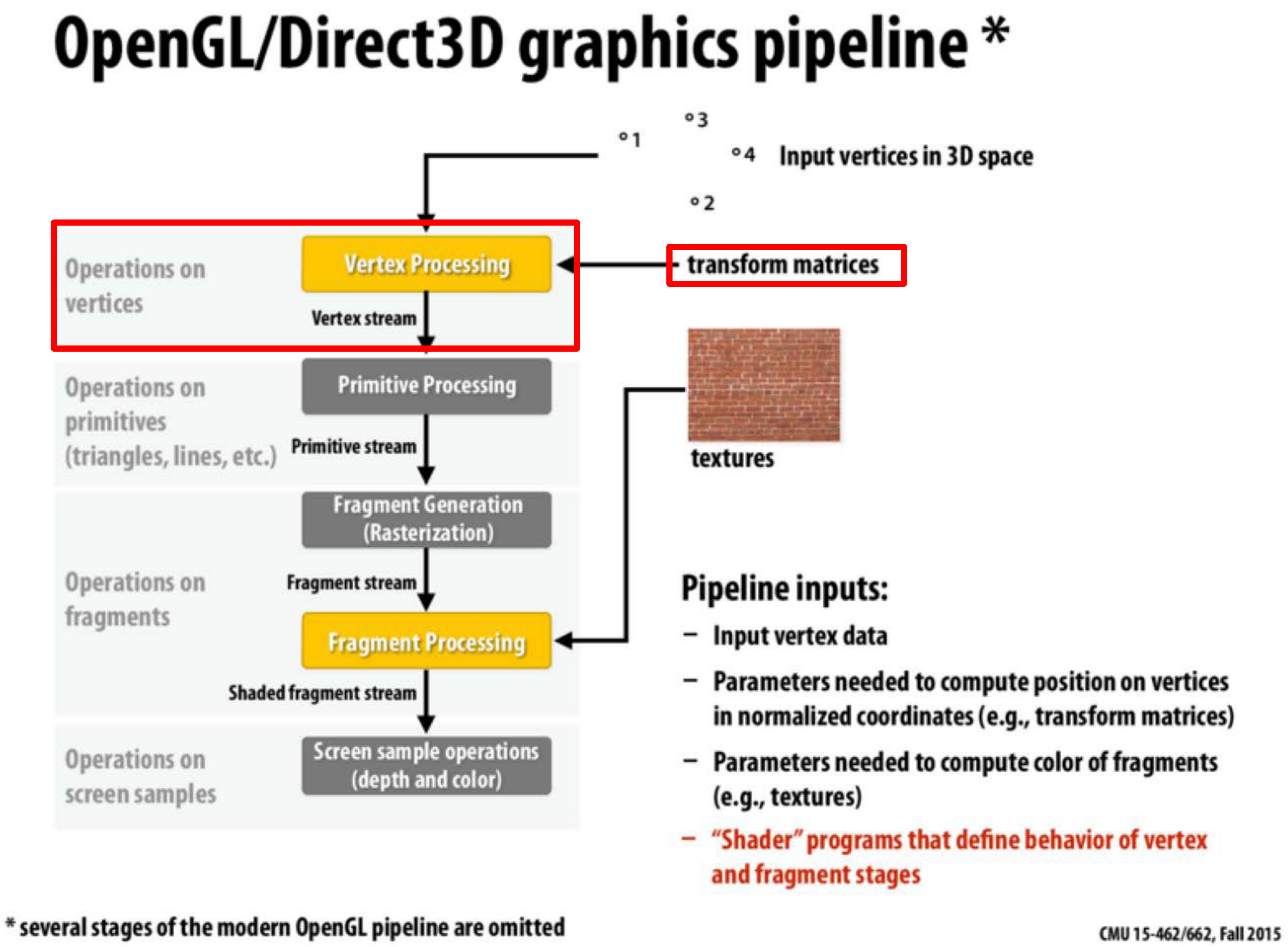


Pipeline inputs:

- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

Modern Graphics Pipeline



OpenGL/Direct3D graphics pipeline *

Pipeline inputs:

- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

CMU 15-462/662, Fall 2015

What is a linear transformation?

