CSC418/2504 Computer Graphics

Some Slides/Images adapted from Marschner and Shirley

Today: Transformations and Shaders

Transforms and Shaders

Reminder – Rasterization

Introduction to the Graphics Pipeline

Transformations

Normal and Bump Mapping

Perlin Noise

Announcements

A2 “… looking very good” – Darren

Assignment 5 and 6 due Feburary 28th

Assignment 6 out soon

Midterm Monday February 24th (both tutorial rooms)

Covers everything up to and including meshes

Last Year: 6 Questions, 50 Marks (1 mark per minute).

Some Example Questions:

Consider a ray emanating from a 3D position $\mathbf{e} \in \mathbb{R}^3$ in the direction defined by a 3D vector $\mathbf{d} \in \mathbb{R}^3$.

i– [1 MARK] Write a *parametric* equation that allows plugging in any non-negative parameter t to trace points along the ray:

$\mathbf{r}(t) =$

ii– [2 MARKS] When using per-vertex normals, each vertex has normal.

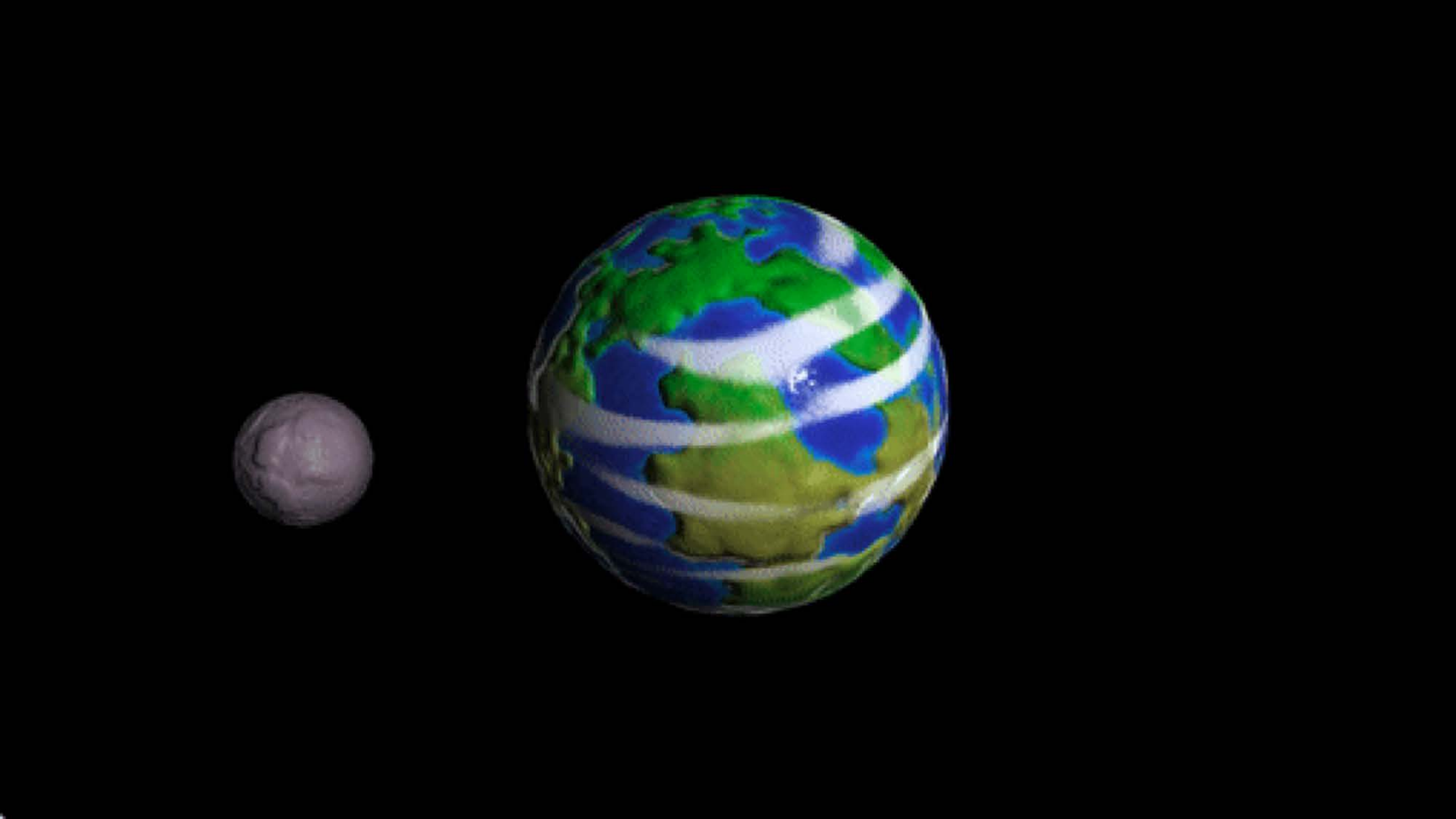
When using per-corner normals, each _____ has _____ normal(s).

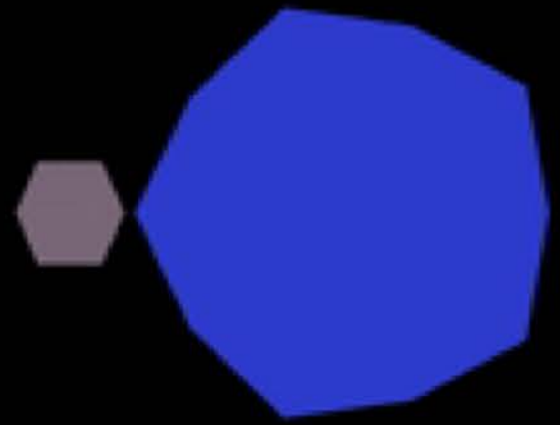
Consider a point $\mathbf{p} \in \mathbb{R}^3$ on a surface with unit normal $\hat{\mathbf{n}} \in \mathbb{R}^3$, viewed from ray emanating from the point $\mathbf{e} \in \mathbb{R}^3$ and point light located at $\boldsymbol{\ell} \in \mathbb{R}^3$. The following formula calculates the color of the object using the Blinn-Phong model:

$$\mathbf{c} = \mathbf{k}_a I_a + \mathbf{k}_d I_d \max(\hat{\mathbf{n}} \cdot L, 0) + \mathbf{k}_s I_s \max\left(\hat{\mathbf{n}} \cdot \frac{L+V}{\|L+V\|}, 0\right)^p, \text{ where } V = \frac{\mathbf{e}-\mathbf{p}}{\|\mathbf{e}-\mathbf{p}\|} \text{ and } L = \frac{\boldsymbol{\ell}-\mathbf{p}}{\|\boldsymbol{\ell}-\mathbf{p}\|}$$

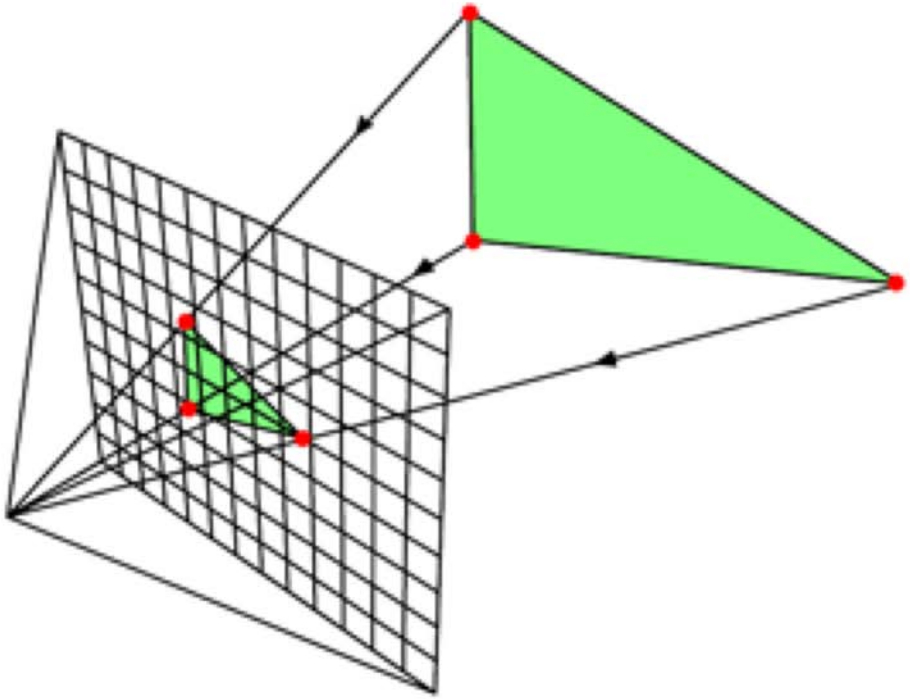
ii– Non-physical diffuse lighting [3 MARKS] Now, imagine we want to create an artistic effect where the diffuse lighting is proportional to how far away the light is from the object. When the object is close, there is little diffuse color, and when the object is far, there is a large diffuse color. Give a modified Blinn-Phong formula to model this pheonomenon.

Any Questions ?

