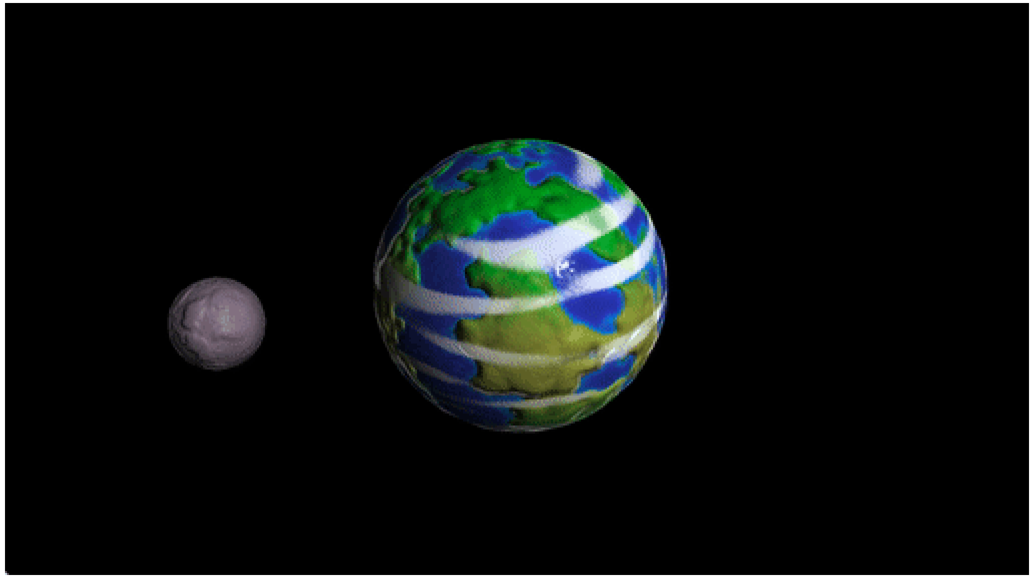
Transformations & Rasterization



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

Transforms and Shaders

Monday:

Reminder – Rasterization

Introduction to the Graphics Pipeline

Transformations

Today:

Review Shader Pipeline

OpenGL Data Structures

Normal and Bump Mapping

Perlin Noise

Midterm

Announcements

Midterm marks out

A3 marks out by 20 July (drop date)

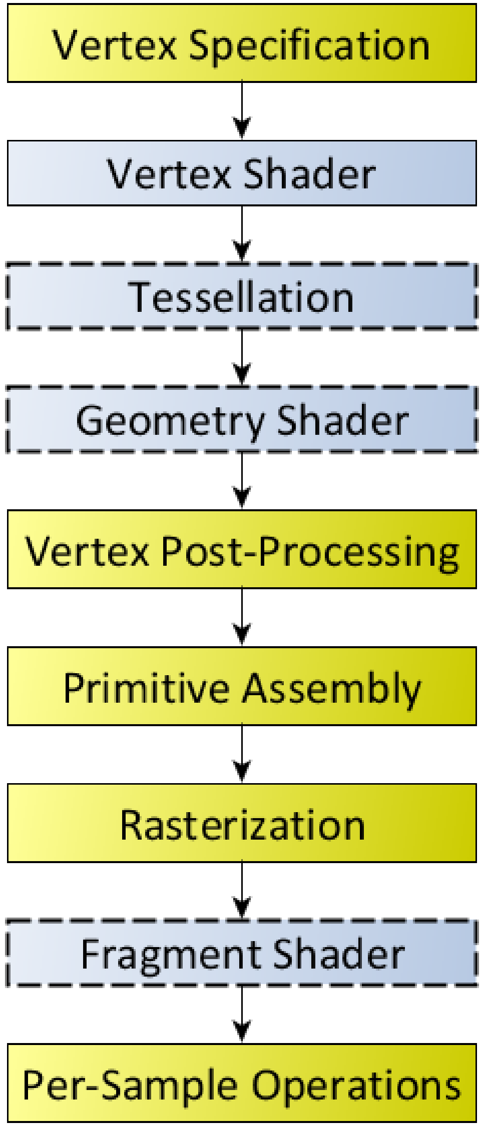
A6 due 23 July – please try to get running asap

Office hours after the lecture today

Tutorial on Friday 17 July to ask about midterm and A6

Any Questions ?

Modern Graphics Pipeline



Set up VAOs and VBOs
To send vertices to OpenGL in their preferred format

https://www.khronos.org/opengl/wiki/Vertex_Shader#Inputs

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

Modern Graphics Pipeline

A Vertex Array Object (VAO) is an object which contains one or more Vertex Buffer Objects and is designed to store the information for a complete rendered object. In our example this is a diamond consisting of four vertices as well as a color for each vertex.

```
/* We're going to create a simple diamond made from lines */
const GLfloat diamond[4][2] = {
{ 0.0, 1.0 }, /* Top point */
{ 1.0, 0.0 }, /* Right point */
{ 0.0, -1.0 }, /* Bottom point */
{ -1.0, 0.0 } }; /* Left point */

const GLfloat colors[4][3] = {
{ 1.0, 0.0, 0.0 }, /* Red */
{ 0.0, 1.0, 0.0 }, /* Green */
{ 0.0, 0.0, 1.0 }, /* Blue */
{ 1.0, 1.0, 1.0 } }; /* White */
```

https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

Modern Graphics Pipeline

A Vertex Array Object (VAO) is an object which contains one or more Vertex Buffer Objects and is designed to store the information for a complete rendered object. In our example this is a diamond consisting of four vertices as well as a color for each vertex.

```
/* Create handles for our Vertex Array Object and two Vertex Buffer Objects */
GLuint vao, vbo[2];

/* Allocate and assign a Vertex Array Object to our handle */
glGenVertexArrays(1, &vao);

/* Bind our Vertex Array Object as the current used object */
glBindVertexArray(vao);
```

https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

Modern Graphics Pipeline

A Vertex Buffer Object (VBO) is a memory buffer in the high speed memory of your video card designed to hold information about vertices. In our example we have two VBOs, one that describes the coordinates of our vertices and another that describes the color associated with each vertex. VBOs can also store information such as normals, texcoords, indices, etc.

```
/* Allocate and assign two Vertex Buffer Objects to our handle */
glGenBuffers(2, vbo);

/* Bind our first VBO as being the active buffer and storing vertex attributes
(coordinates) */
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
```

https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

Modern Graphics Pipeline

A Vertex Buffer Object (VBO) is a memory buffer in the high speed memory of your video card designed to hold information about vertices. In our example we have two VBOs, one that describes the coordinates of our vertices and another that describes the color associated with each vertex. VBOs can also store information such as normals, texcoords, indices, etc.

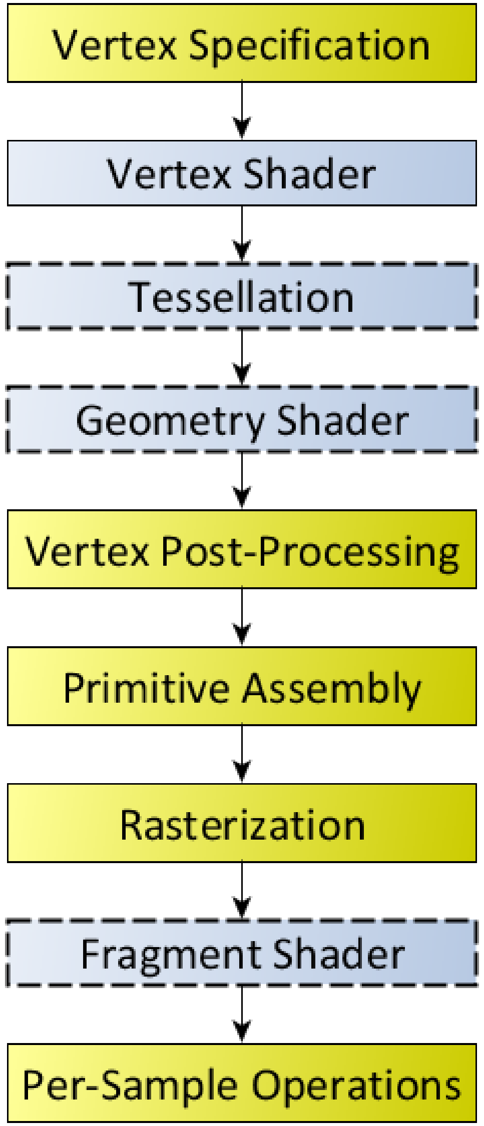
```
/* Copy the vertex data from diamond to our buffer */
/* 8 * sizeof(GLfloat) is the size of the diamond array, since it contains 8
GLfloat values */
glBufferData(GL_ARRAY_BUFFER, 8 * sizeof(GLfloat), diamond, GL_STATIC_DRAW);

/* Specify that our coordinate data is going into attribute index 0, and
contains two floats per vertex */
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);

/* Enable attribute index 0 as being used */
glEnableVertexAttribArray(0);
```