A: For vectors, a linear transformation is any operation performed by a matrix

$$Ax = b$$

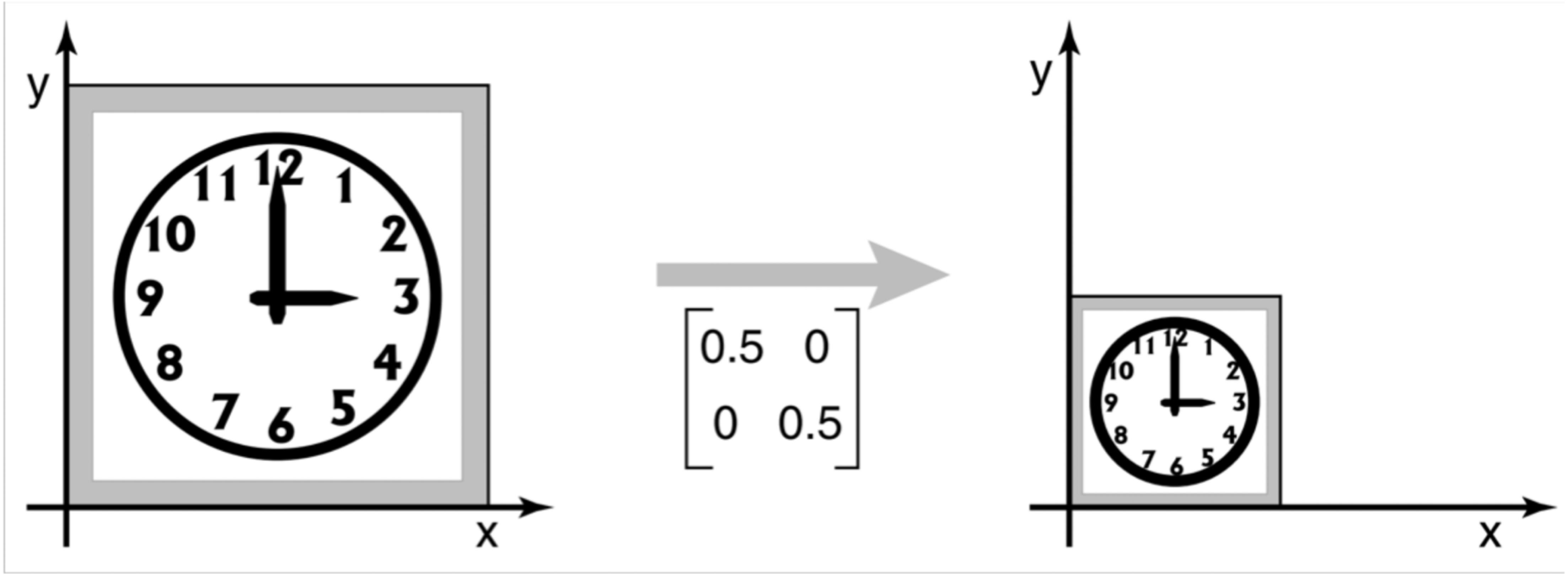
2D Linear Transformations

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

2D Linear Transformations - Scale

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

Linear transformations in 2D: Scale

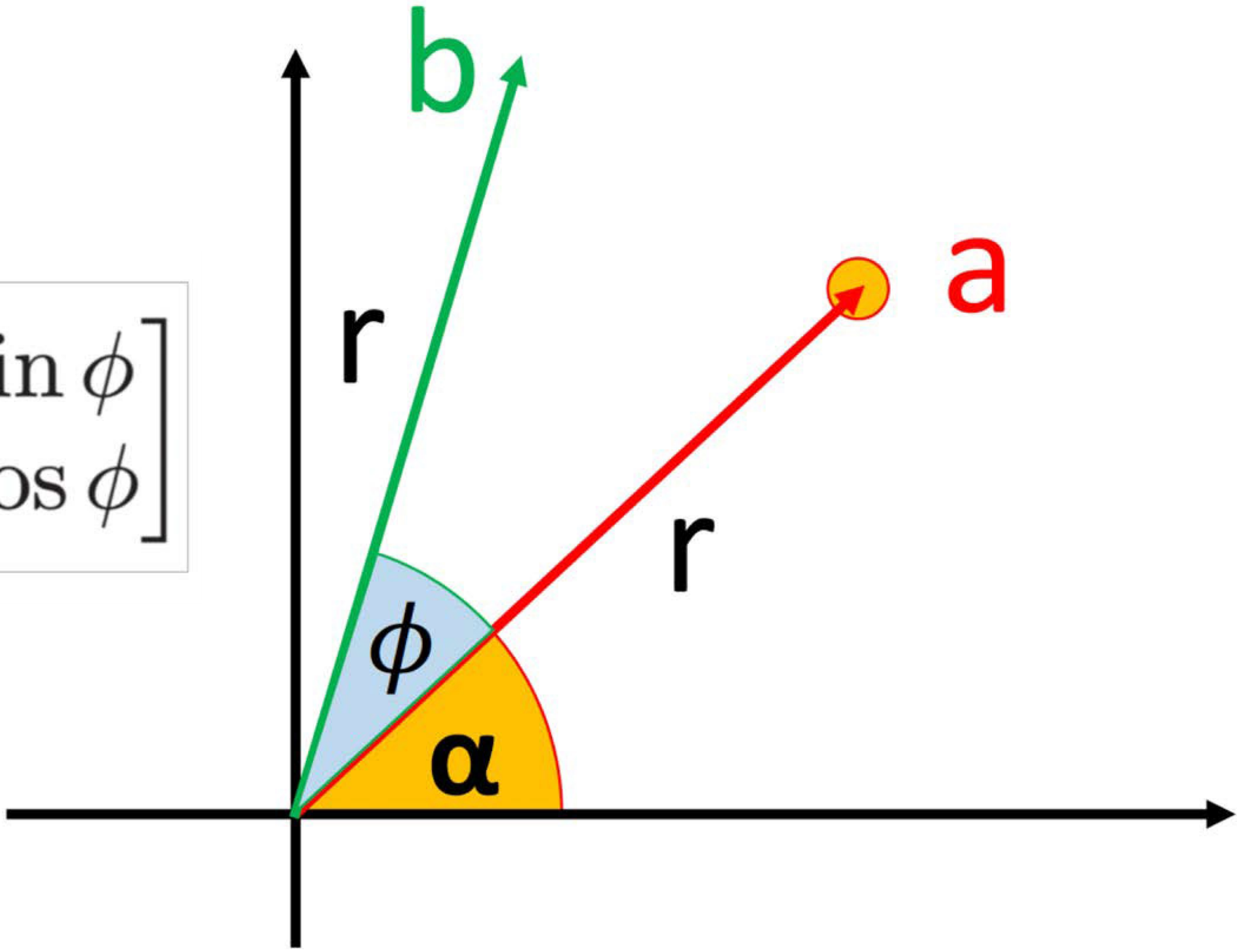


$$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

When Sx = Sy we say the scaling is uniform

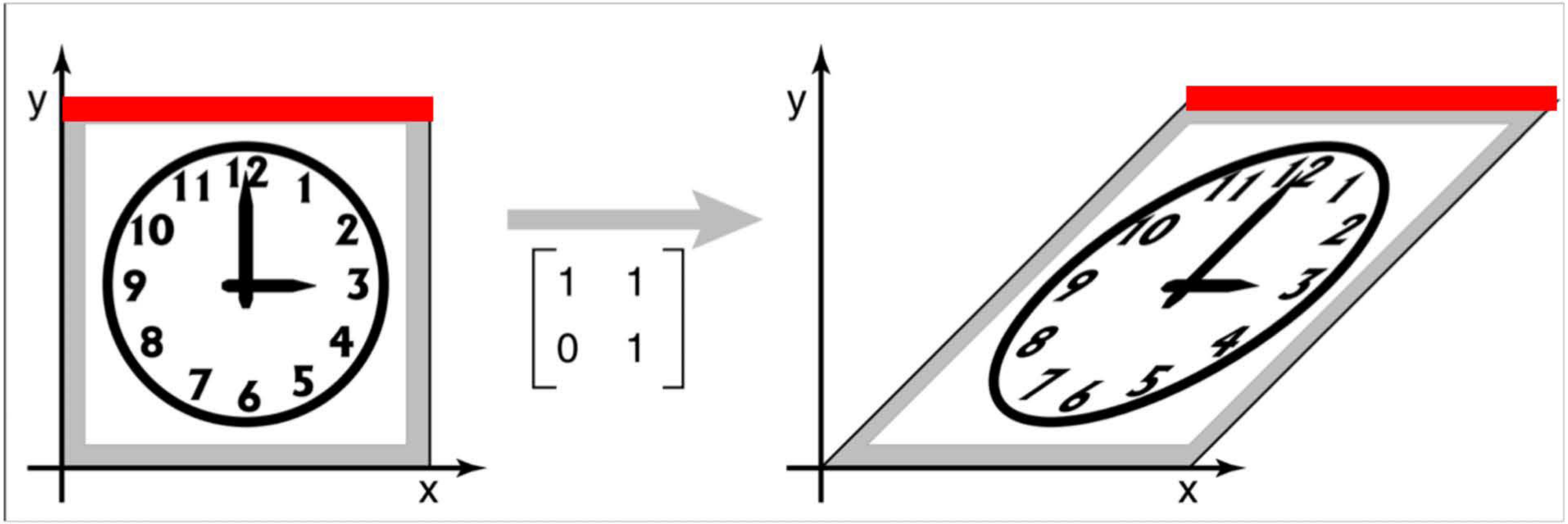
2D Linear Transformations - Rotation

$$\text{rotate}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$



2D Linear Transformations - Shear