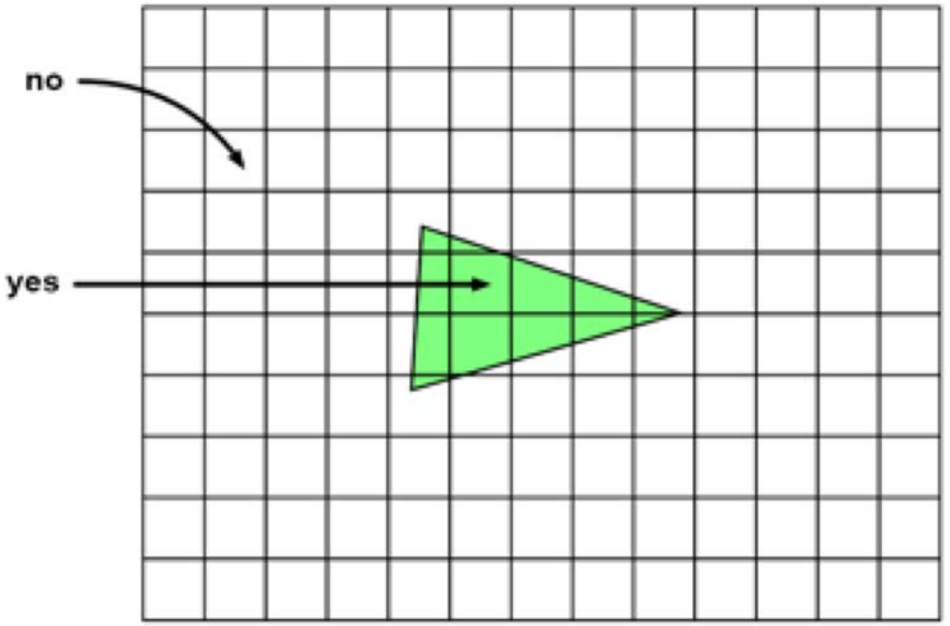




Rasterization



1. Project Vertices to Image Plane

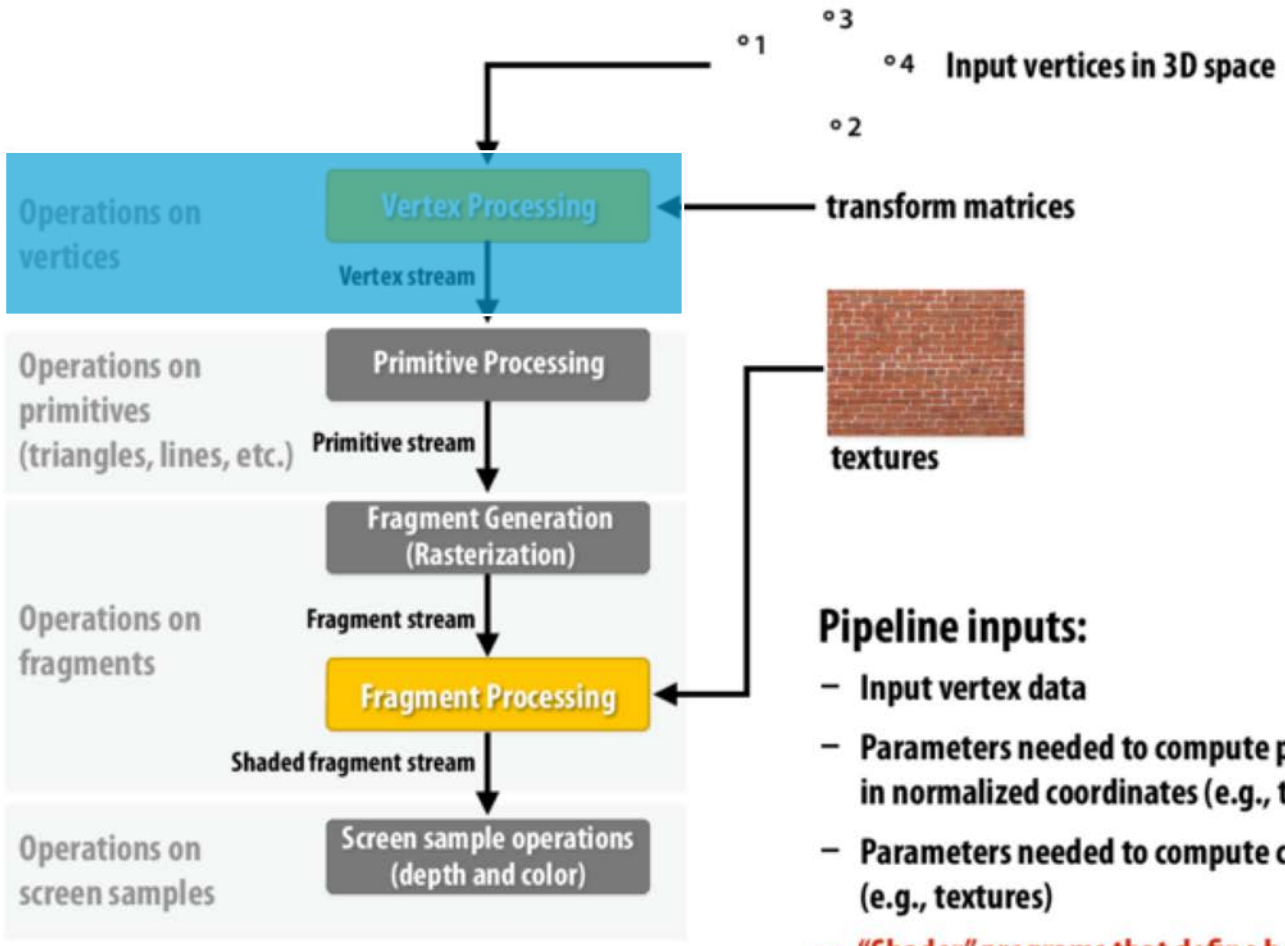


2. Turn on pixels inside triangle

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

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

What is a linear transformation?