https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

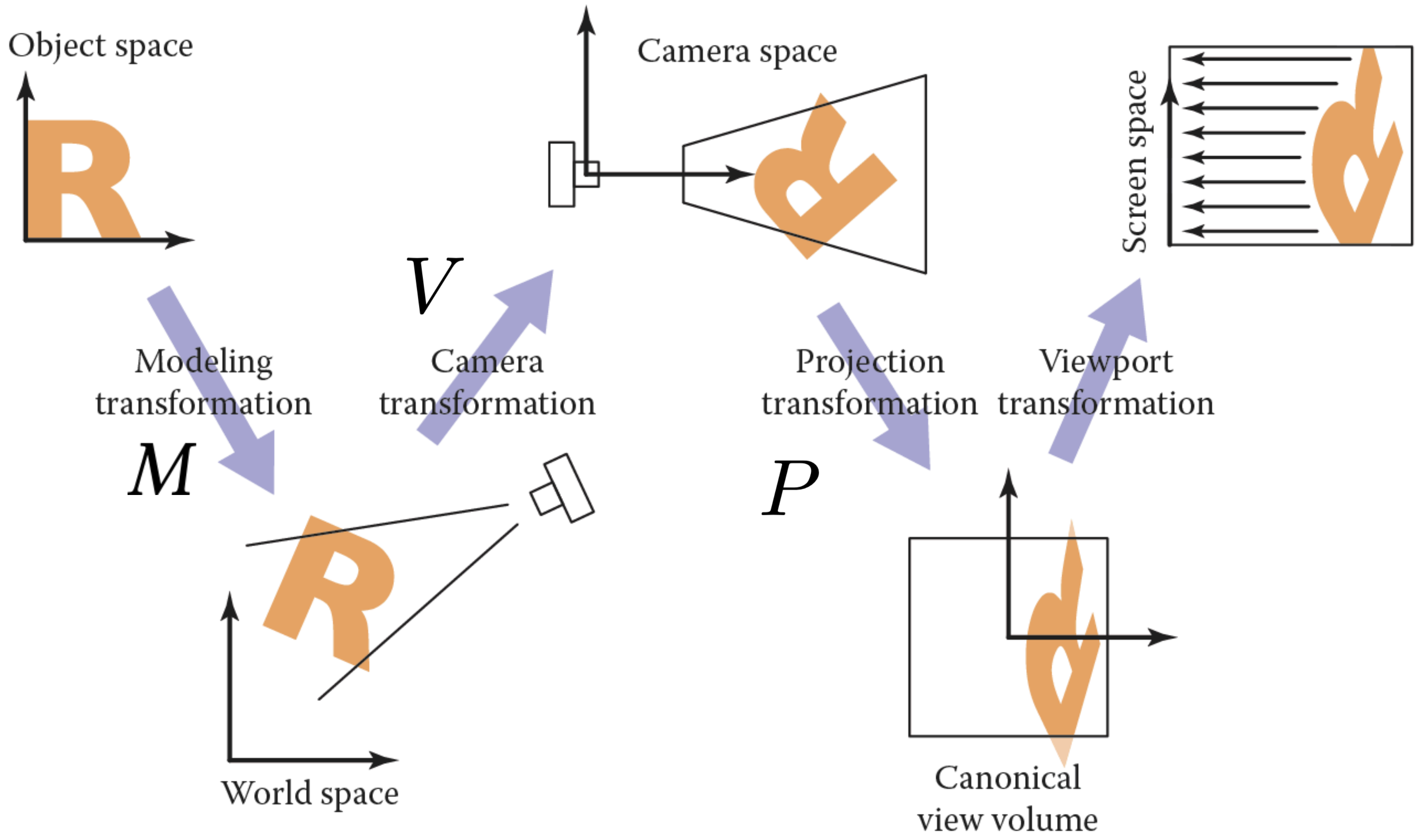
Modern Graphics Pipeline



Programmable vertex shader
Performs operation to every vertex
Model, View, Projection

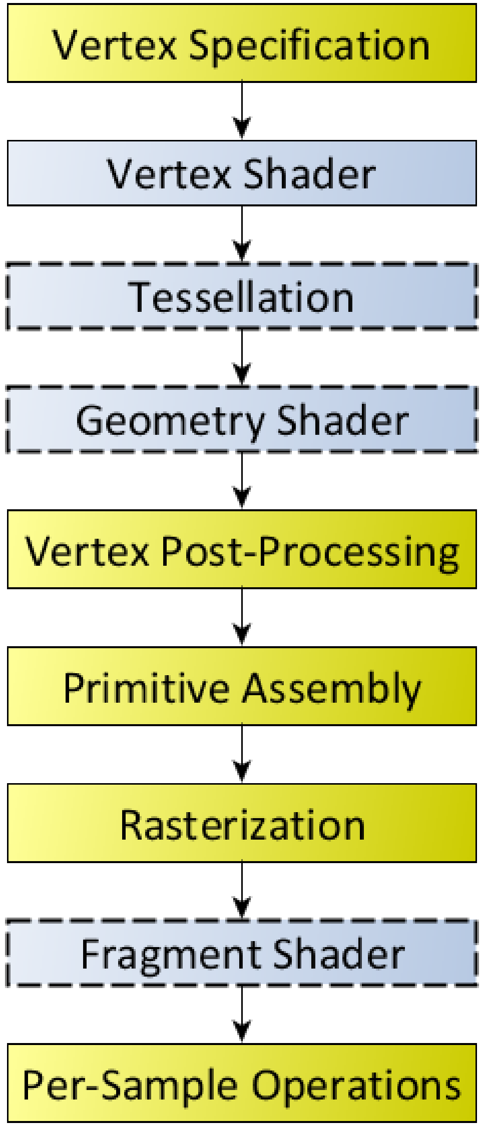
https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

Getting Things Onto The Screen

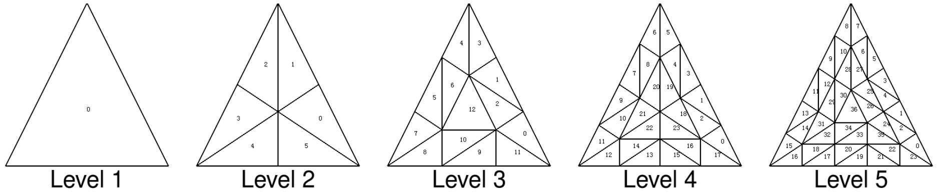


Good explanation on the derivation of these matrices found here and Chapter 7 of the book
https://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/