$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + sy \\ y \end{bmatrix}$$



$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

These are always the same length

2D Linear Transformations - Translation

$$T\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

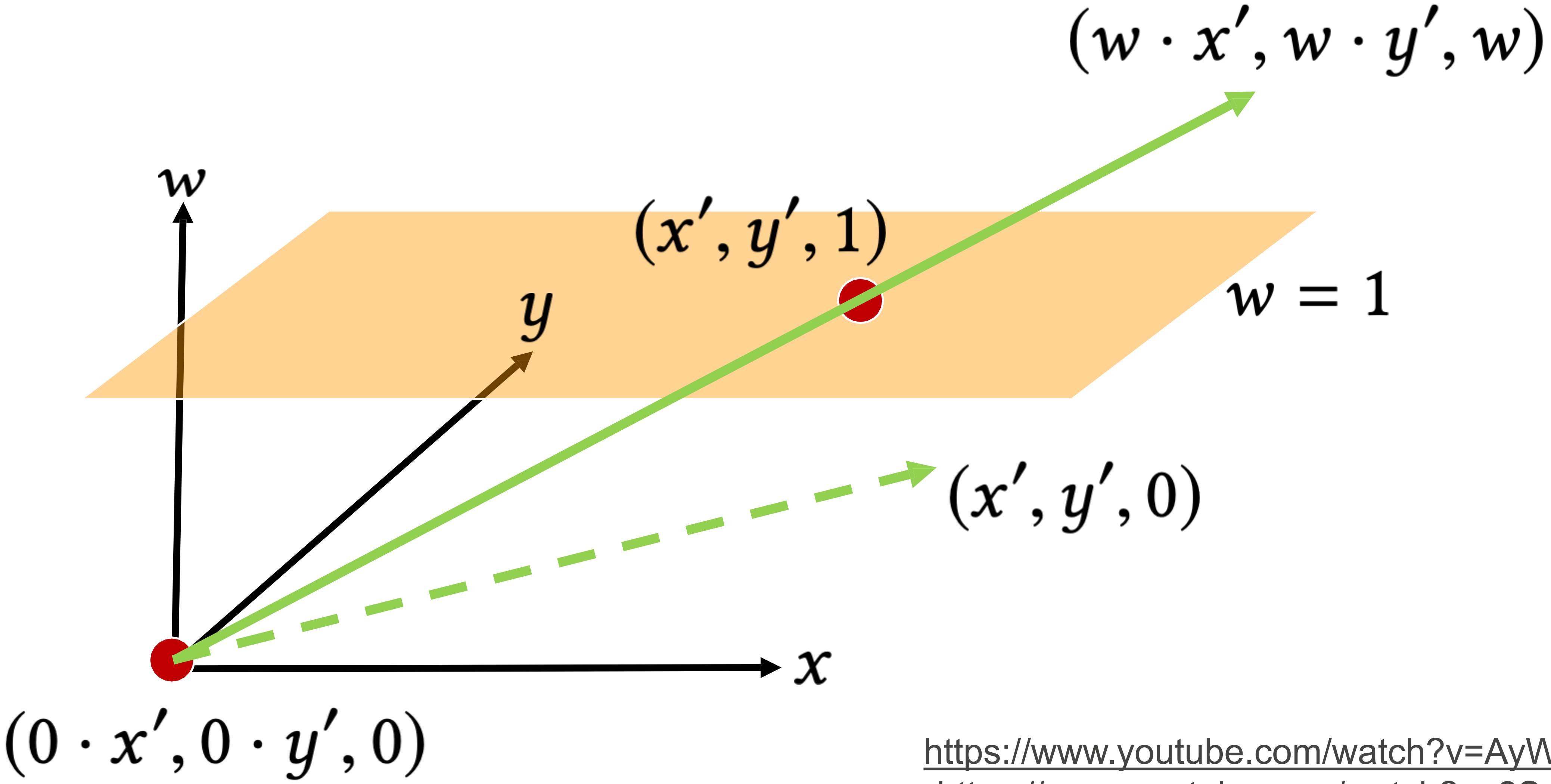
2D Affine Transformations - Translation

$$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_x \\ a_{21}x + a_{22}y + t_y \\ 1 \end{bmatrix}$$

$$Ax + t = b$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow[\text{homogeneous coordinates}]{\text{Considered as a point in 3D}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Geometric Intuition