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

$$Ax = b$$

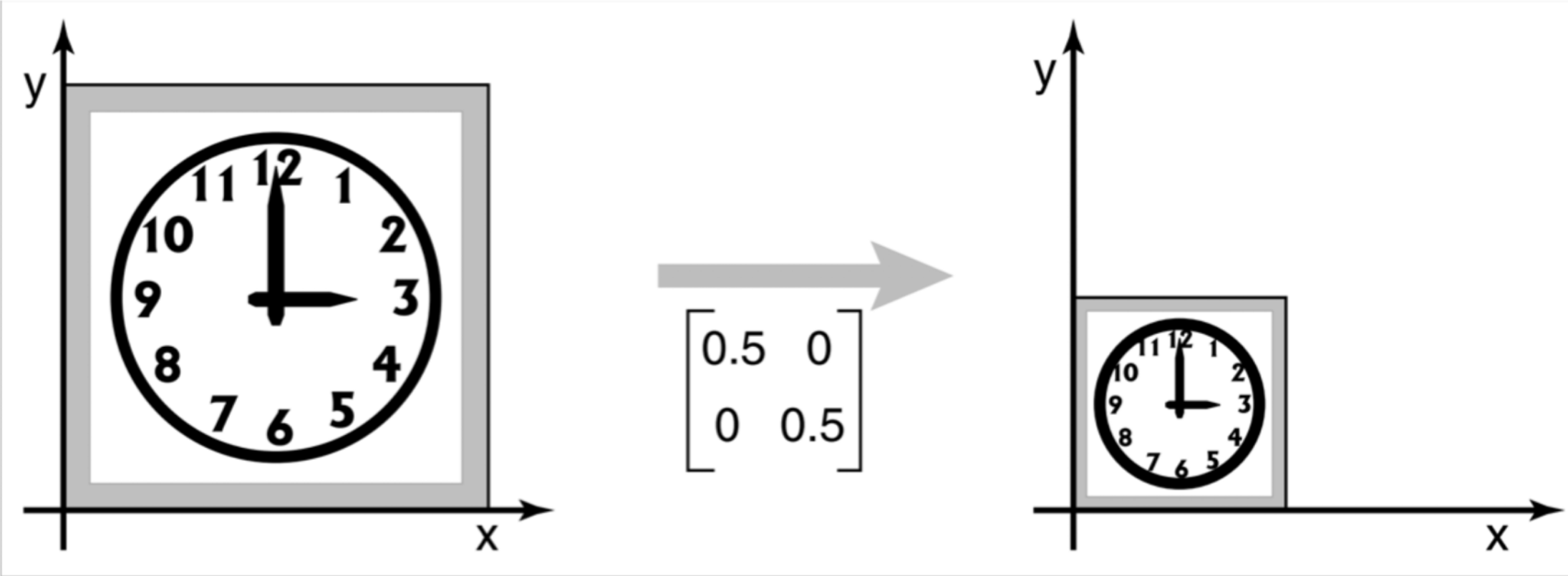
2D Linear Transformations

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

2D Linear Transformations - Scale

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

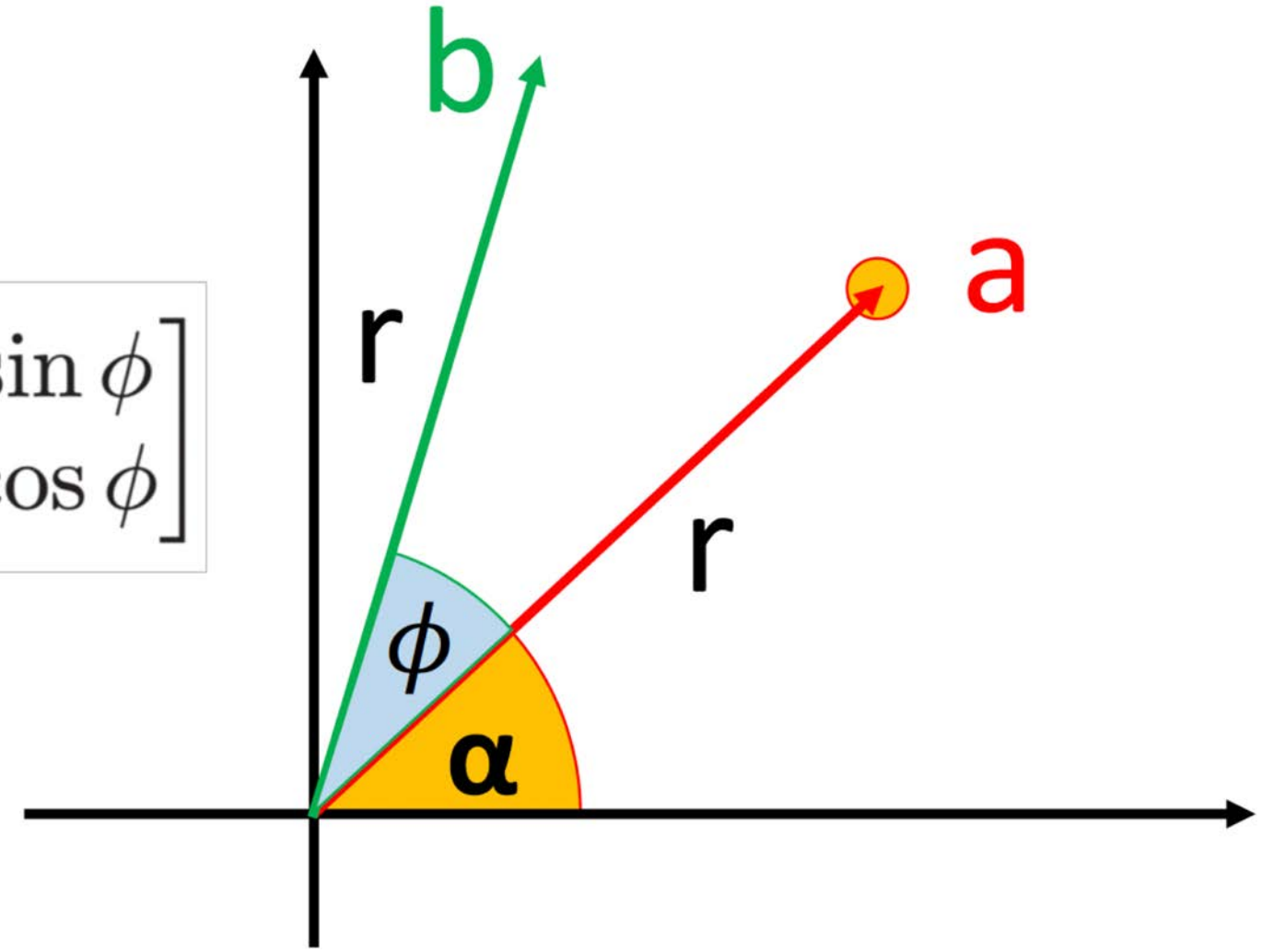
Linear transformations in 2D: Scale



When Sx = Sy we say the scaling is uniform

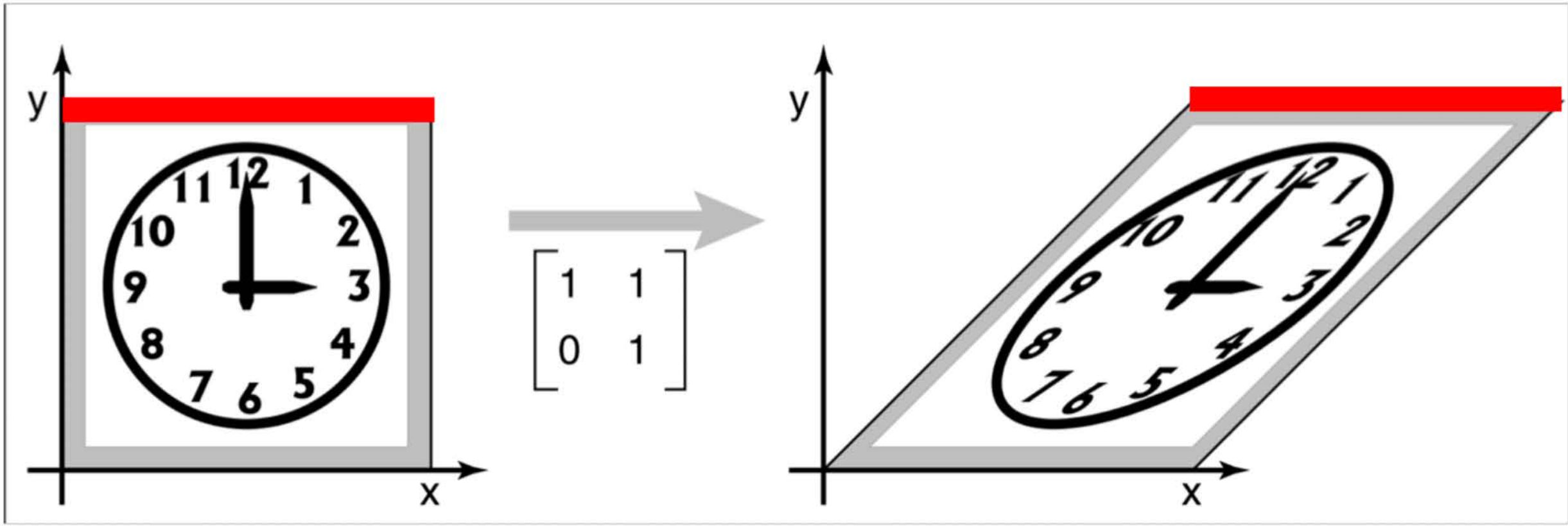
2D Linear Transformations - Rotation

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



2D Linear Transformations - Shear

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



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

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

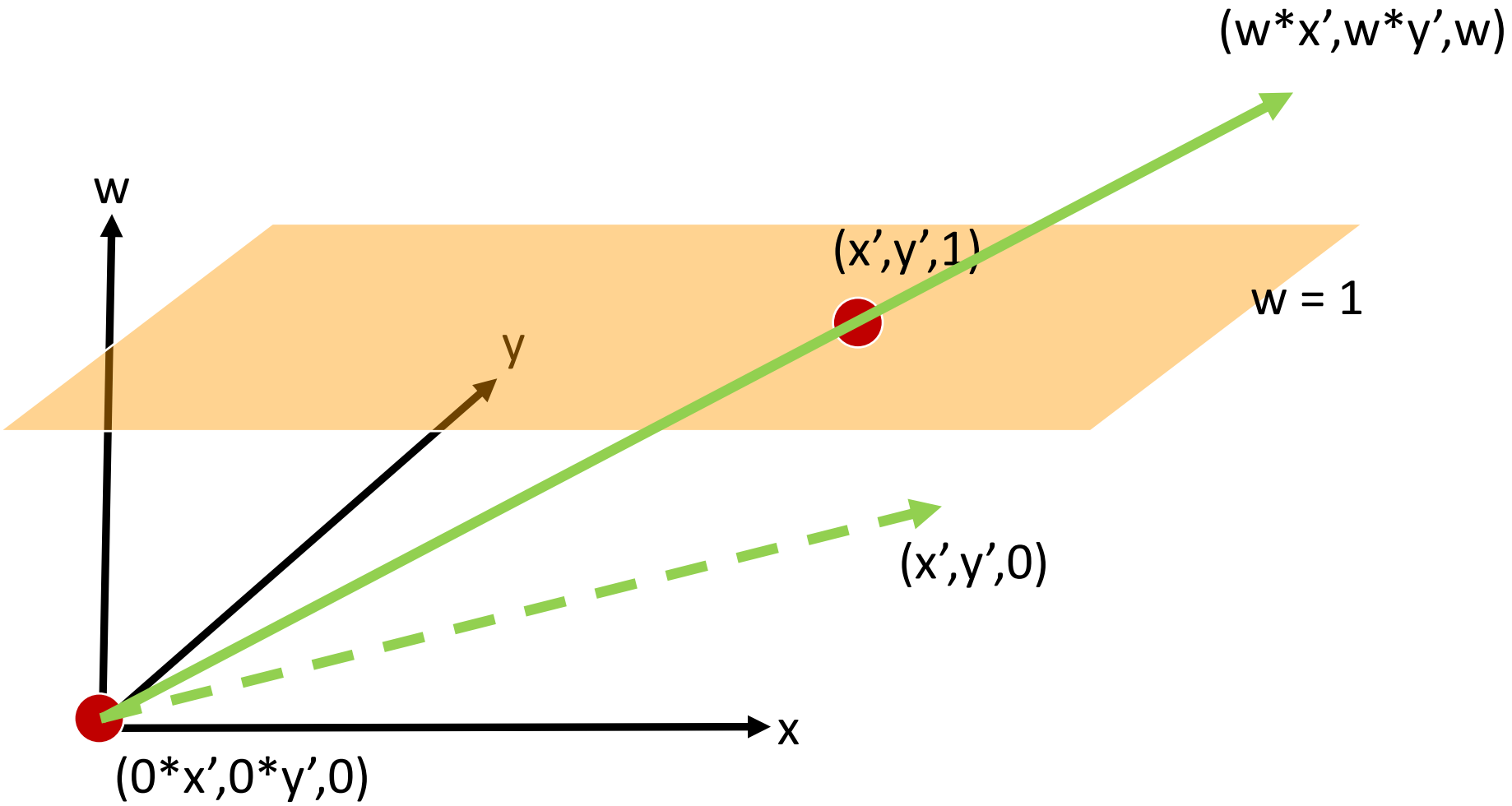
$$Ax + t = b$$

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

Considered as a point in 3D homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Geometric Intuition



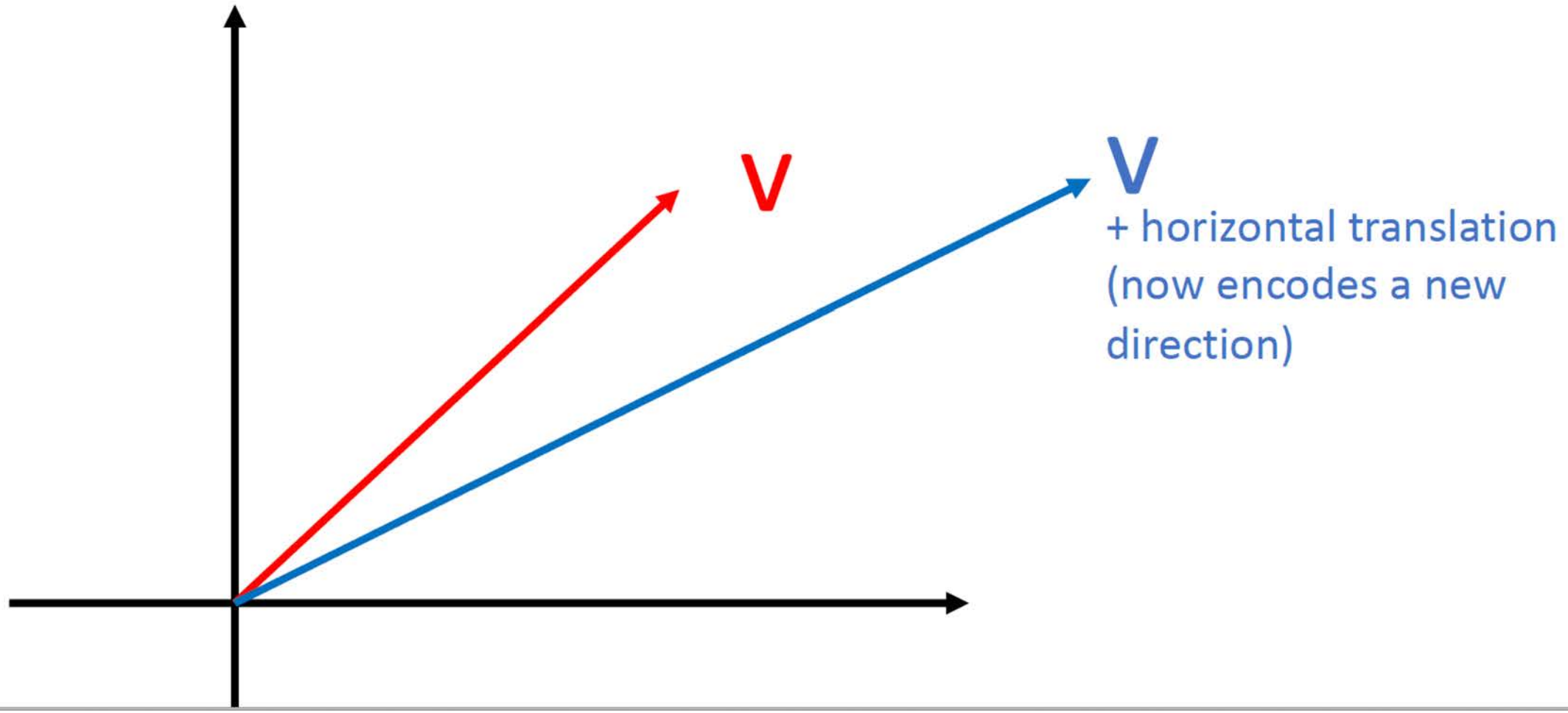
What about vectors?

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

Considered as a ***vector*** in 3D homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

We don’t want to be able to translate a vector



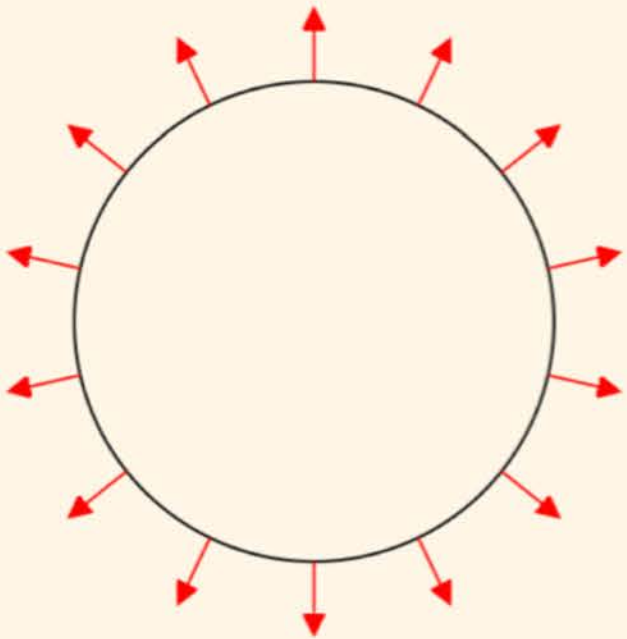
3D Linear Transformations

$$Ax = b$$

Speaking of vectors, do normals get transformed the same way an object does?

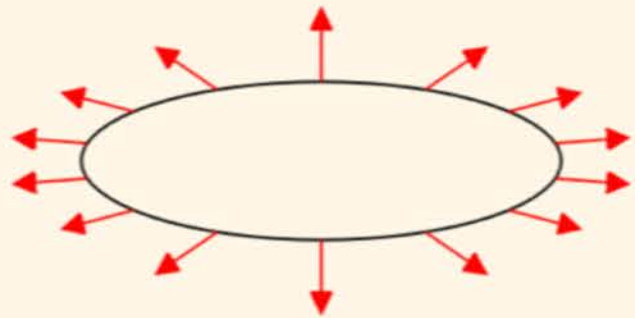
No, thank you for asking.

Original object



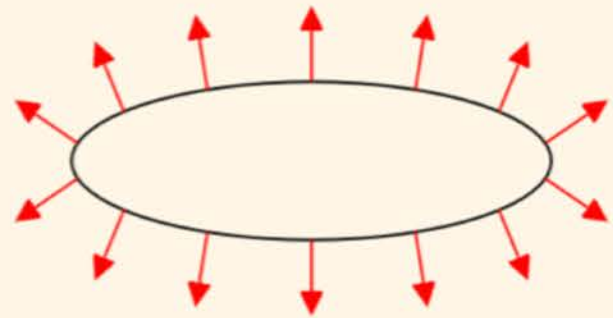
Transformed

With normals using the same transformation matrix



Transformed

With correct transformation applied to normals



https://paroj.github.io/gltut/Illumination/Tut09%20Normal%20Transformation.html

What's the right way to transform a normal vector?

$$n'^T t' = 0$$

$$(Xn)^T (Mt) = 0$$

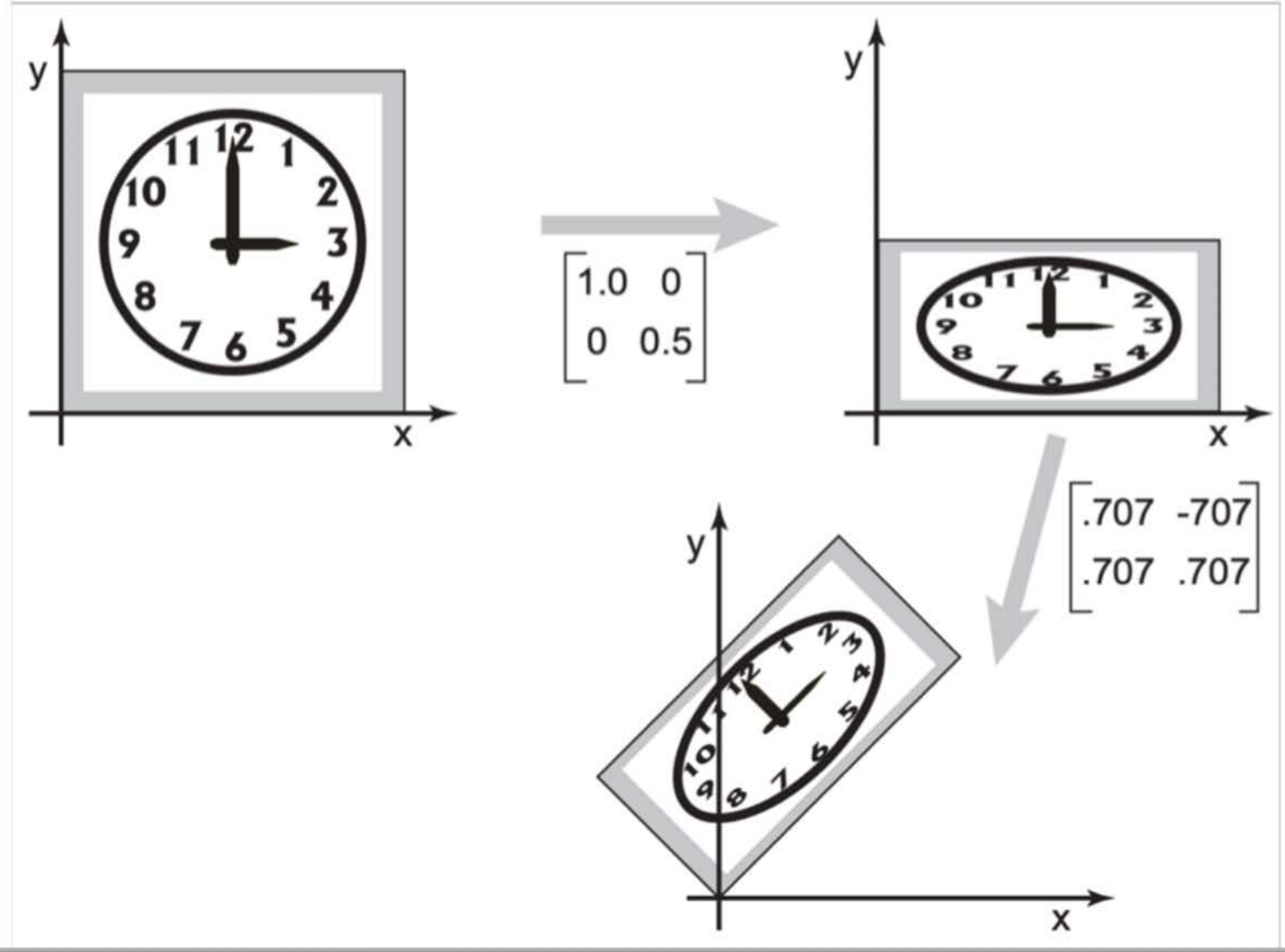
What's the right way to transform a normal vector?