Modern Graphics Pipeline

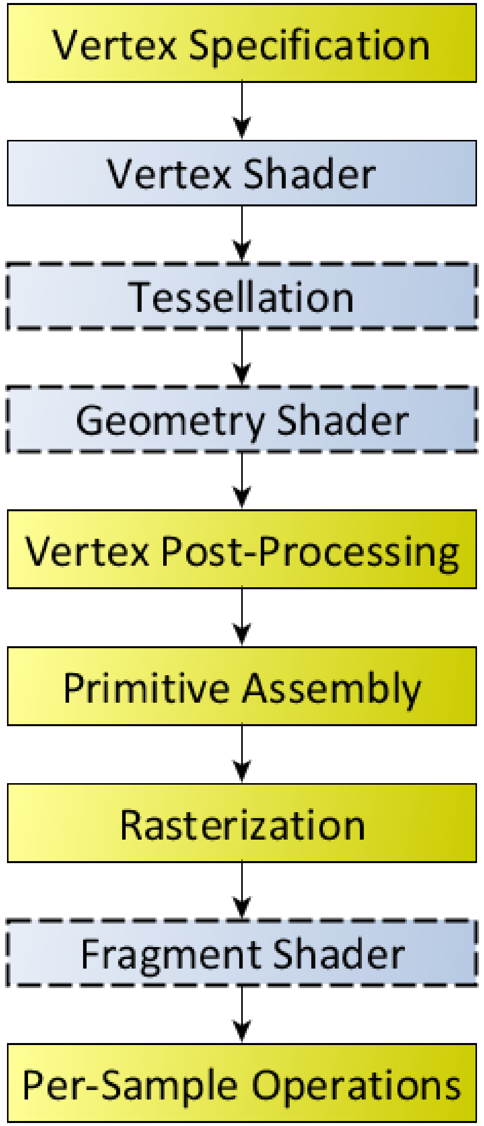


Tessellation Control Shader
Tessellation Evaluation Shader



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

Modern Graphics Pipeline



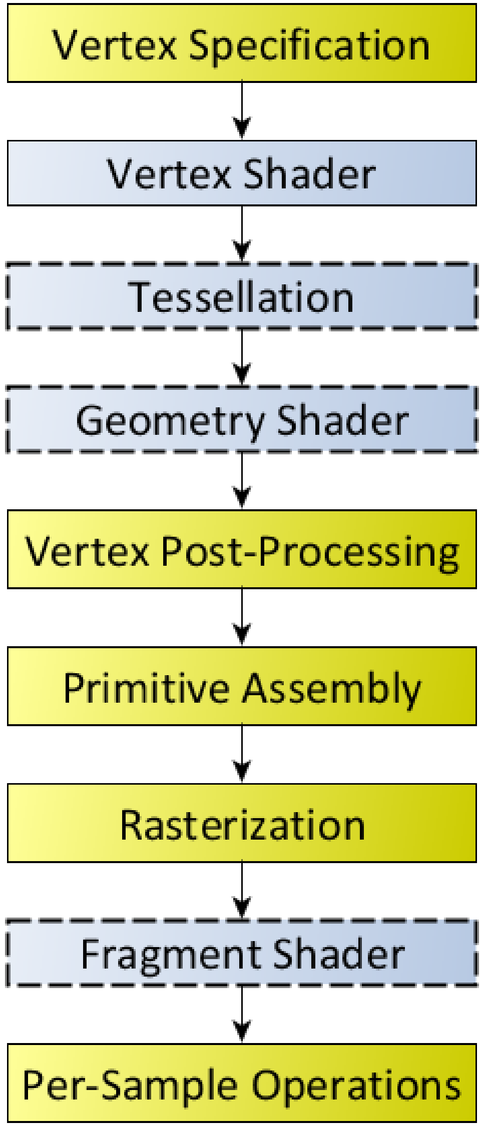
Clipping
Perspective divide and viewport transform to window space

After the perspective projection matrix multiplication in the vertex shader, we need to divide all x,y,z values by w to get them into normalized device coordinates. Then puts things into window space correctly.

https://www.learnopengles.com/tag/perspective-divide/

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

Modern Graphics Pipeline



Fragment generation from primitives
At least one fragment for every pixel area covered by primitive being rasterized

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

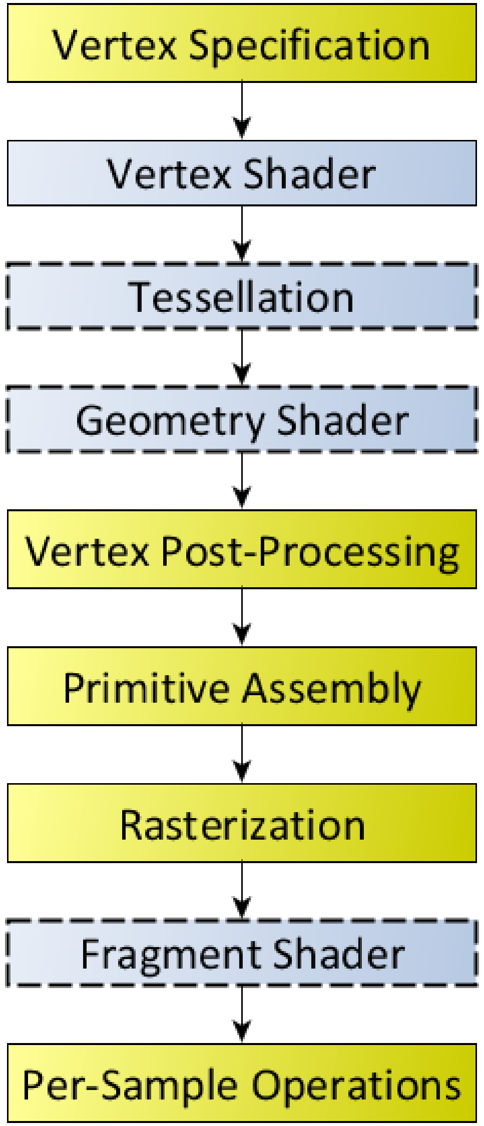
Modern Graphics Pipeline

Drawing to the screen

```
glBindVertexArray(VAO);
glDrawElements(GL_PATCHES, F.size(), GL_UNSIGNED_INT, 0);
glBindVertexArray(0);
```

https://www.khronos.org/opengl/wiki/Tutorial2:_VAOs,_VBOs,_Vertex_and_Fragment_Shaders_(C_/_SDL)

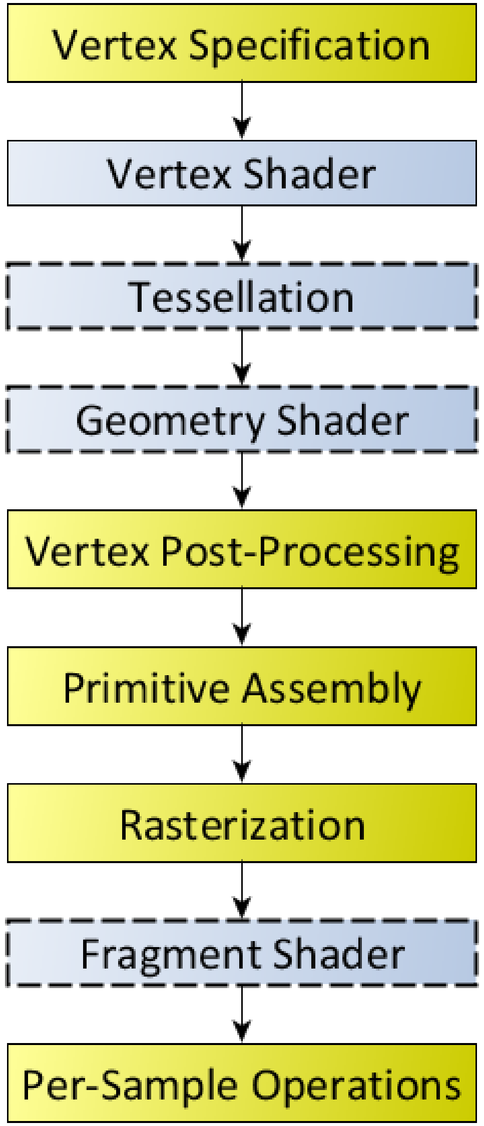
Modern Graphics Pipeline



Programmable shader operates on every fragment
Each output fragment has window space position, depth value, and and zero or more color values