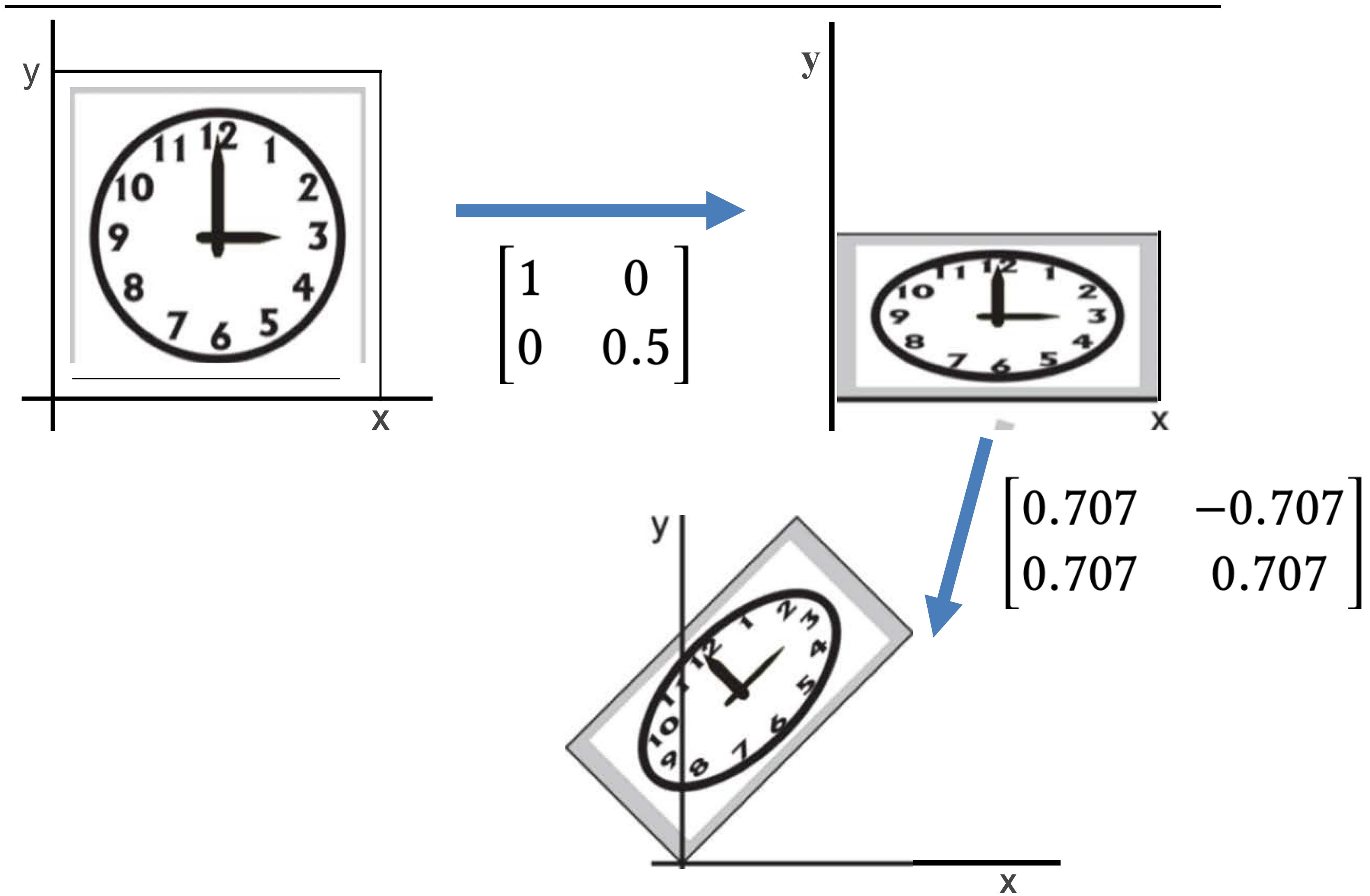


https://www.youtube.com/watch?v=AyW4Y8APjK4
https://www.youtube.com/watch?v=2Snoepcmi9U
https://www.youtube.com/watch?v=Q2uItHa7GFQ

What about vectors?

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow[\text{homogeneous coordinates}]{\text{Considered as a } \underline{\textbf{\textit{vector}}} \text{ in 3D}} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Composing transformations



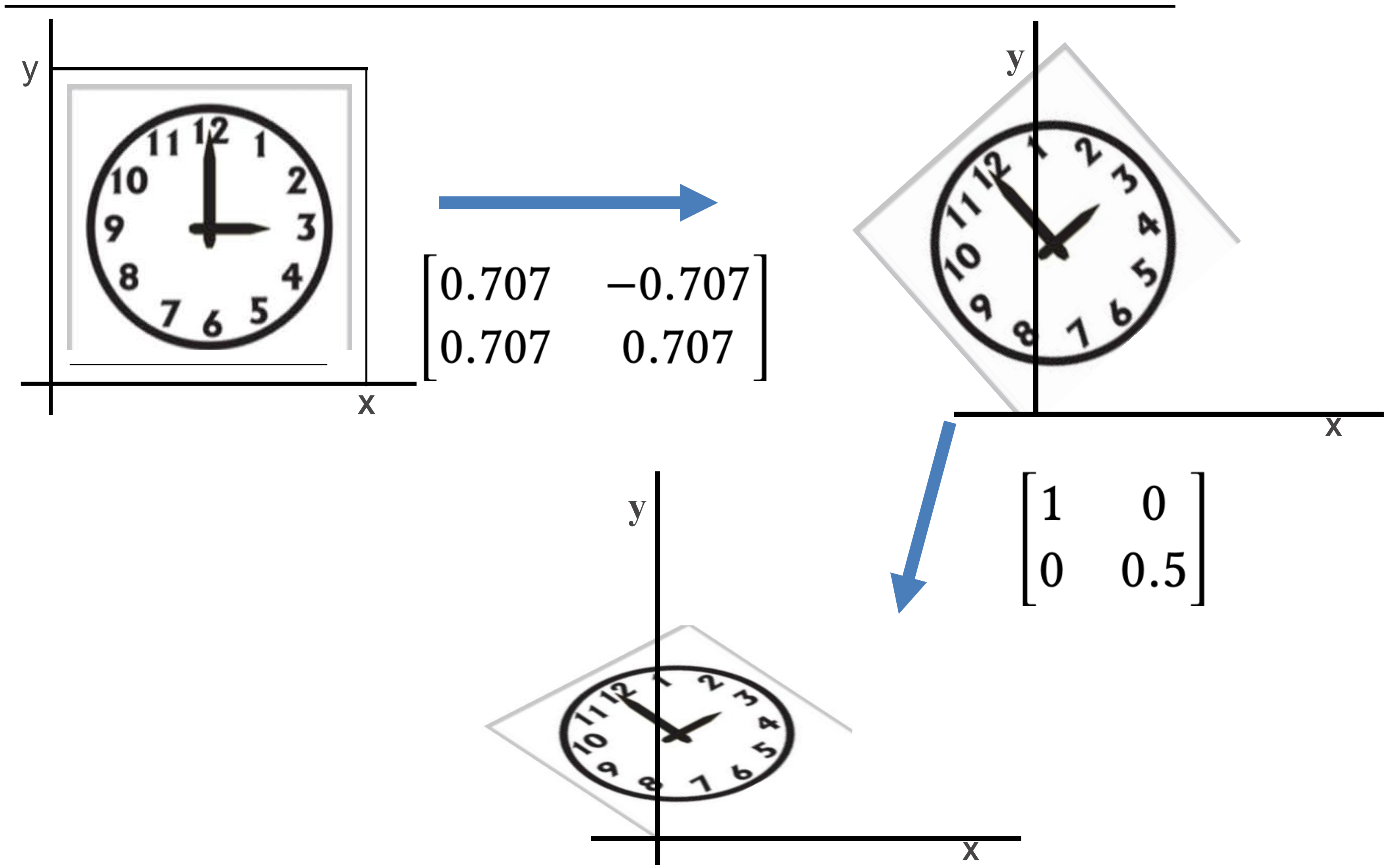
$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}$$

Composing transformations

$$\underbrace{\begin{bmatrix} .707 & -.707 \\ .707 & .707 \end{bmatrix}}_{\text{2nd transformation}} \underbrace{\begin{bmatrix} 1.0 & 0 \\ 0 & 0.5 \end{bmatrix}}_{\text{1st transformation}} = \begin{bmatrix} .707 & -.353 \\ .707 & .353 \end{bmatrix}$$