We want $n^T X^T M t = 0$

So, if $X = (M^{-1})^T$ then,

$$n^T X^T M t = n^T M^{-1} M t = n^T t = 0$$

Composing transformations

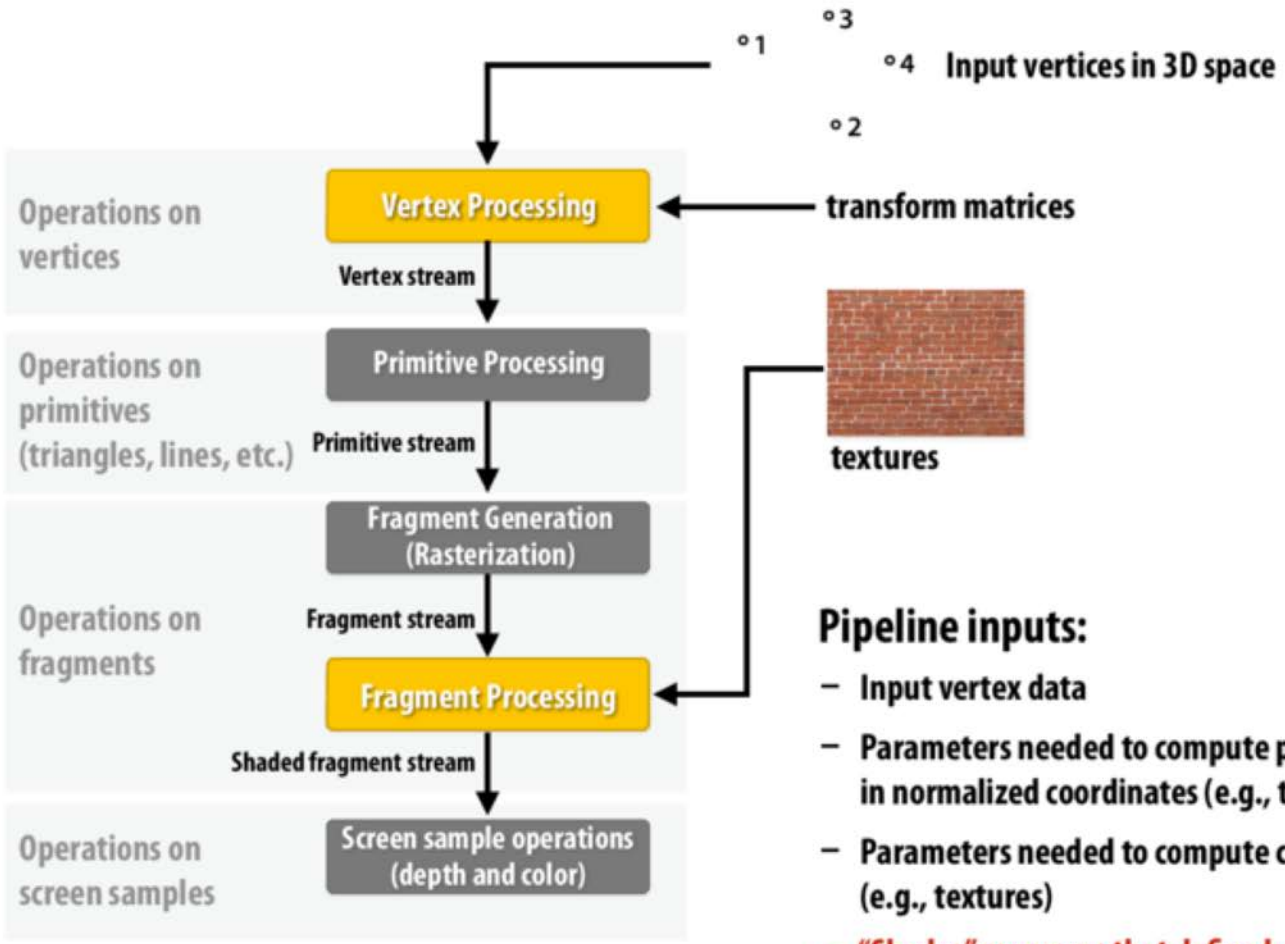


Composing transformations

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

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

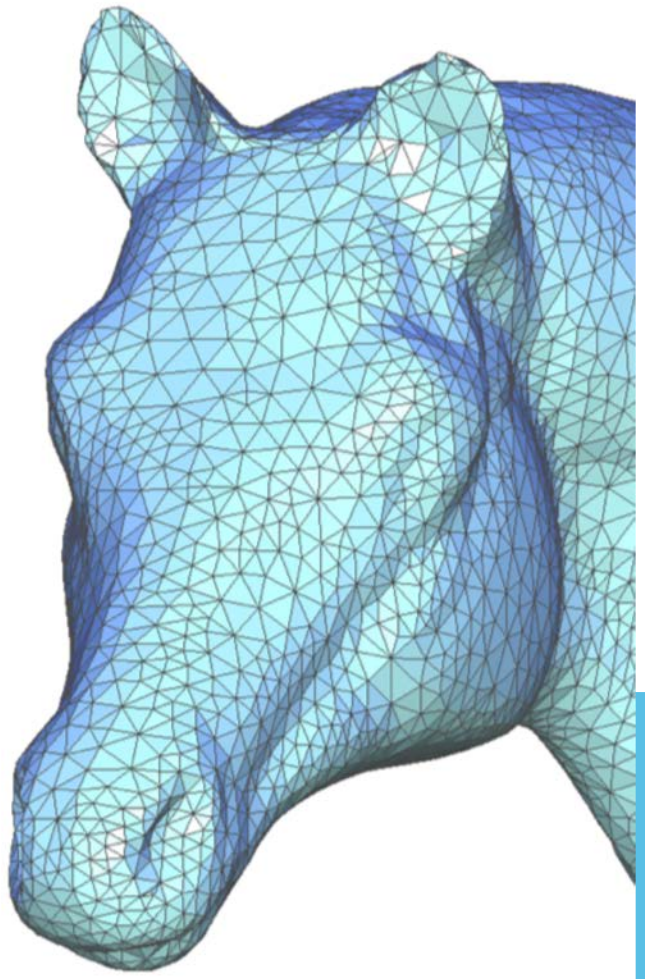


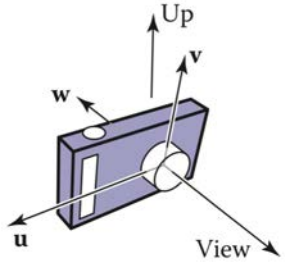
Pipeline inputs:

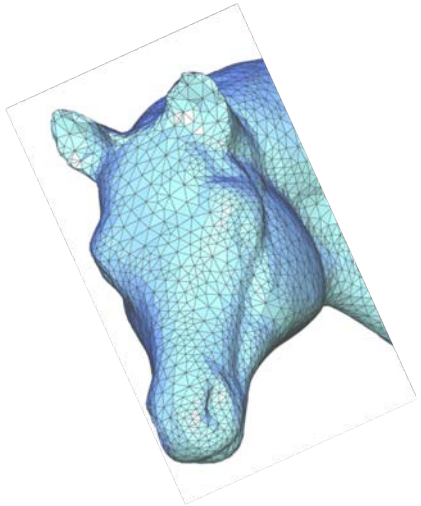
- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

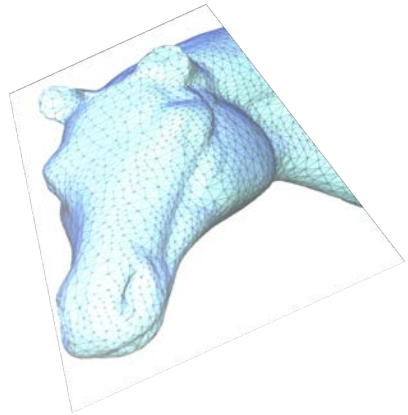
* several stages of the modern OpenGL pipeline are omitted