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

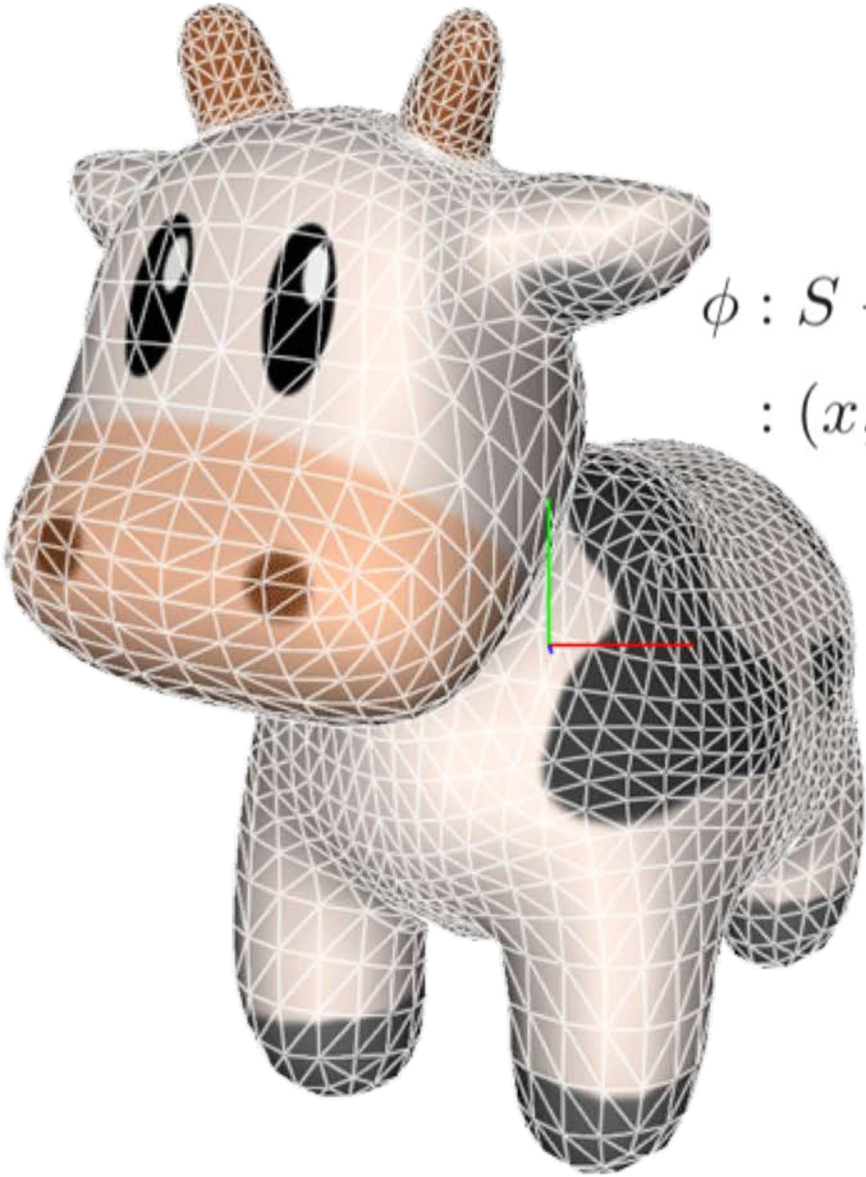
Modern Graphics Pipeline



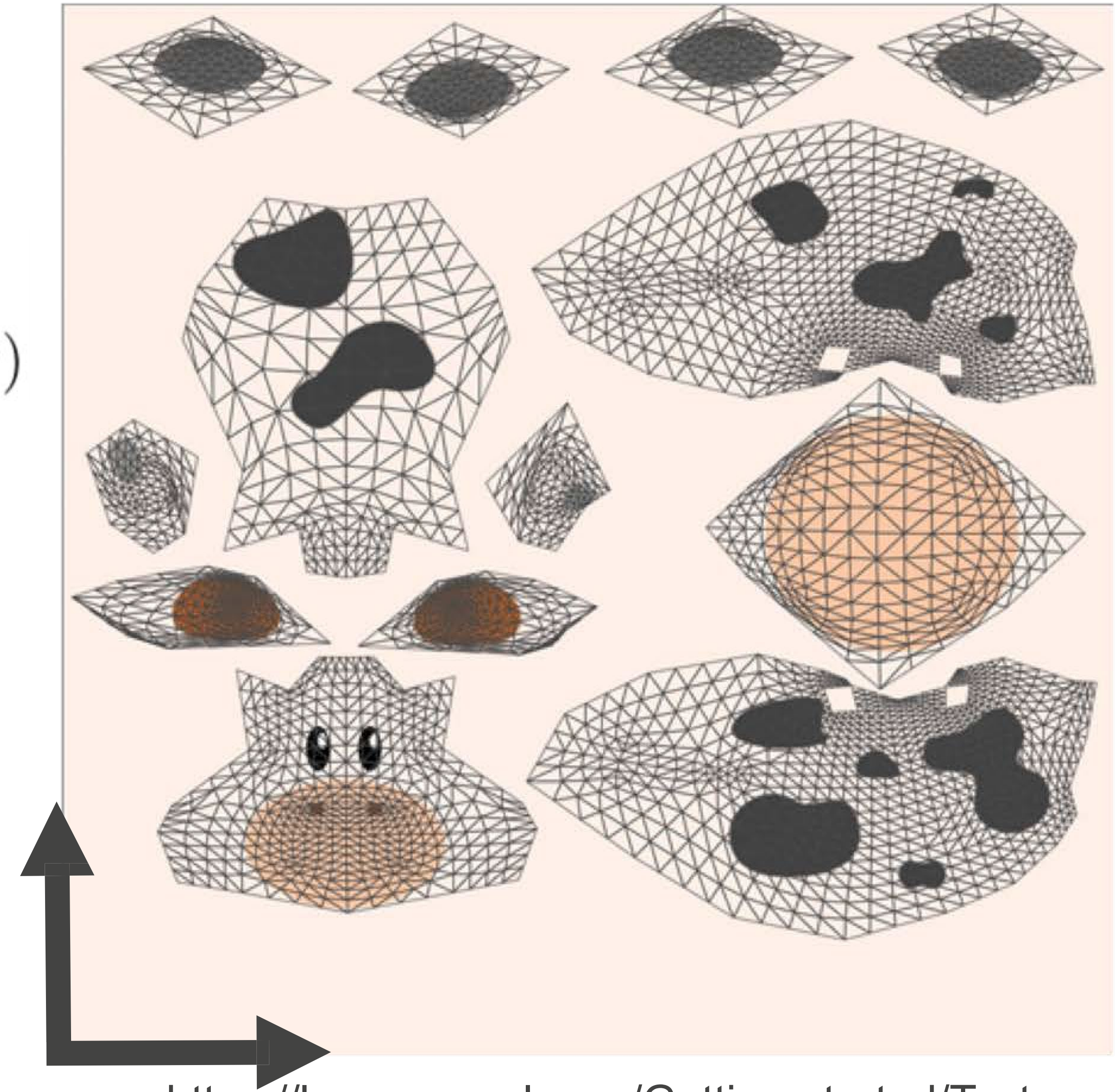
Depth culling, masking, etc.