Composing transformations

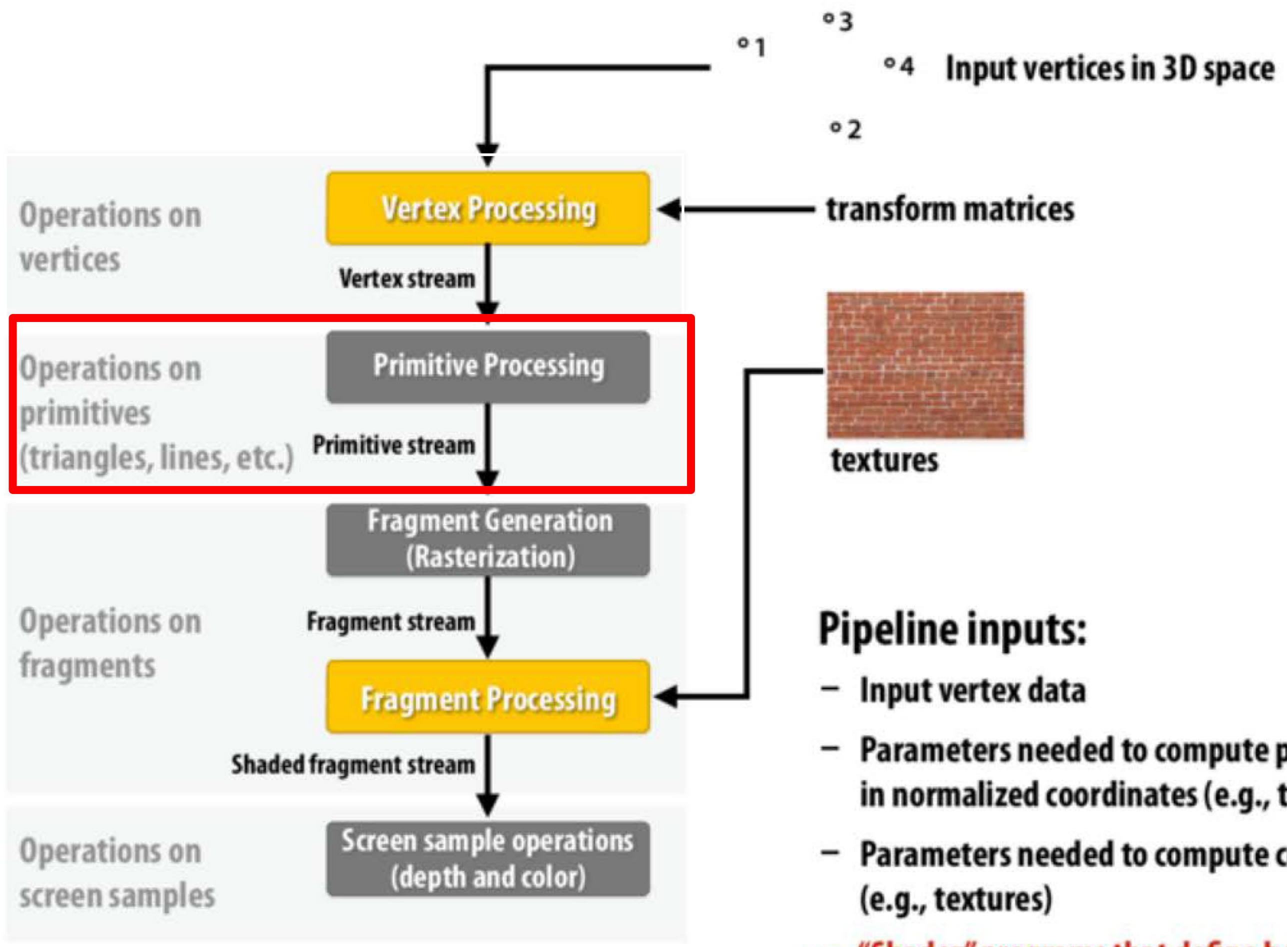


$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

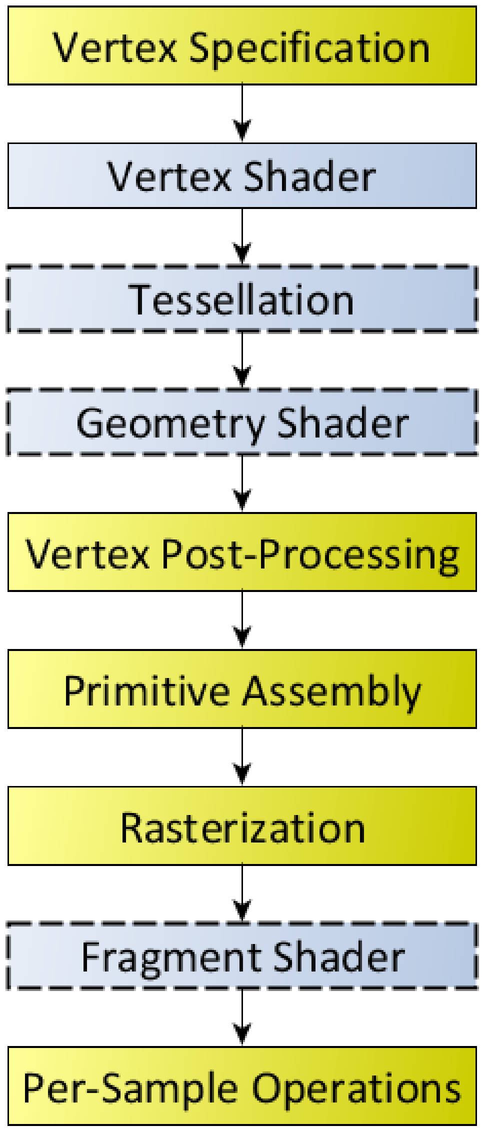


Pipeline inputs:

- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

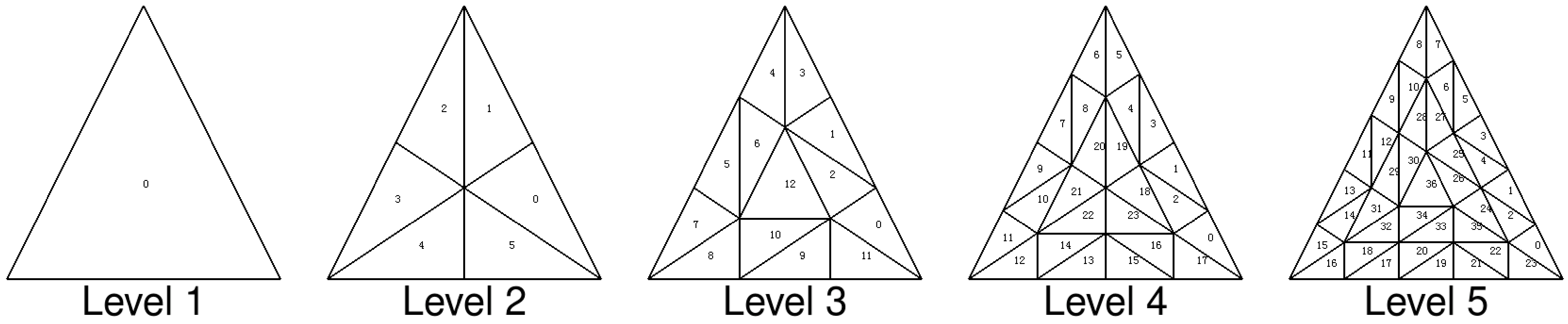
Modern Graphics Pipeline



Tessellation Control Shader
Tessellation Evaluation Shader

https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

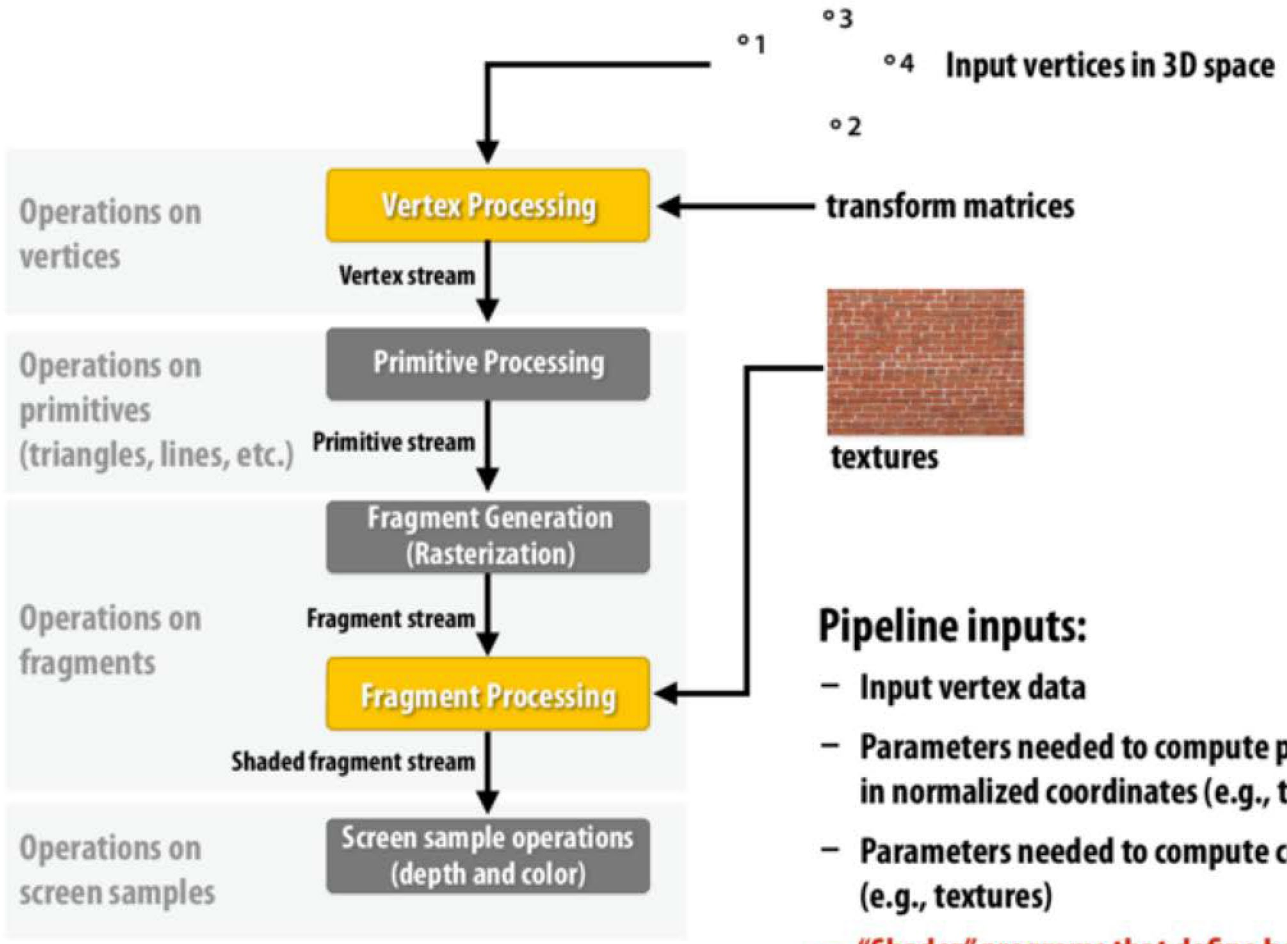