Getting Things Onto The Screen



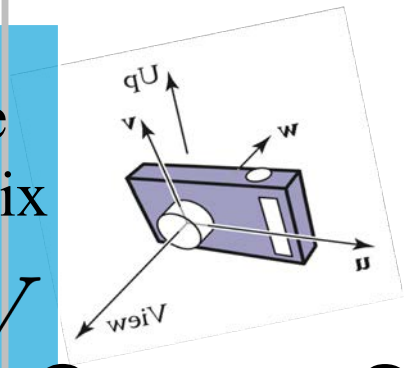


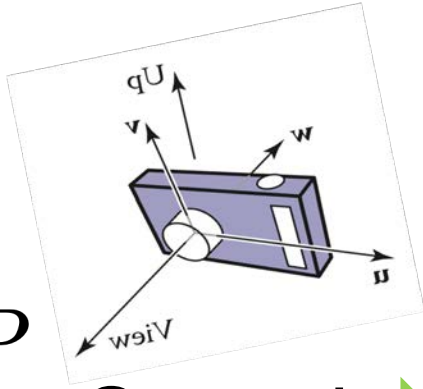




Open GL Combines these into the ModelView Matrix

M V P

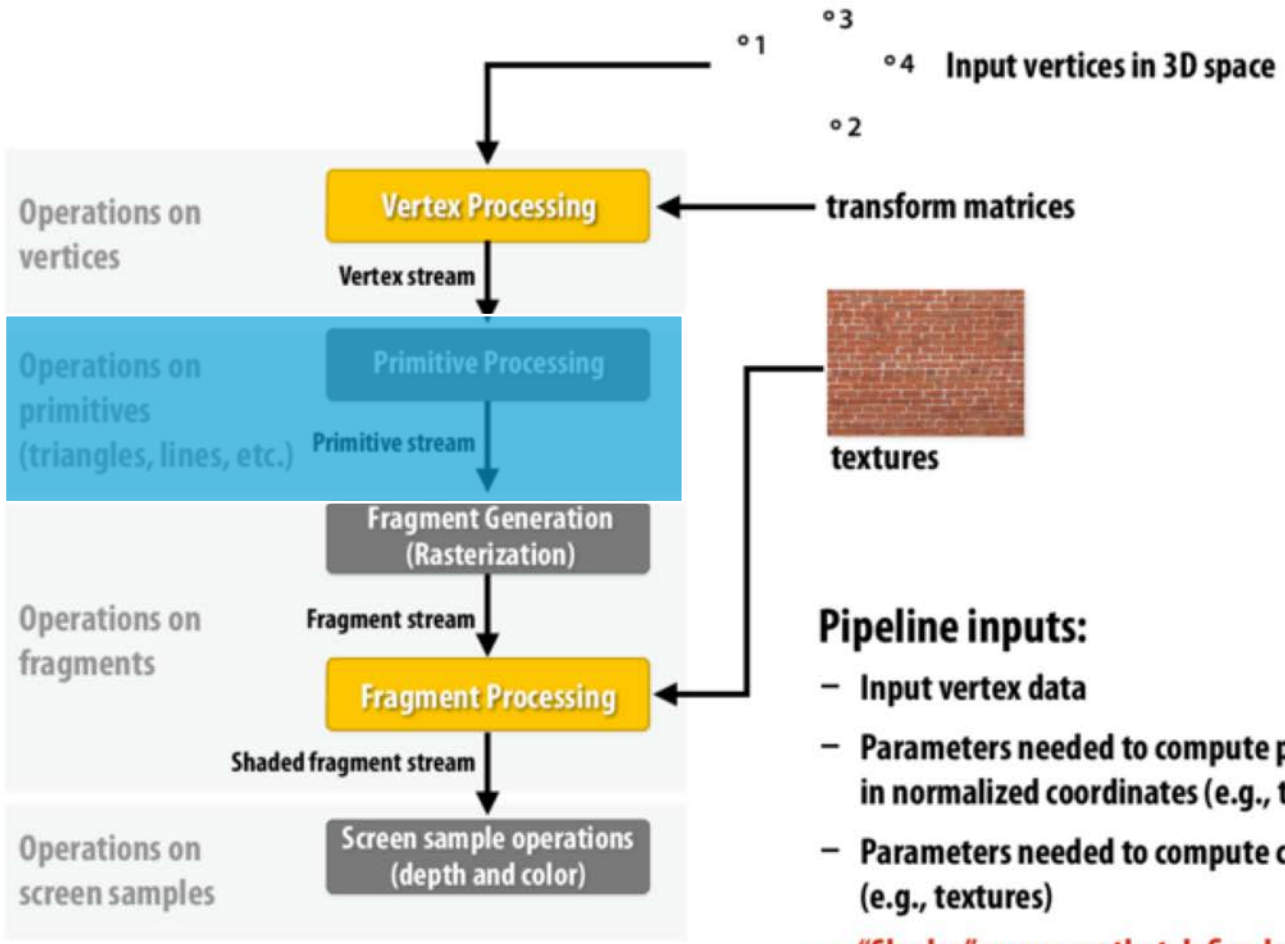




Object Space | World Space | Camera Space | Canonical Space

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

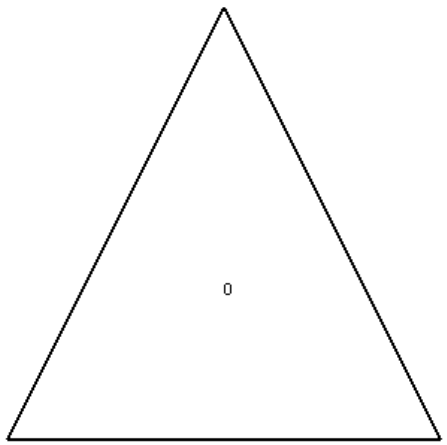


Pipeline inputs:

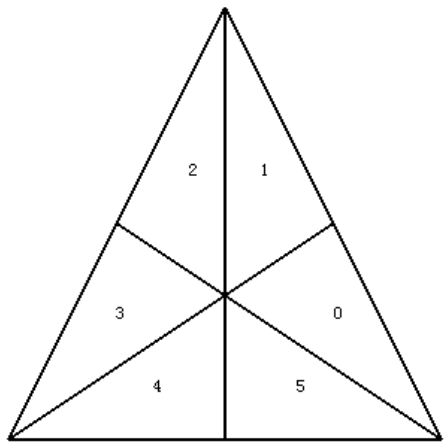
- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

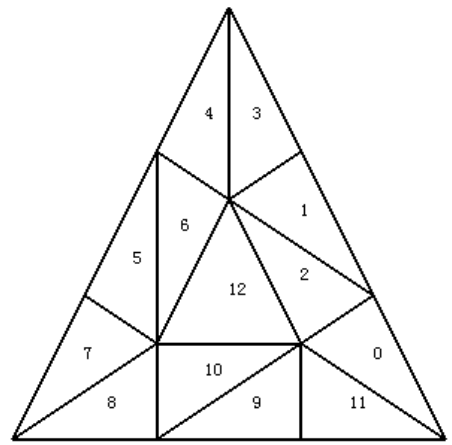
Tessellation Shader



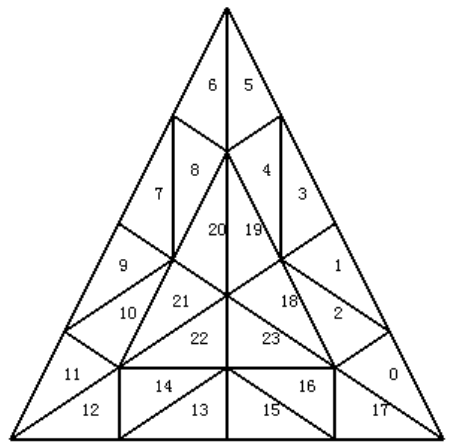
Level 1



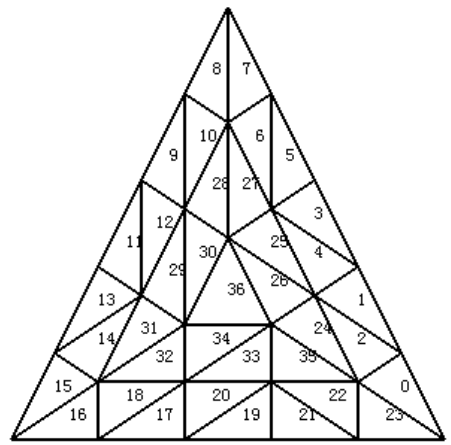
Level 2



Level 3



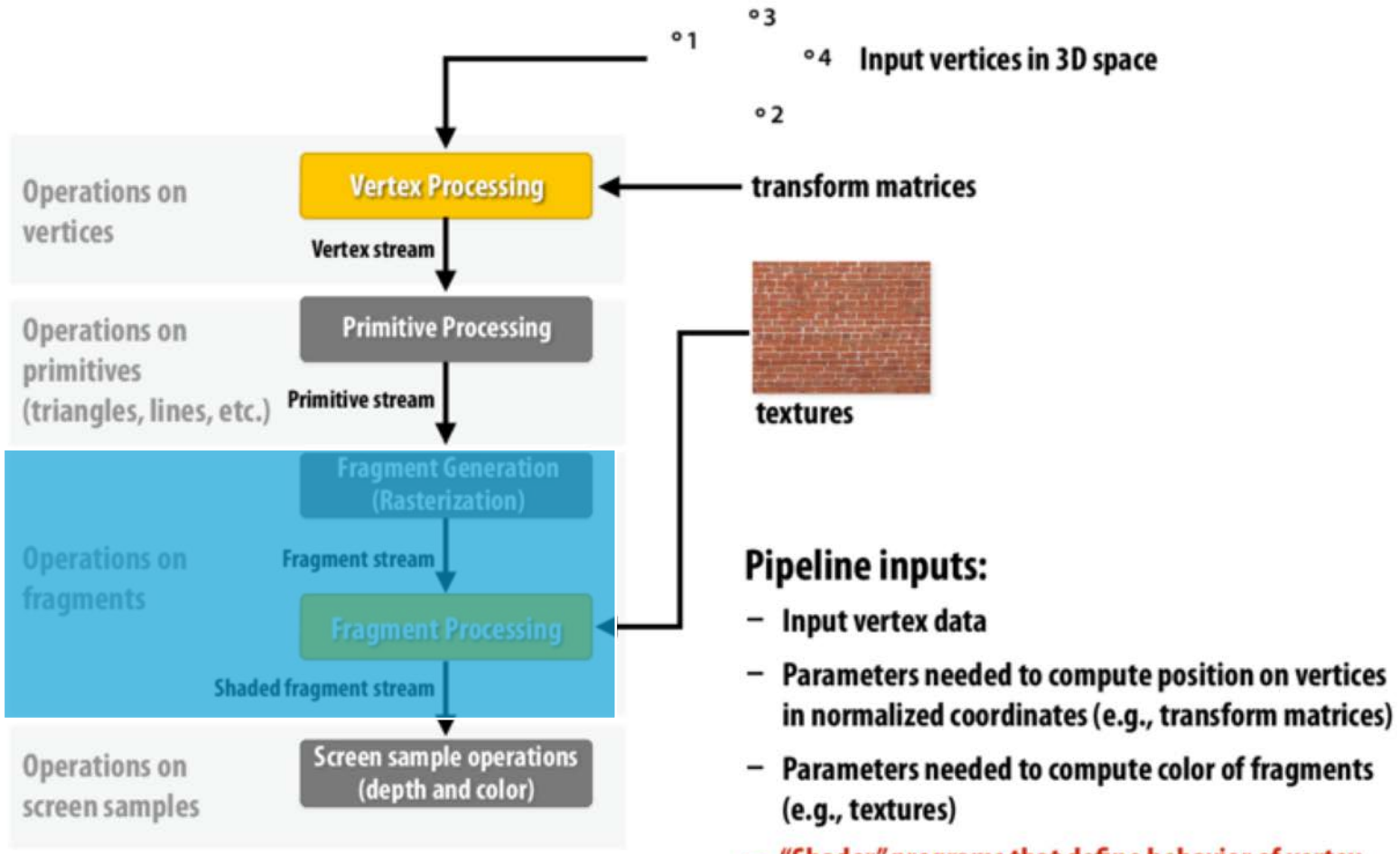
Level 4



Level 5

Modern Graphics Pipeline

OpenGL/Direct3D graphics pipeline *

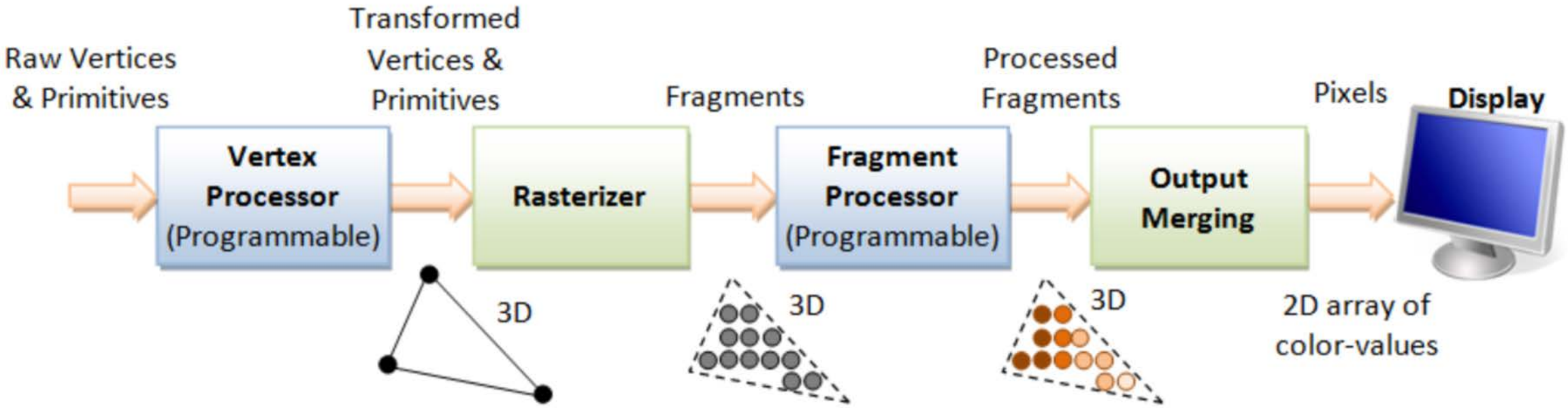


Pipeline inputs:

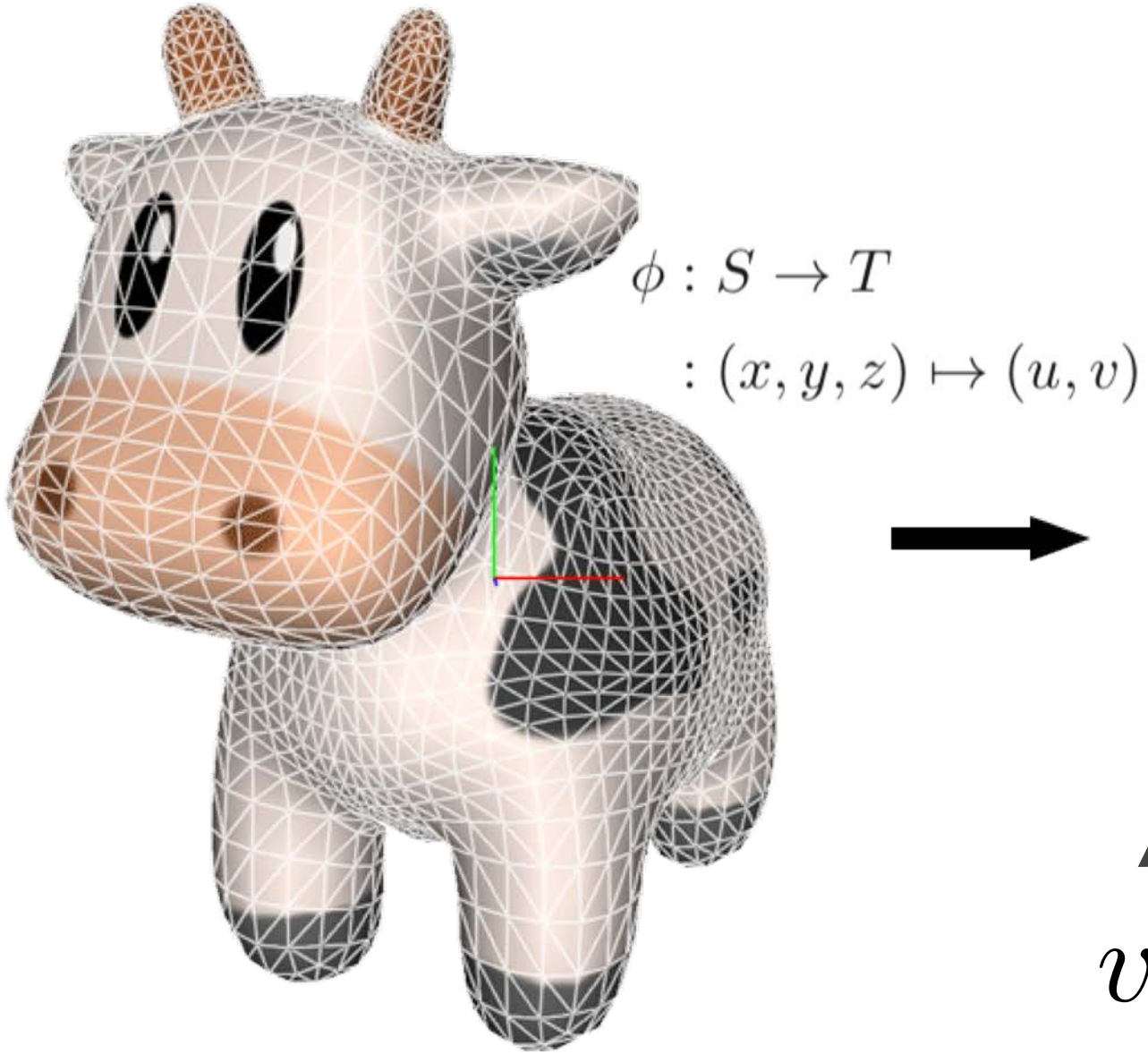
- Input vertex data
- Parameters needed to compute position on vertices in normalized coordinates (e.g., transform matrices)
- Parameters needed to compute color of fragments (e.g., textures)
- "Shader" programs that define behavior of vertex and fragment stages

* several stages of the modern OpenGL pipeline are omitted

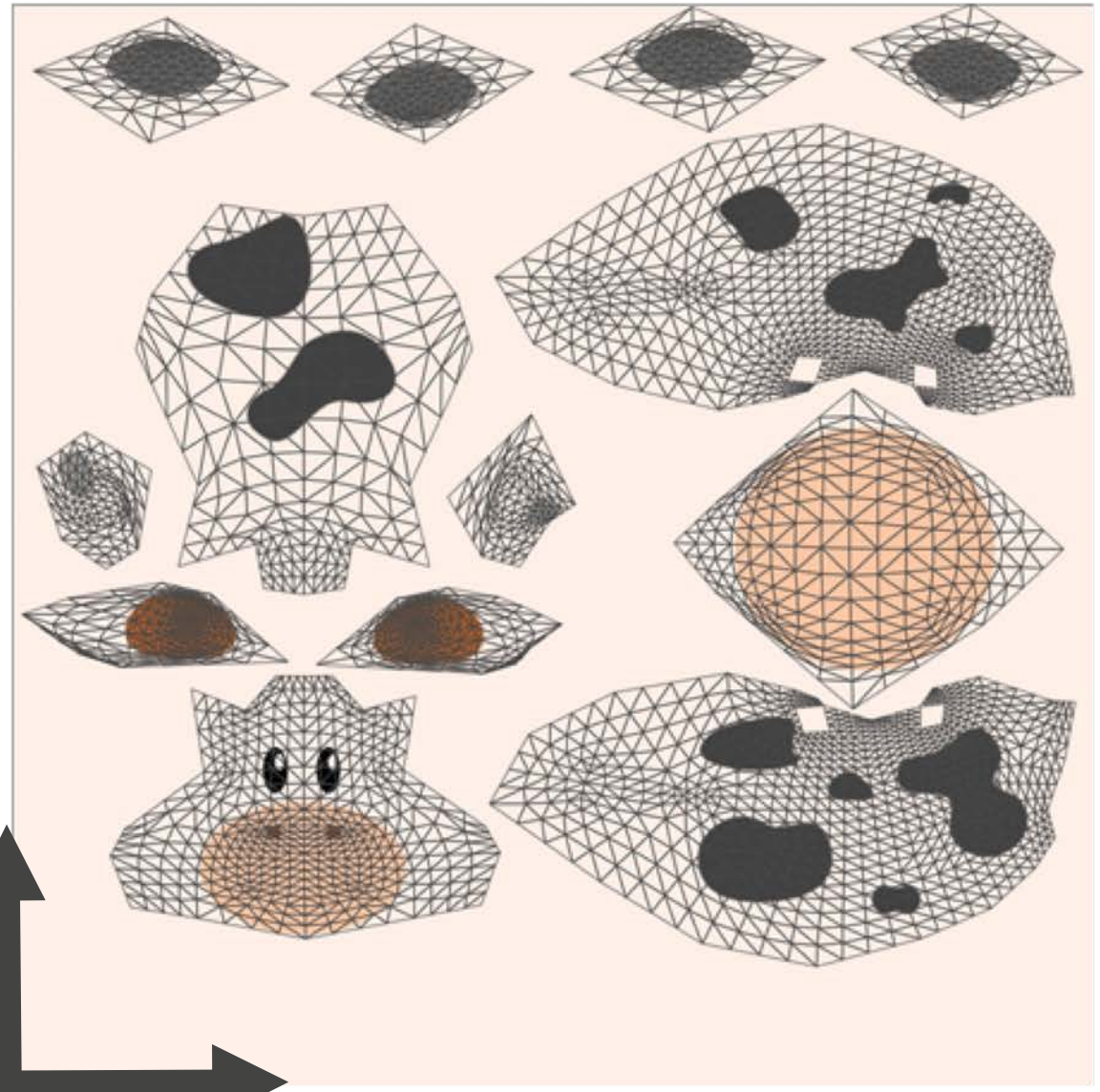
Fragment Shader



Texture coordinates



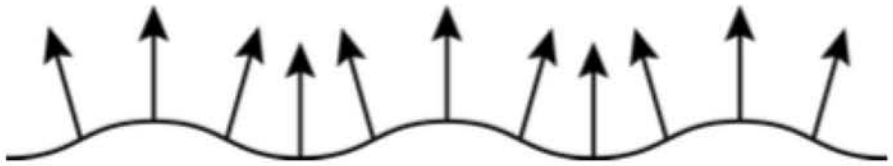
$$\phi : S \to T$$
$$: (x, y, z) \mapsto (u, v)$$

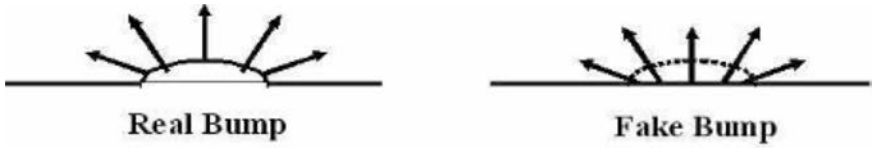


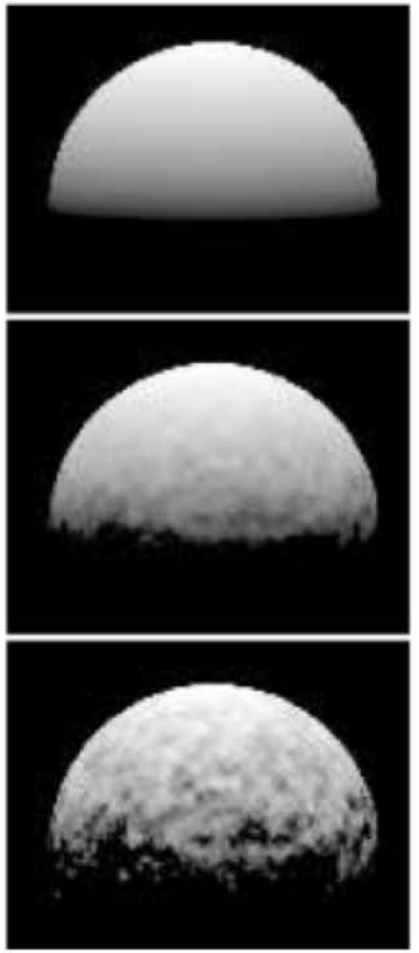
Normal Mapping

One of the reasons why we apply texture mapping:

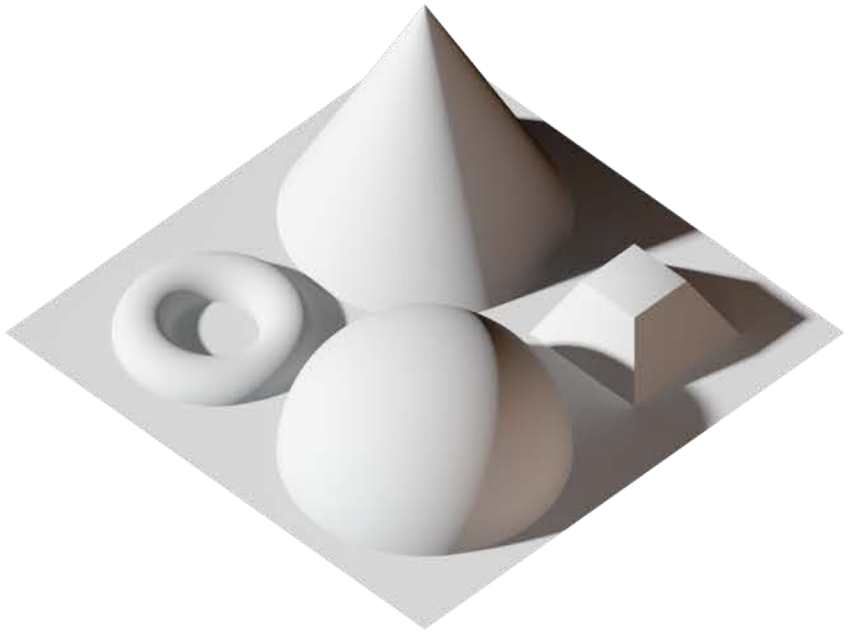
Real surfaces are hardly flat but often rough and bumpy. These bumps cause (slightly) different reflections of the light.