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

Texture coordinates

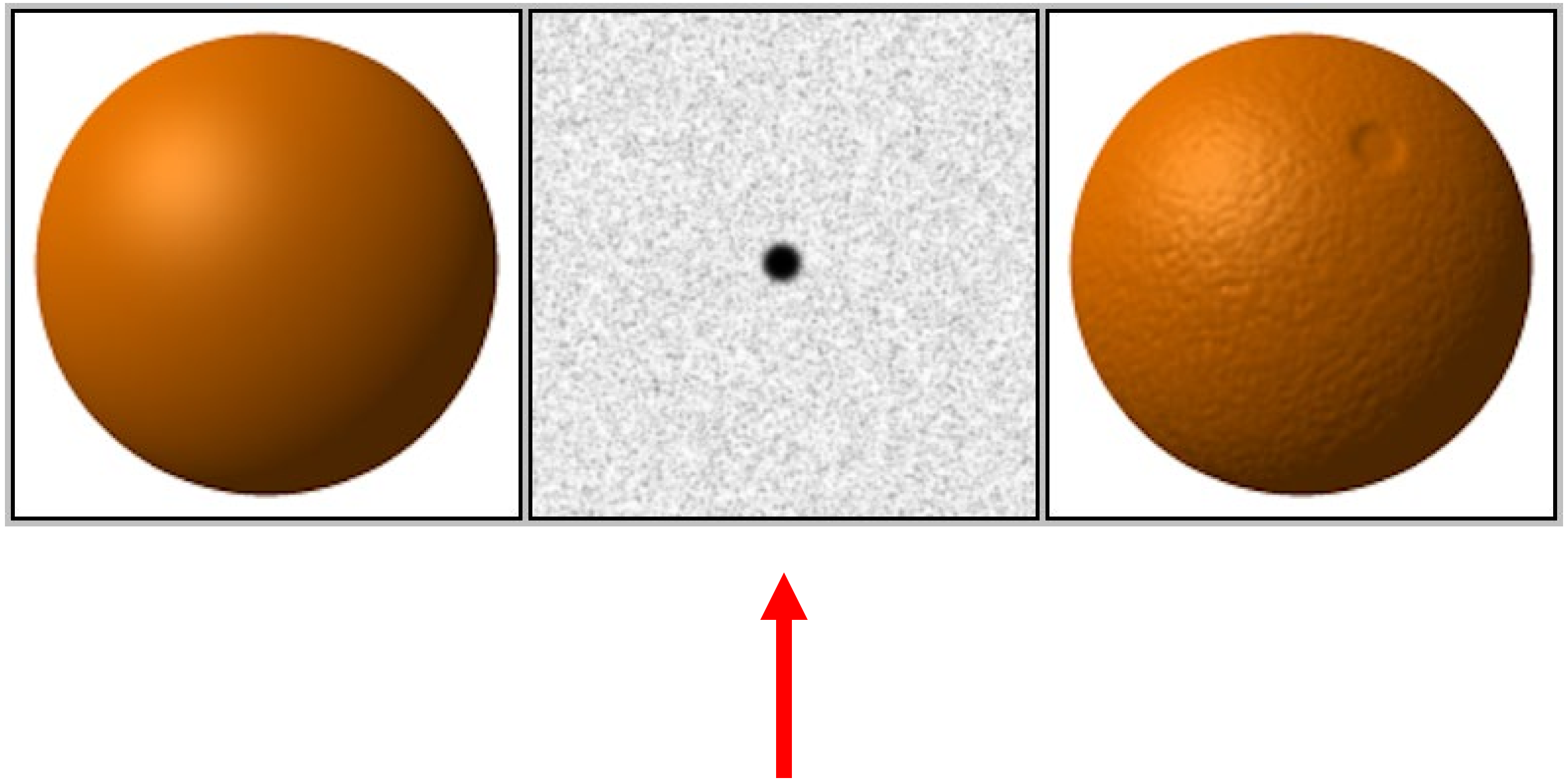


$$\phi : S \to T$$
$$: (x, y, z) \mapsto (u, v)$$



https://learnopengl.com/Getting-started/Textures

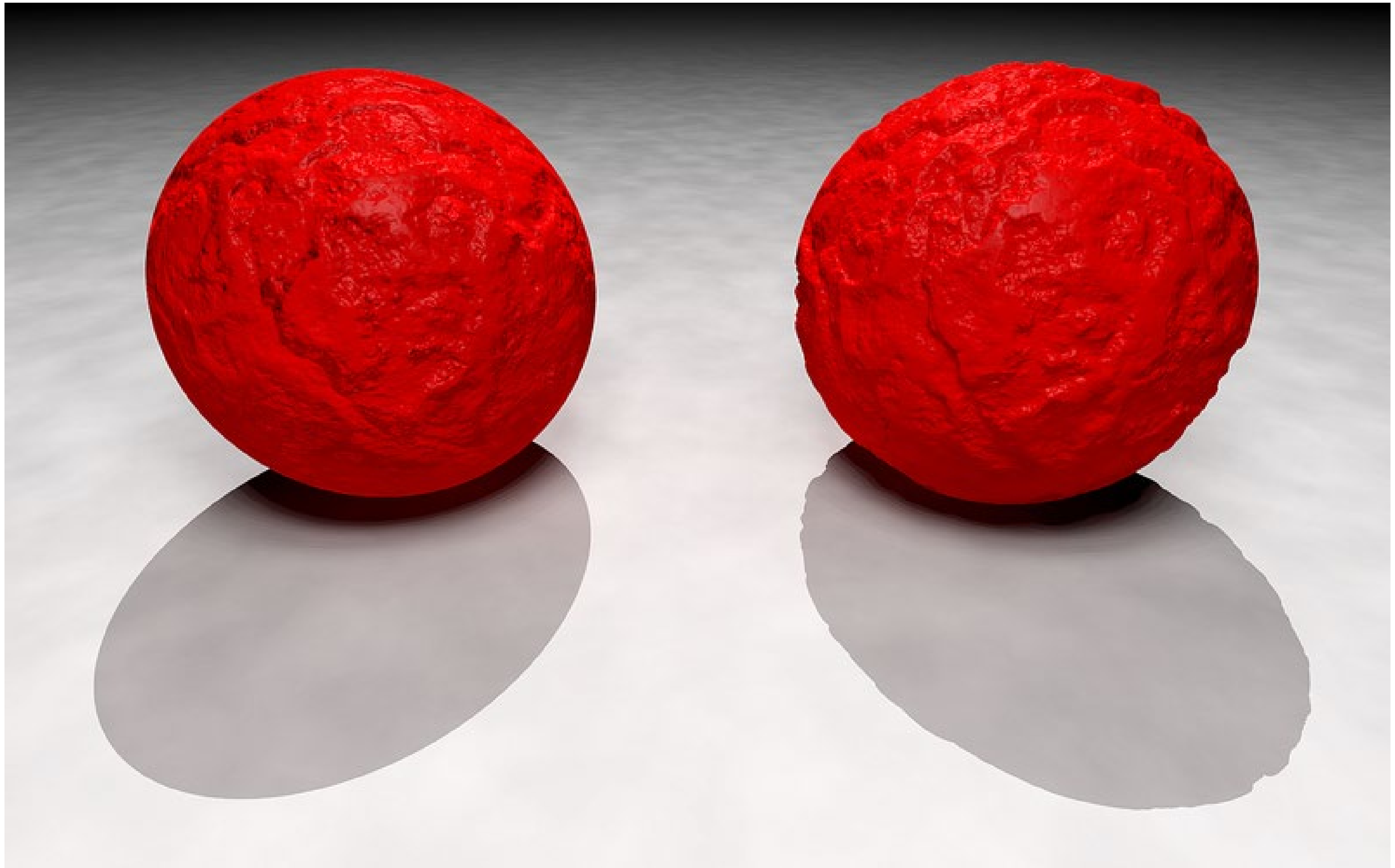
What is Bump Mapping?



said “bump map”

Fun fact! Invented by James Blinn in 1978.

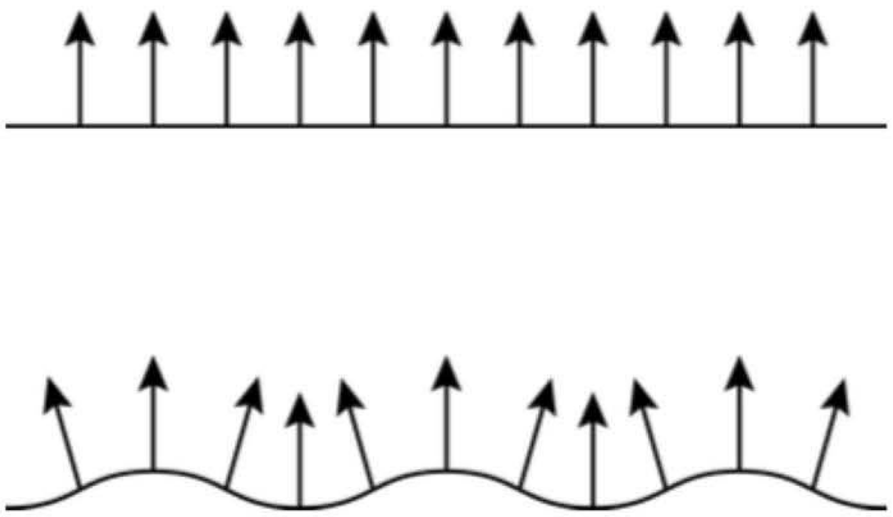
Bump Mapping Effects

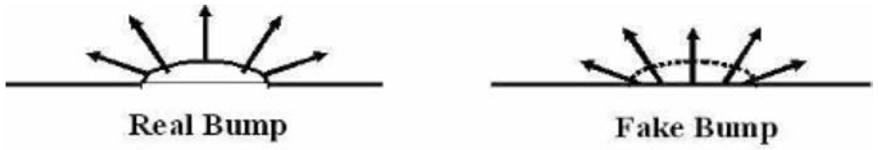


Bump Mapping Effects

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.





Bump Mapping Algorithm

Want to move each point on the surface to a new position.

$$\tilde{\mathbf{p}}(\mathbf{p}) = \mathbf{p} + h(\mathbf{p})\hat{\mathbf{n}}(\mathbf{p})$$