Tessellation Shader



https://erkaman.github.io/posts/tess_opt.html

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

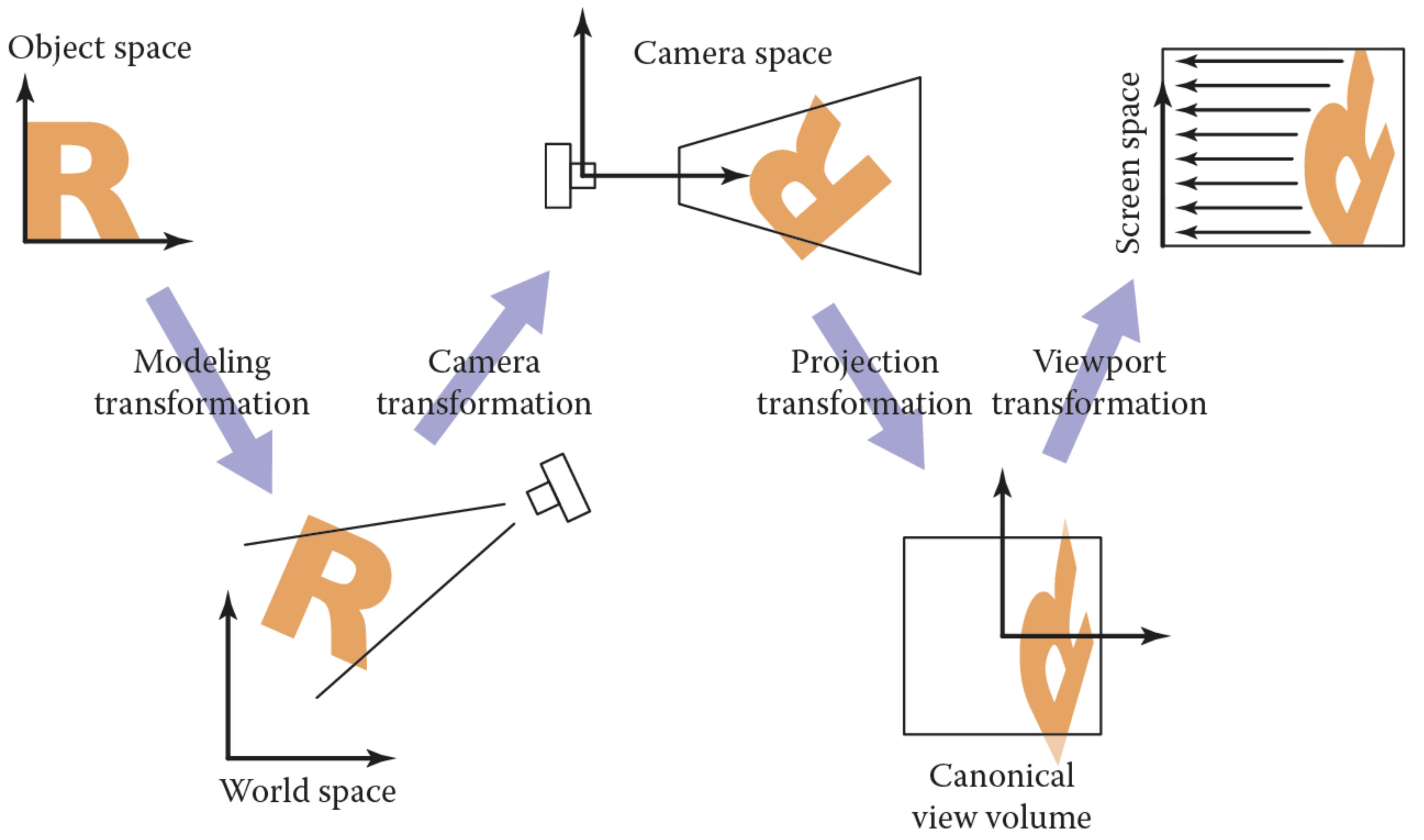


Pipeline inputs:

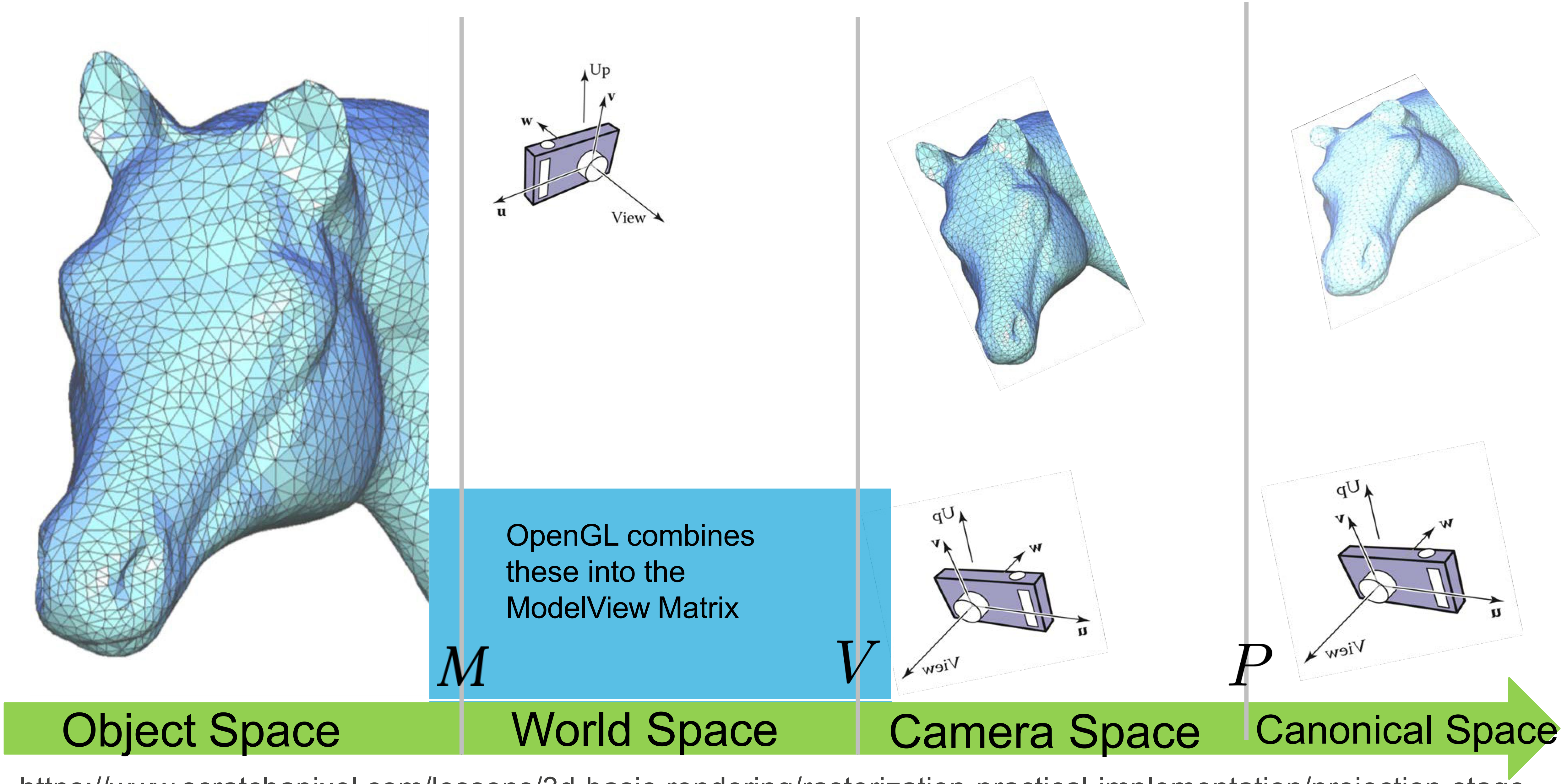
- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