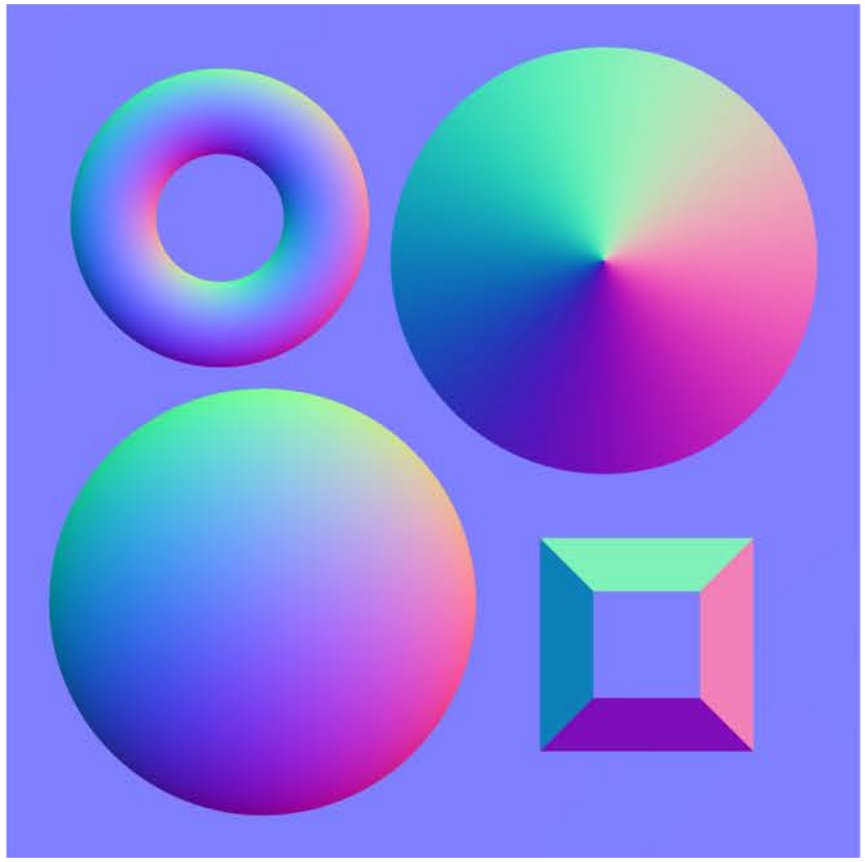


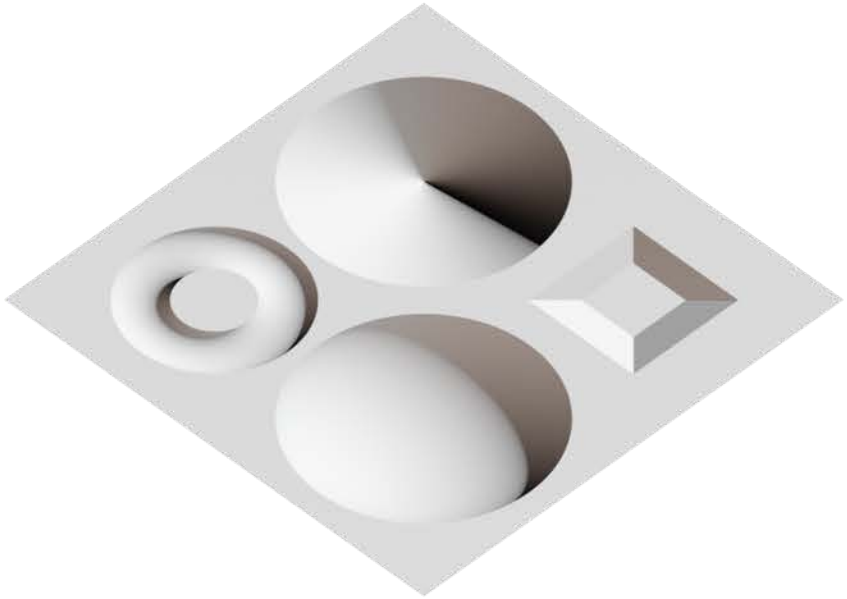




Normal Mapping

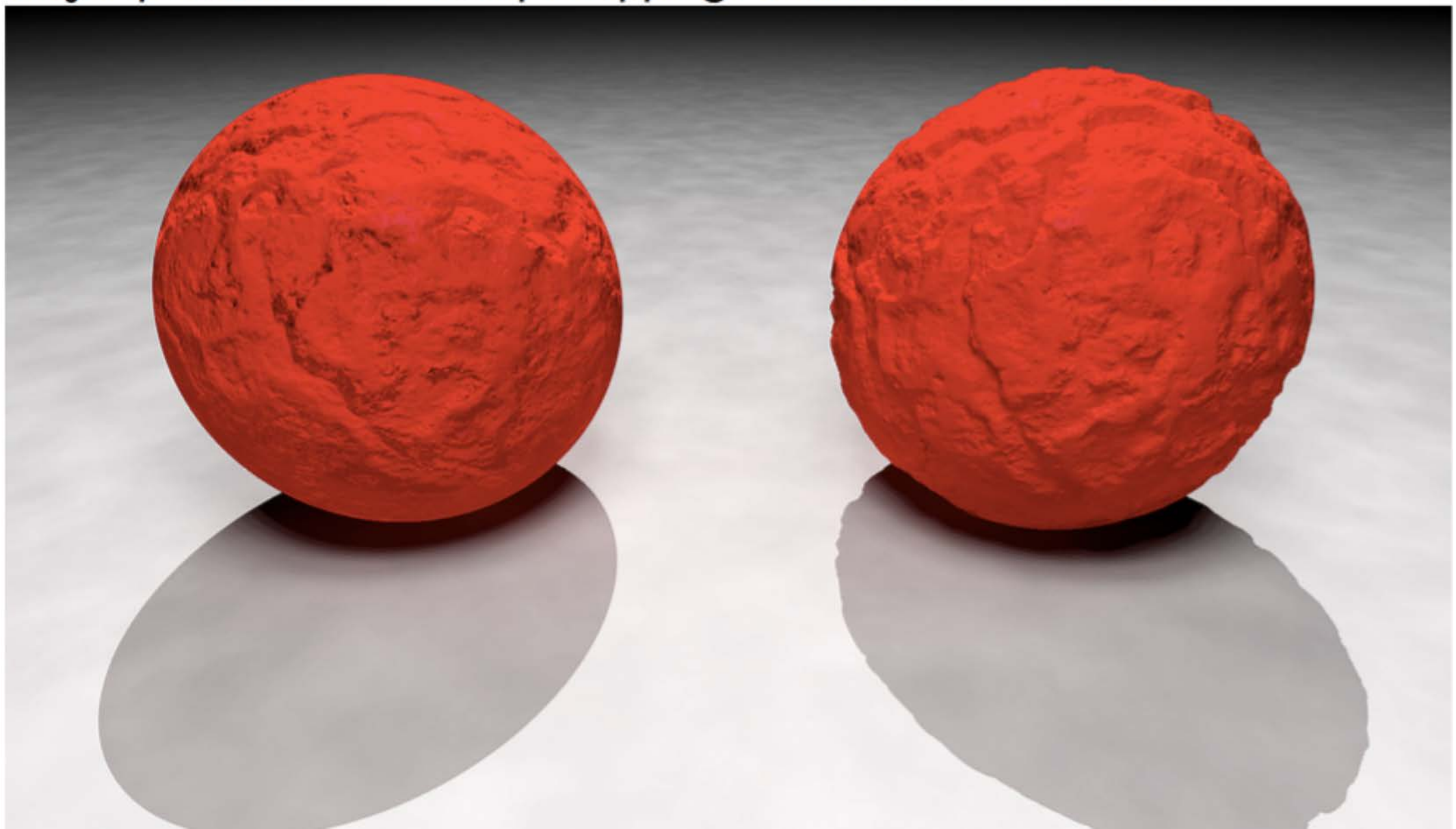




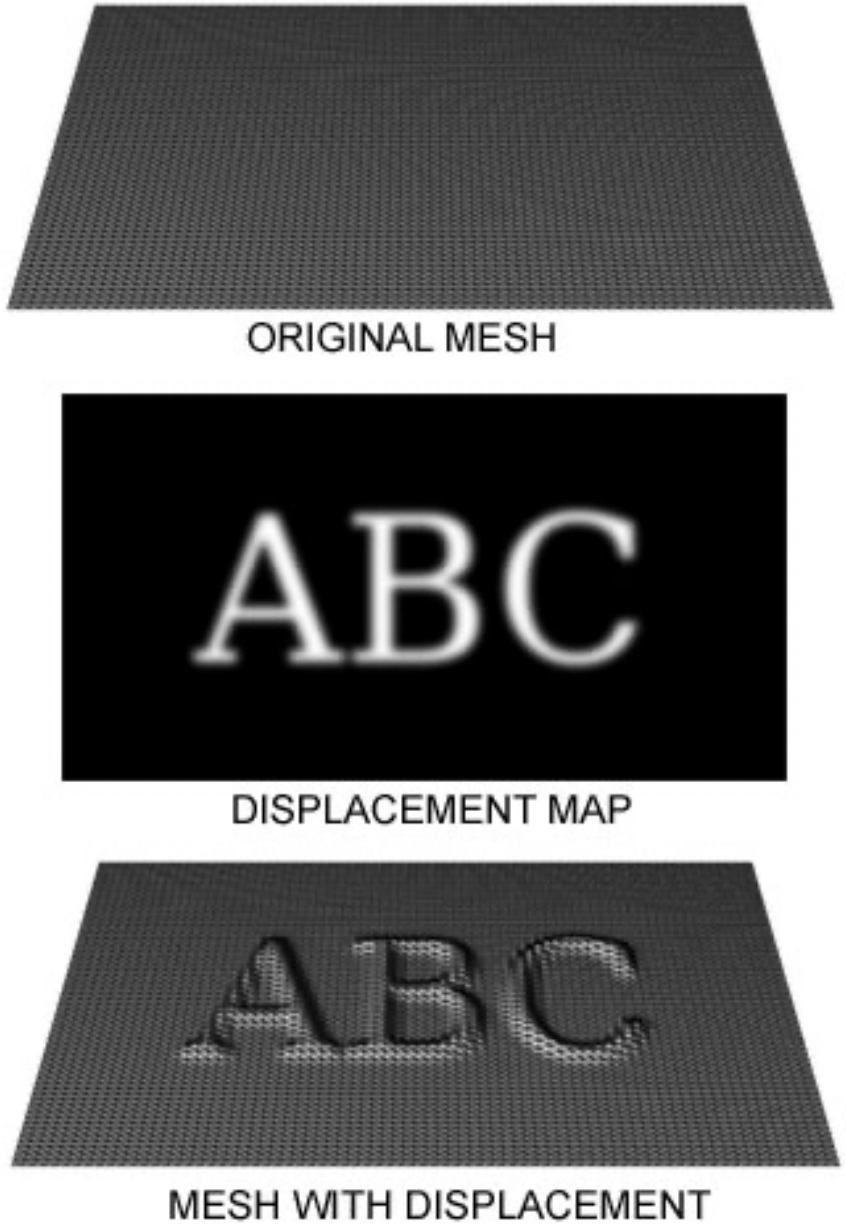


Normal Mapping vs. Geometry

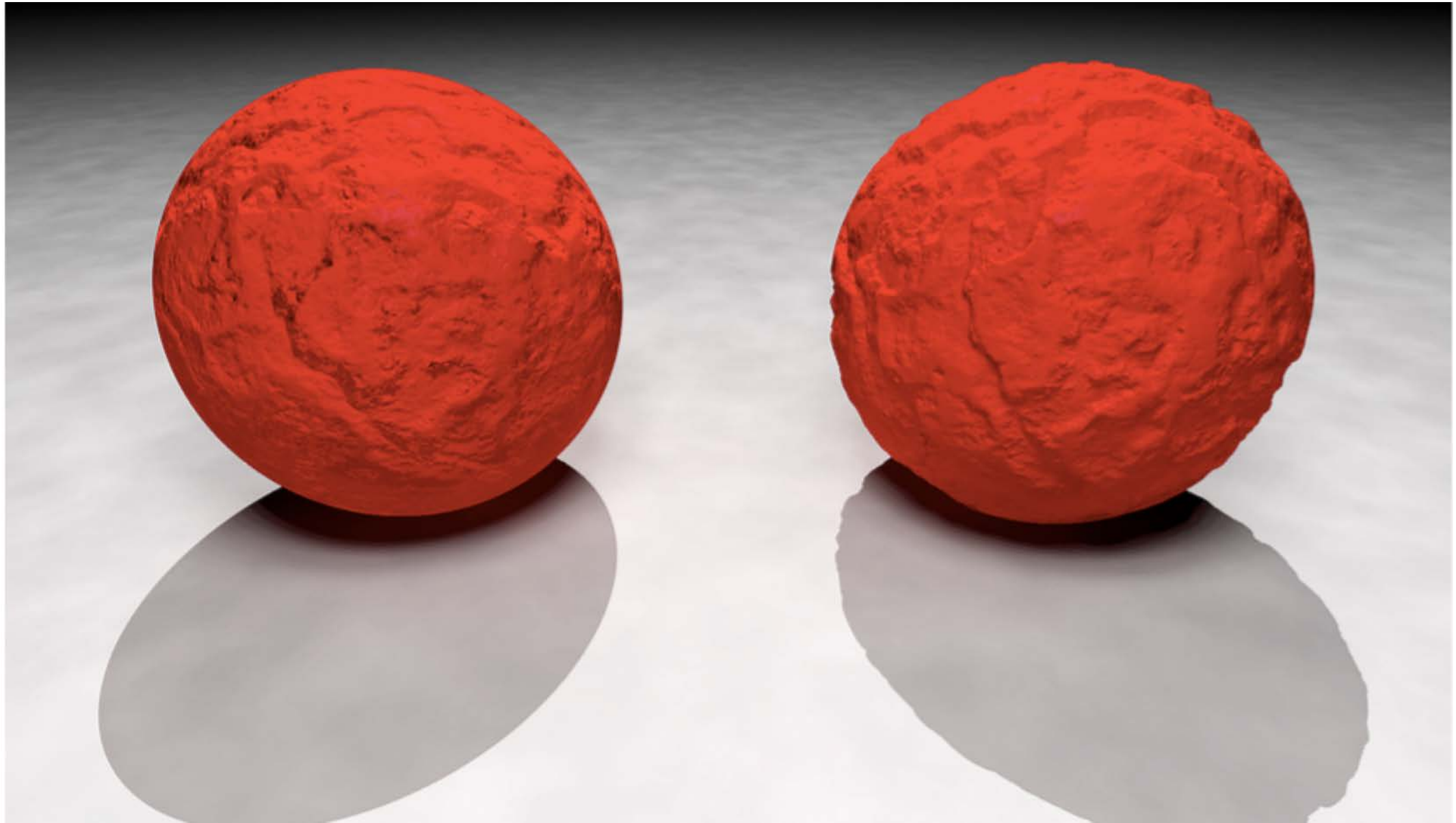
Major problems with bump mapping: silhouettes and shadows