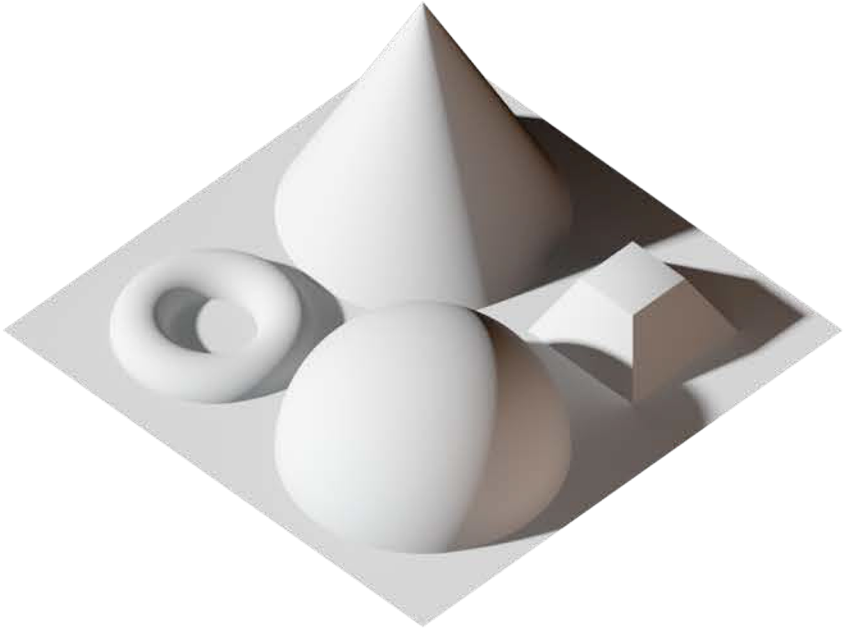
bump position function

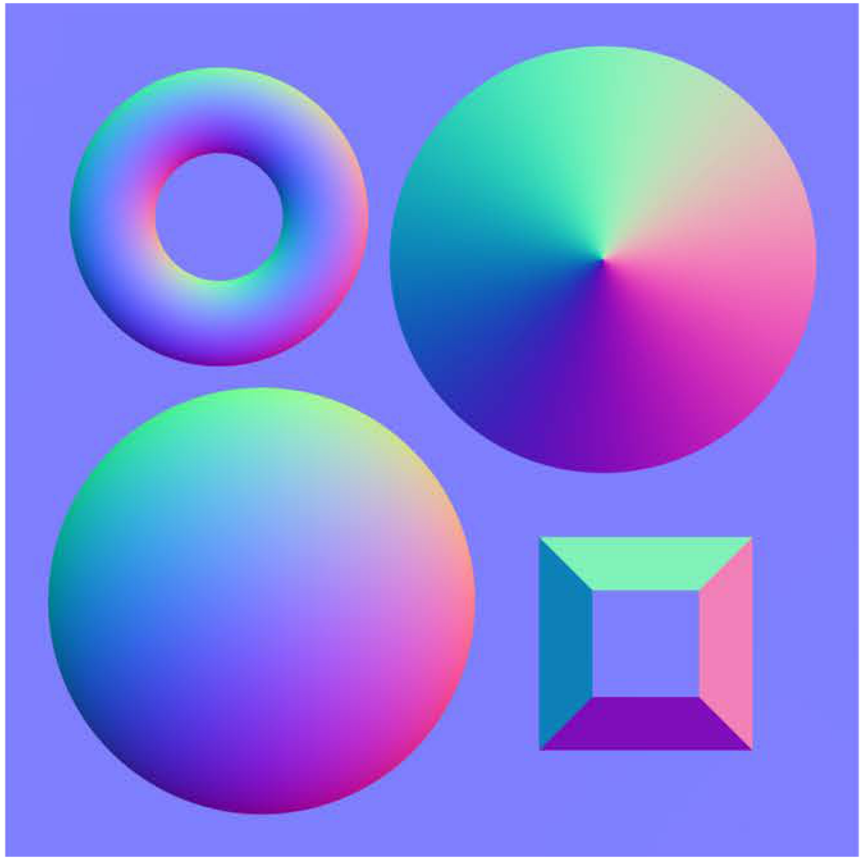
$$h : \mathbb{R}^3 \rightarrow \mathbb{R}$$

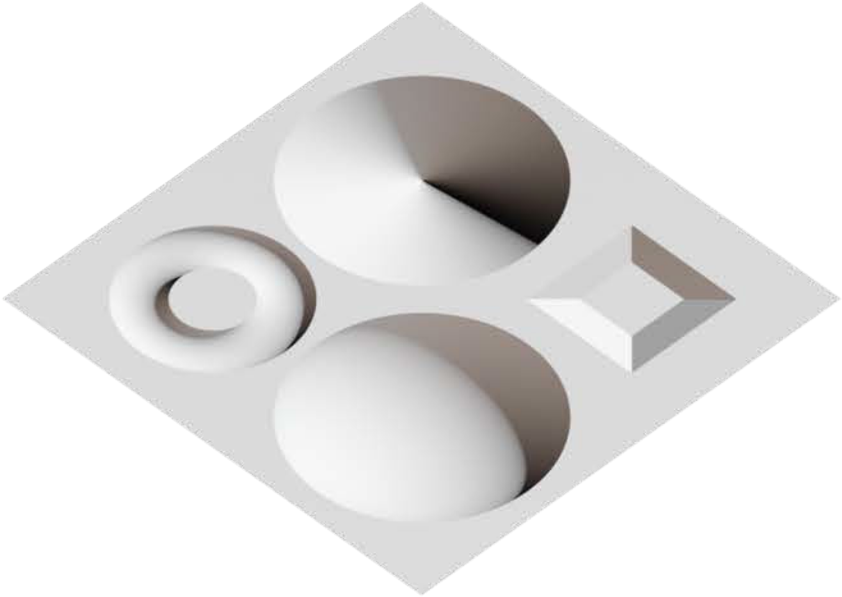
bump height function

https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps
https://en.wikipedia.org/wiki/Bump_mapping

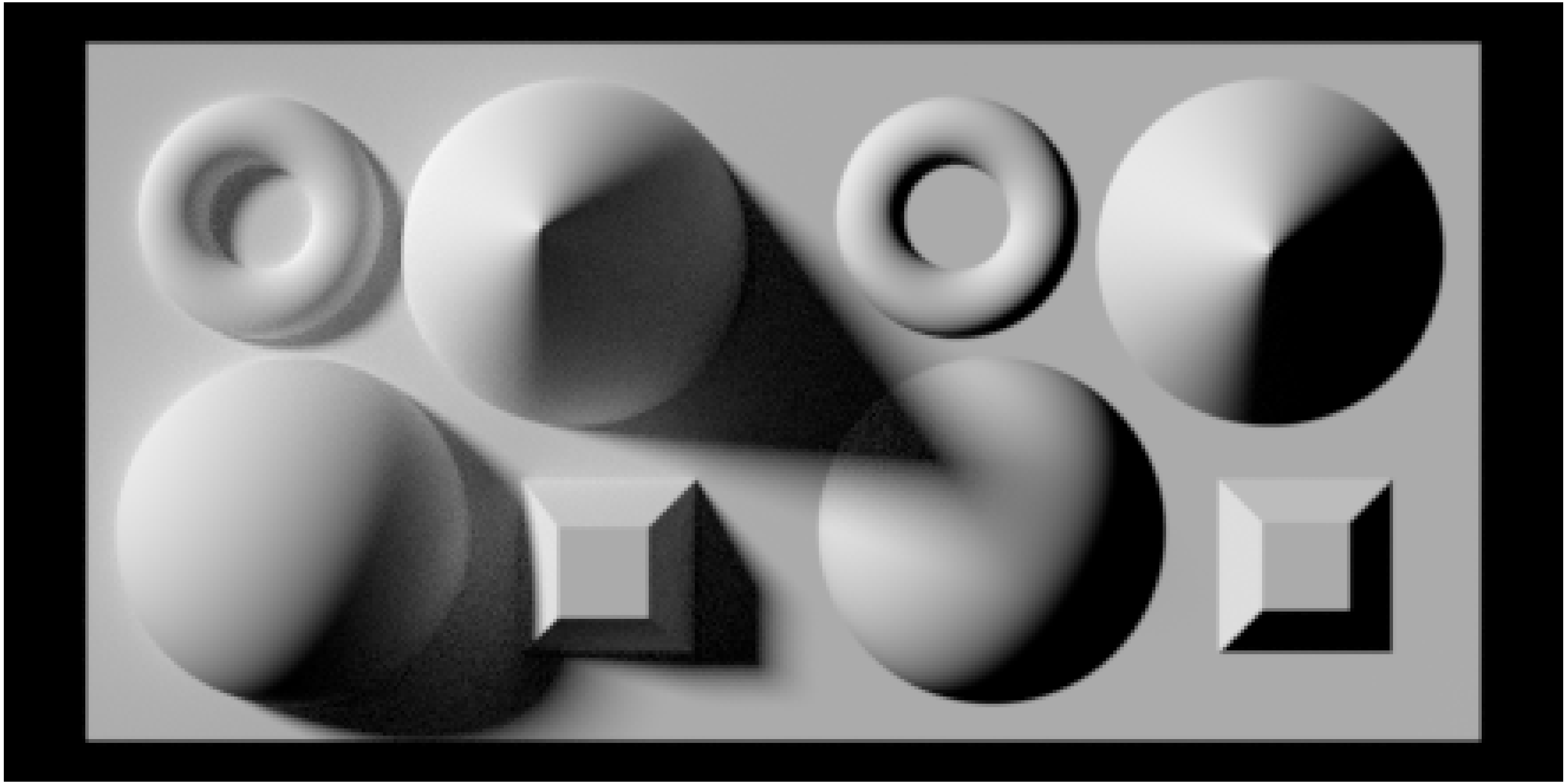
Normal Mapping







Normal Mapping



Bump Mapping Algorithm

Want to move each point on the surface to a new position.

$$\tilde{\mathbf{p}}(\mathbf{p}) = \mathbf{p} + h(\mathbf{p})\hat{\mathbf{n}}(\mathbf{p})$$ bump position function

$$h : \mathbb{R}^3 \to \mathbb{R}$$ bump height function

Calculate the **perceived n** using the **finite difference** method.