Getting Things Onto The Screen



Getting Things Onto The Screen

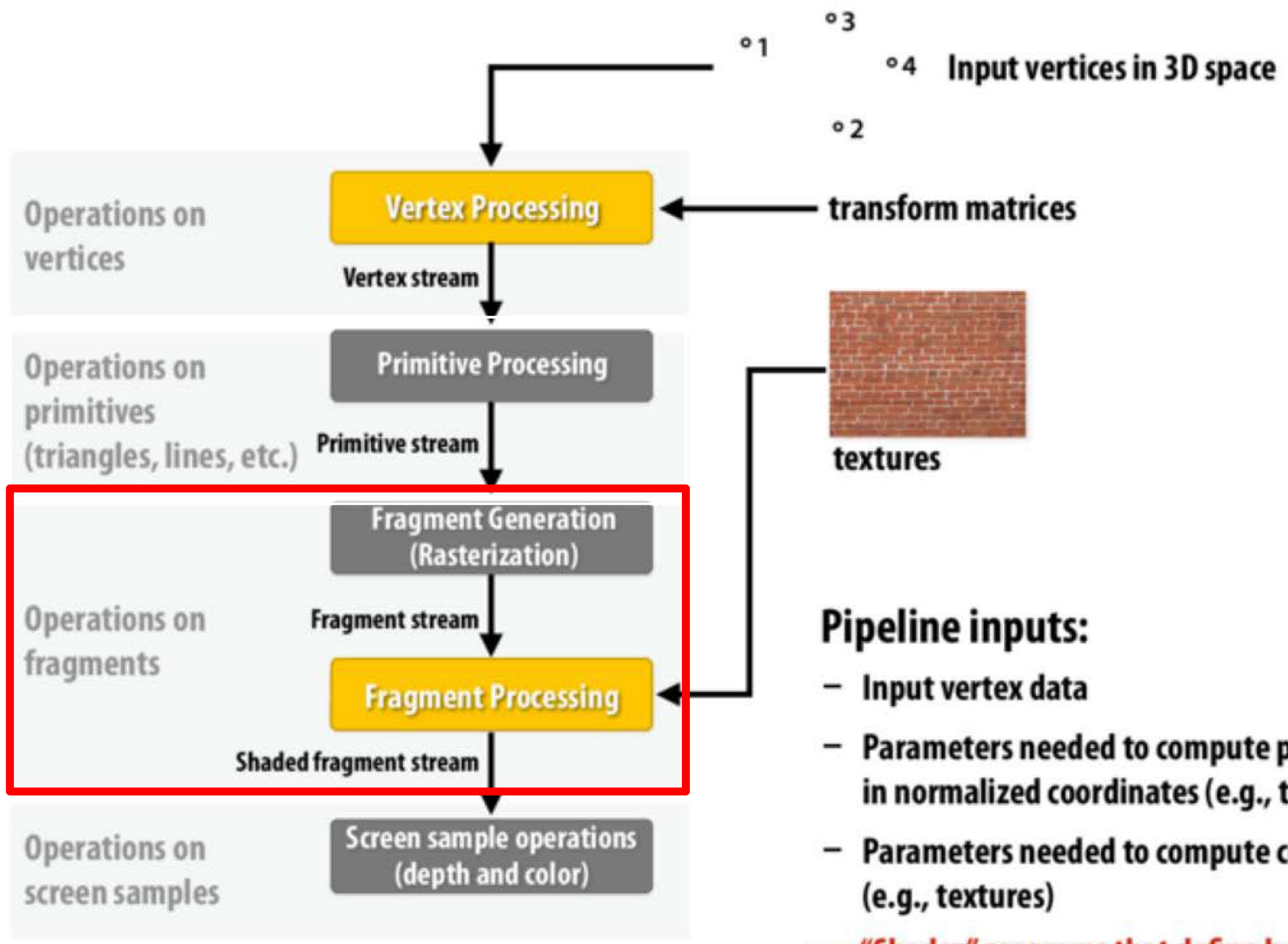


OpenGL combines these into the ModelView Matrix

https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/projection-stage

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

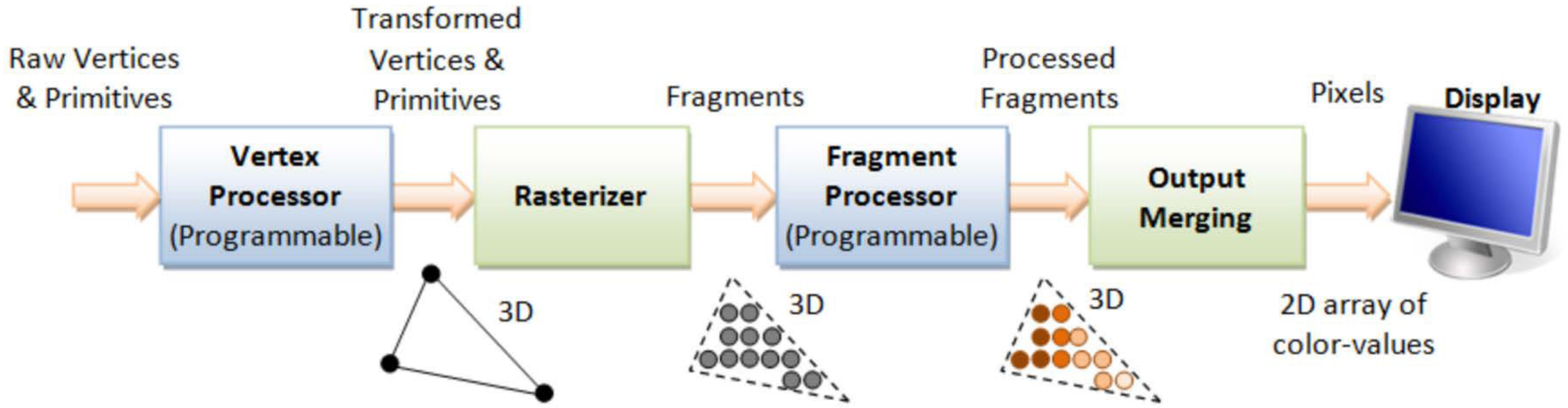


Pipeline inputs:

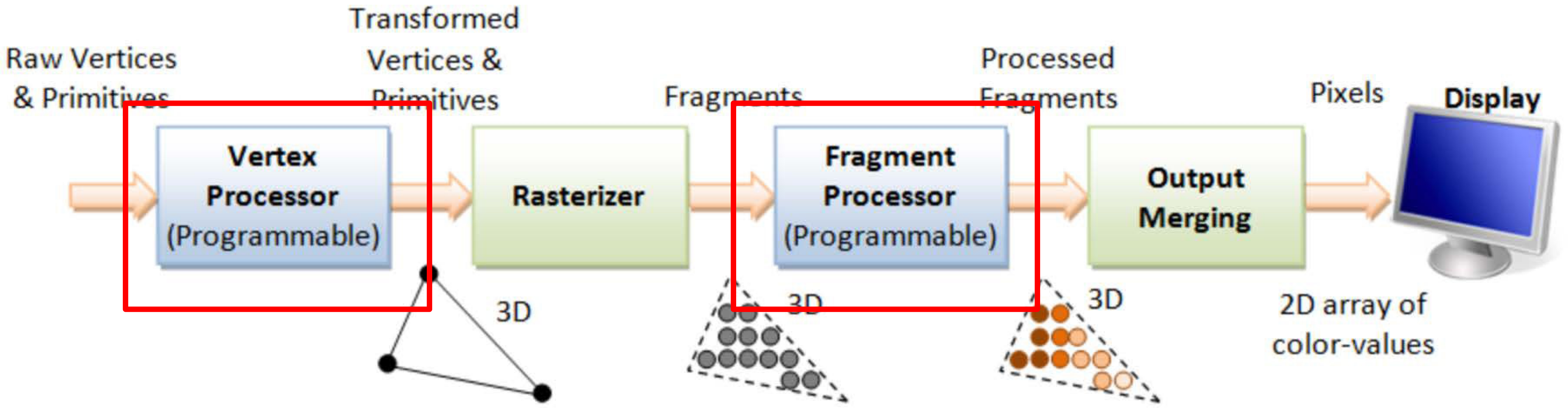
- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

Fragment Shader


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
3D
Fragments
Fragment Processor (Programmable)
3D
Processed Fragments
Output Merging
3D
Pixels
Display
2D array of color-values

Fragment Shader


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
3D
Rasterizer
Fragments
3D
Fragment Processor (Programmable)
Processed Fragments
3D
Output Merging
Pixels
2D array of color-values
Display

All Done For Today