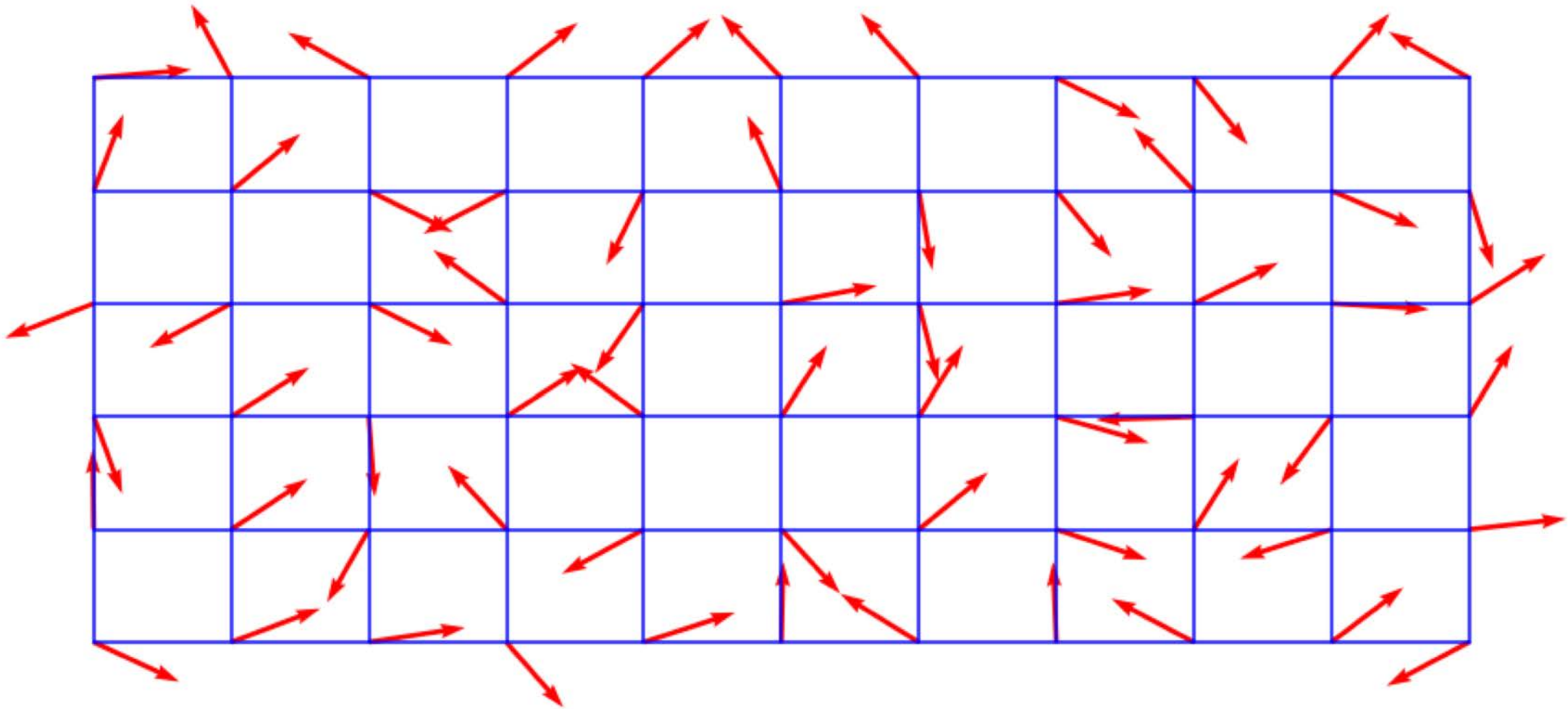
Displacement (Bump) Mapping

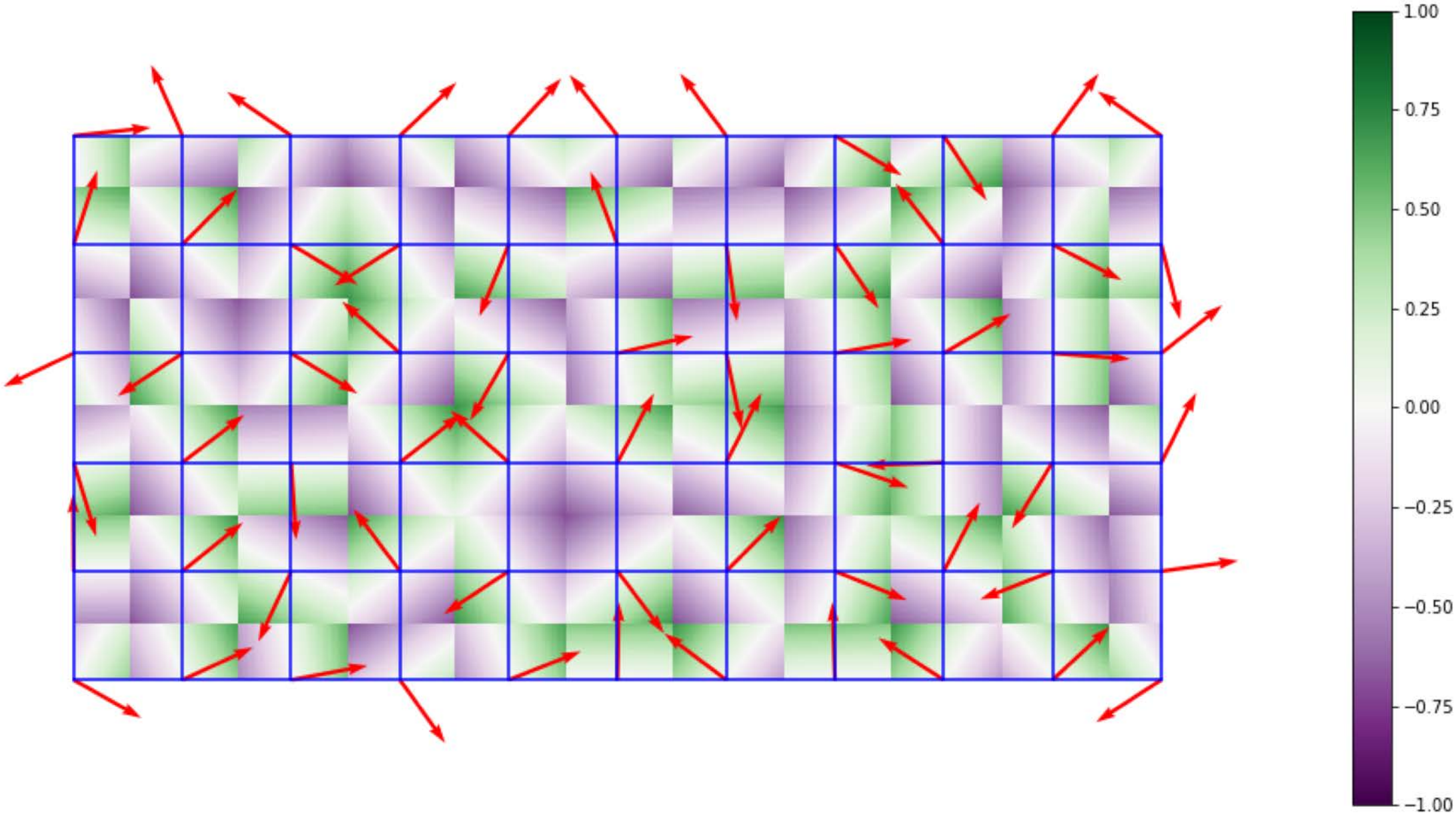


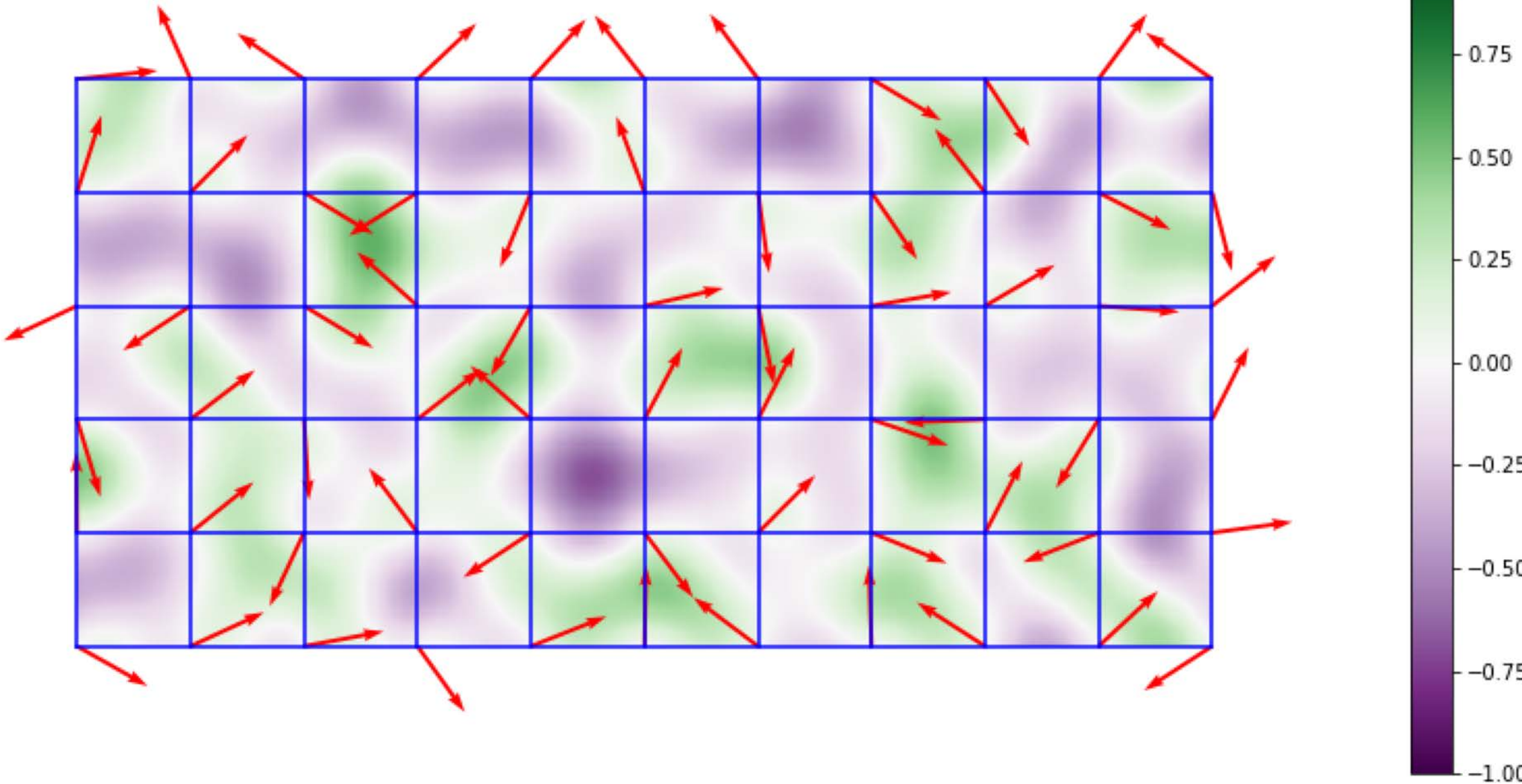
Normal Mapping vs. Displacement Mapping



Perlin Noise




1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

All Done For Today

Office Hours Now in BA5268