$$\tilde{\mathbf{n}} = \frac{\partial\, \mathbf{p}}{\partial\, \mathbf{T}} \times \frac{\partial\, \mathbf{p}}{\partial\, \mathbf{B}} \approx \left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon\mathbf{T}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right) \times \left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon\mathbf{B}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right)$$

https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps
https://en.wikipedia.org/wiki/Bump_mapping

Bump Mapping Algorithm

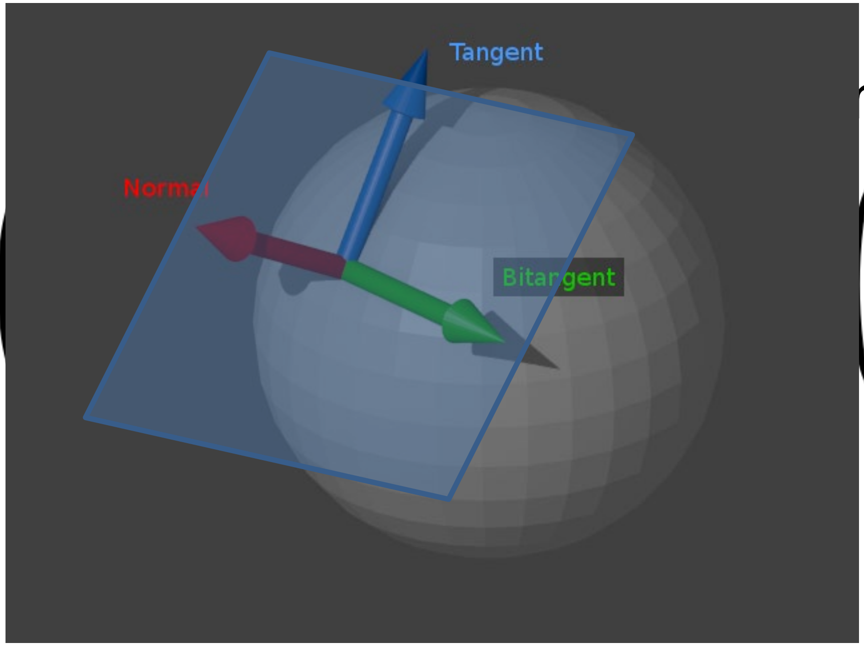
Want to move each point on the surface to a new position.

$$\tilde{\mathbf{p}}(\mathbf{p}) = \mathbf{p} + h(\mathbf{p})\hat{\mathbf{n}}(\mathbf{p})$$ bump position function

$$h : \mathbb{R}^3 \rightarrow \mathbb{R}$$ bump height function

Calculate the **perceived** method.

$$\tilde{\mathbf{n}} = \frac{\partial\, \mathbf{p}}{\partial\, \mathbf{T}} \times \frac{\partial\, \mathbf{p}}{\partial\, \mathbf{B}} \approx \Bigg($$



$$\left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon \mathbf{B}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right)$$

https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps
https://en.wikipedia.org/wiki/Bump_mapping

Bump Mapping Algorithm

Want to move each point on the surface to a new position.

$$\tilde{\mathbf{p}}(\mathbf{p}) = \mathbf{p} + h(\mathbf{p})\hat{\mathbf{n}}(\mathbf{p})$$

bump position function

$$h : \mathbb{R}^3 \to \mathbb{R}$$

bump height function

Calculate the **perceived n** using the **finite difference** method.

$$\tilde{\mathbf{n}} = \frac{\partial\ \mathbf{p}}{\partial\ \mathbf{T}} \times \frac{\partial\ \mathbf{p}}{\partial\ \mathbf{B}} \approx \left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon\mathbf{T}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right) \times \left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon\mathbf{B}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right)$$

https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps
https://en.wikipedia.org/wiki/Bump_mapping

Bump Mapping Algorithm

Want to move each point on the surface to a new position.

$$\tilde{\mathbf{p}}(\mathbf{p}) = \mathbf{p} + h(\mathbf{p})\hat{\mathbf{n}}(\mathbf{p})$$ bump position function

$$h : \mathbb{R}^3 \rightarrow \mathbb{R}$$ bump height function

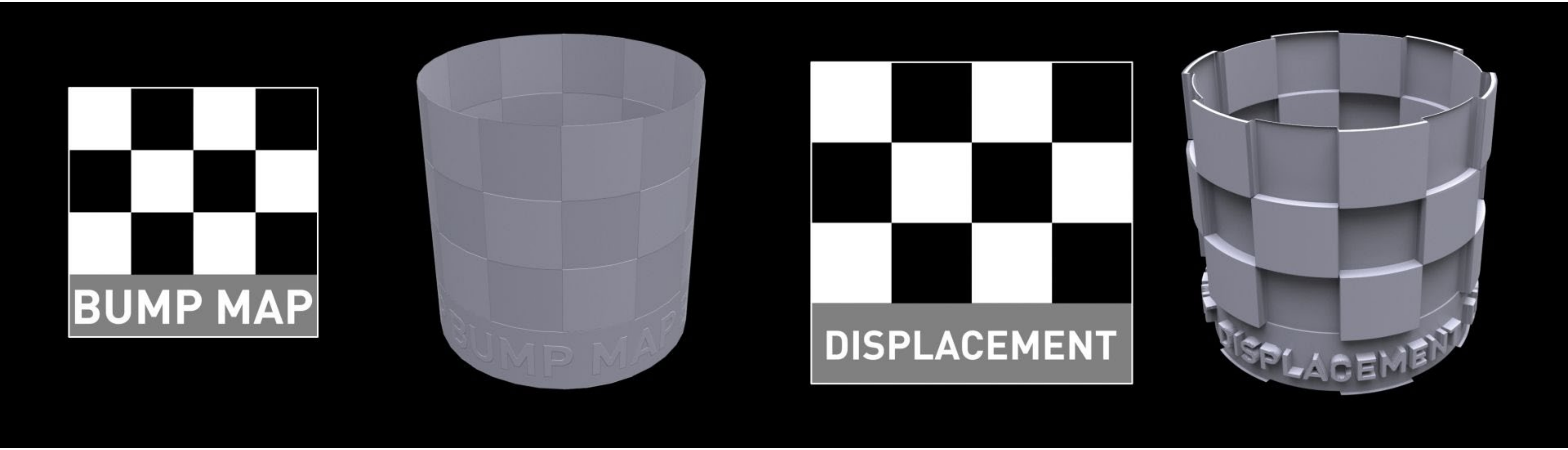
Calculate the **perceived n** using the **finite difference** method.

$$\tilde{\mathbf{n}} = \frac{\partial\, \mathbf{p}}{\partial\, \mathbf{T}} \times \frac{\partial\, \mathbf{p}}{\partial\, \mathbf{B}} \approx \left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon\mathbf{T}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right) \times \left(\frac{\tilde{\mathbf{p}}(\mathbf{p} + \epsilon\mathbf{B}) - \tilde{\mathbf{p}}(\mathbf{p})}{\epsilon}\right)$$

Normalize it, then use the new normal in the shading function.

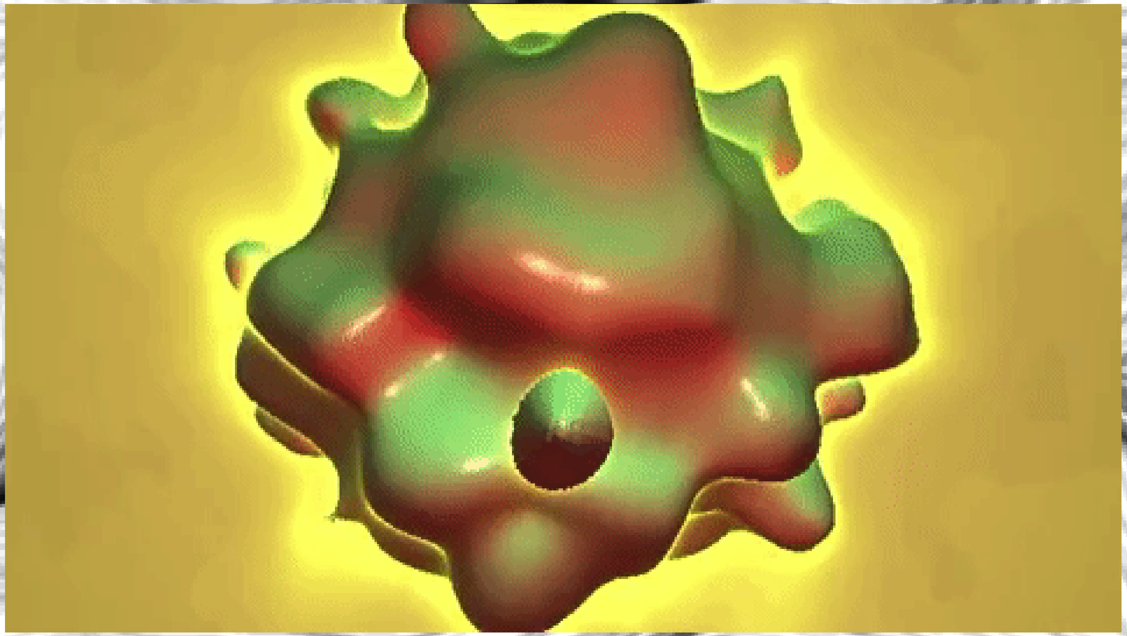
https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps
https://en.wikipedia.org/wiki/Bump_mapping

Normal Mapping vs. Displacement Mapping

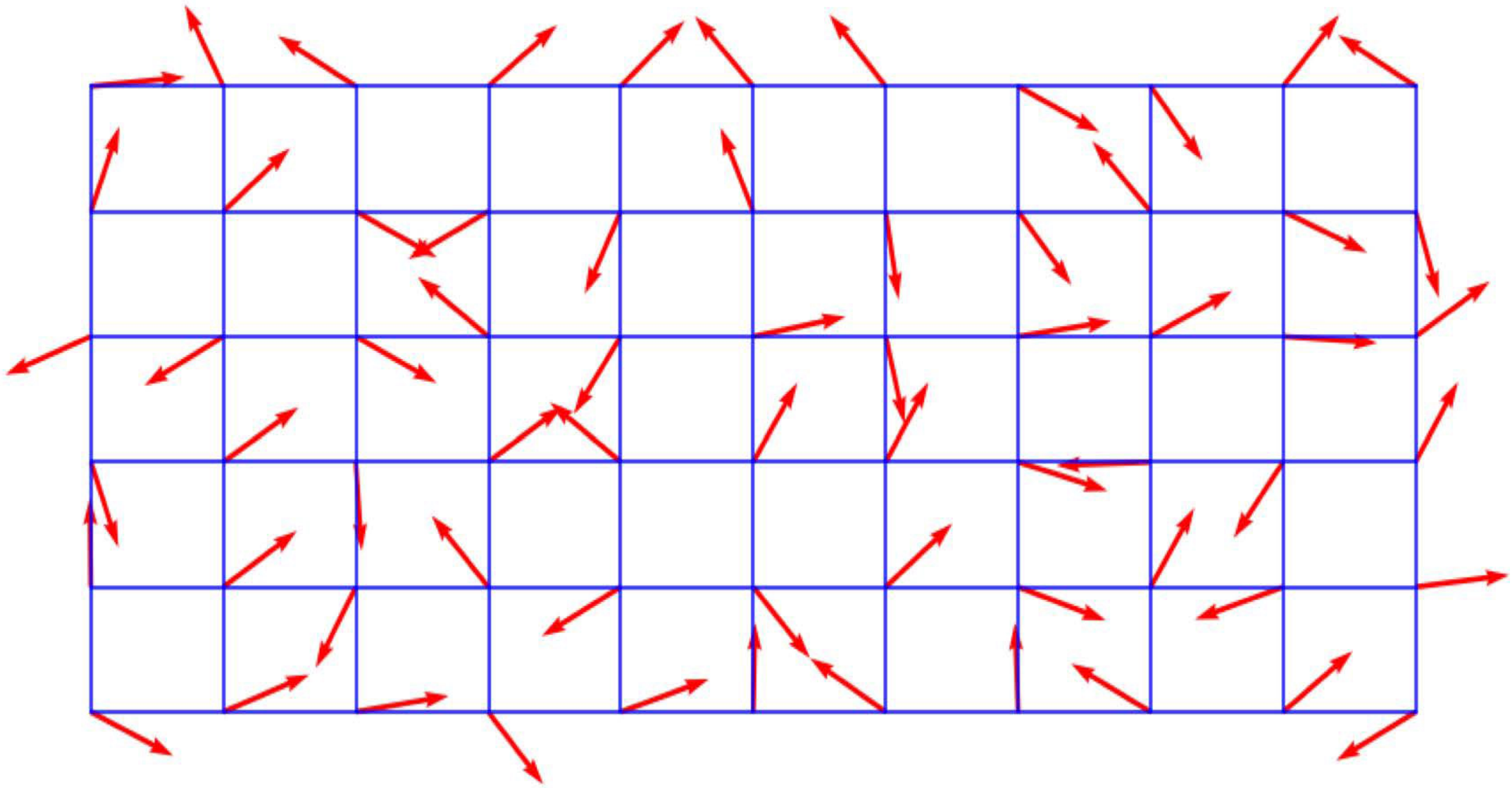


https://www.creativebloq.com/features/a-beginners-guide-to-displacement-and-bump-map

Perlin Noise

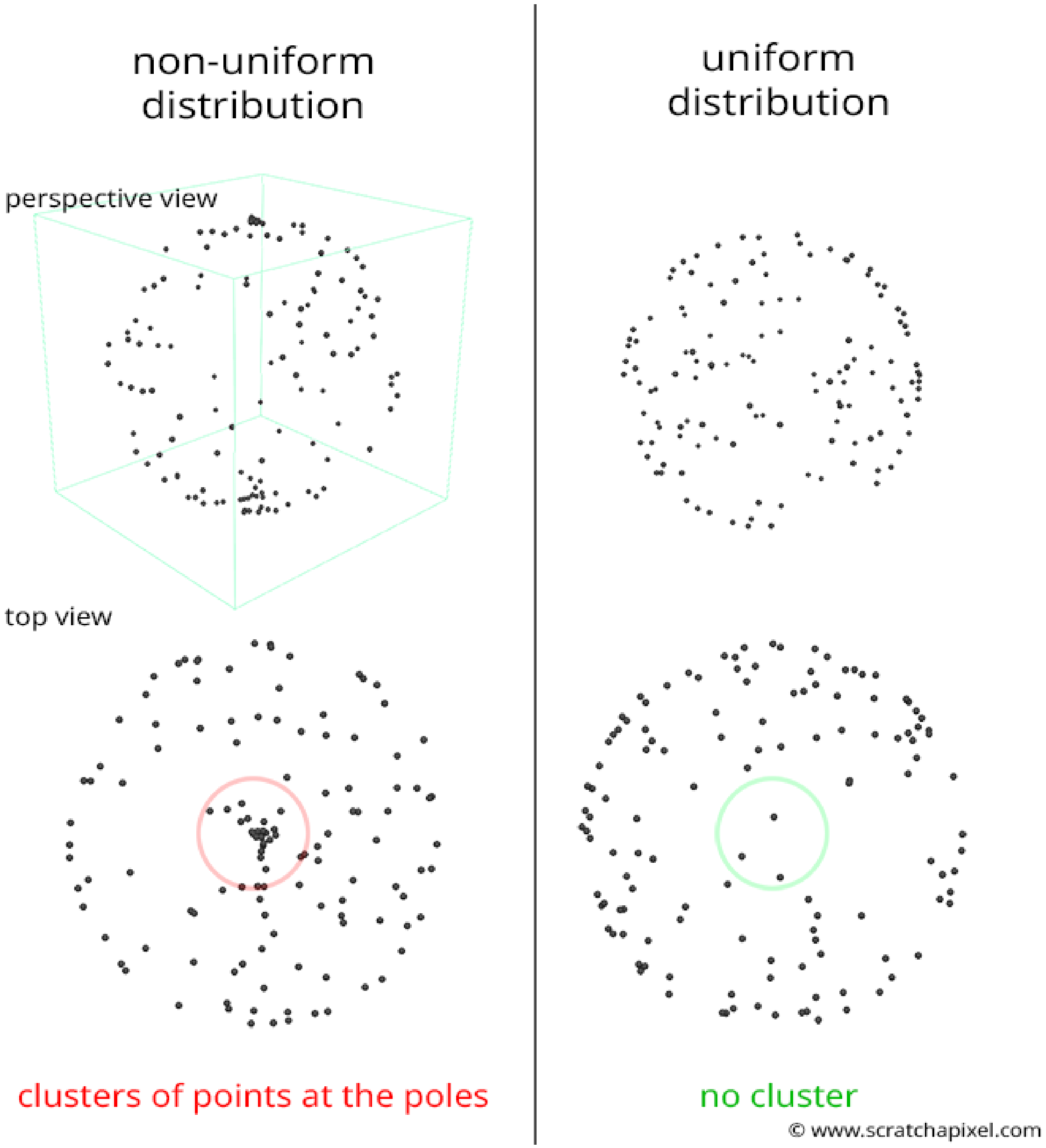


https://www.shadertoy.com/view/Msf3l



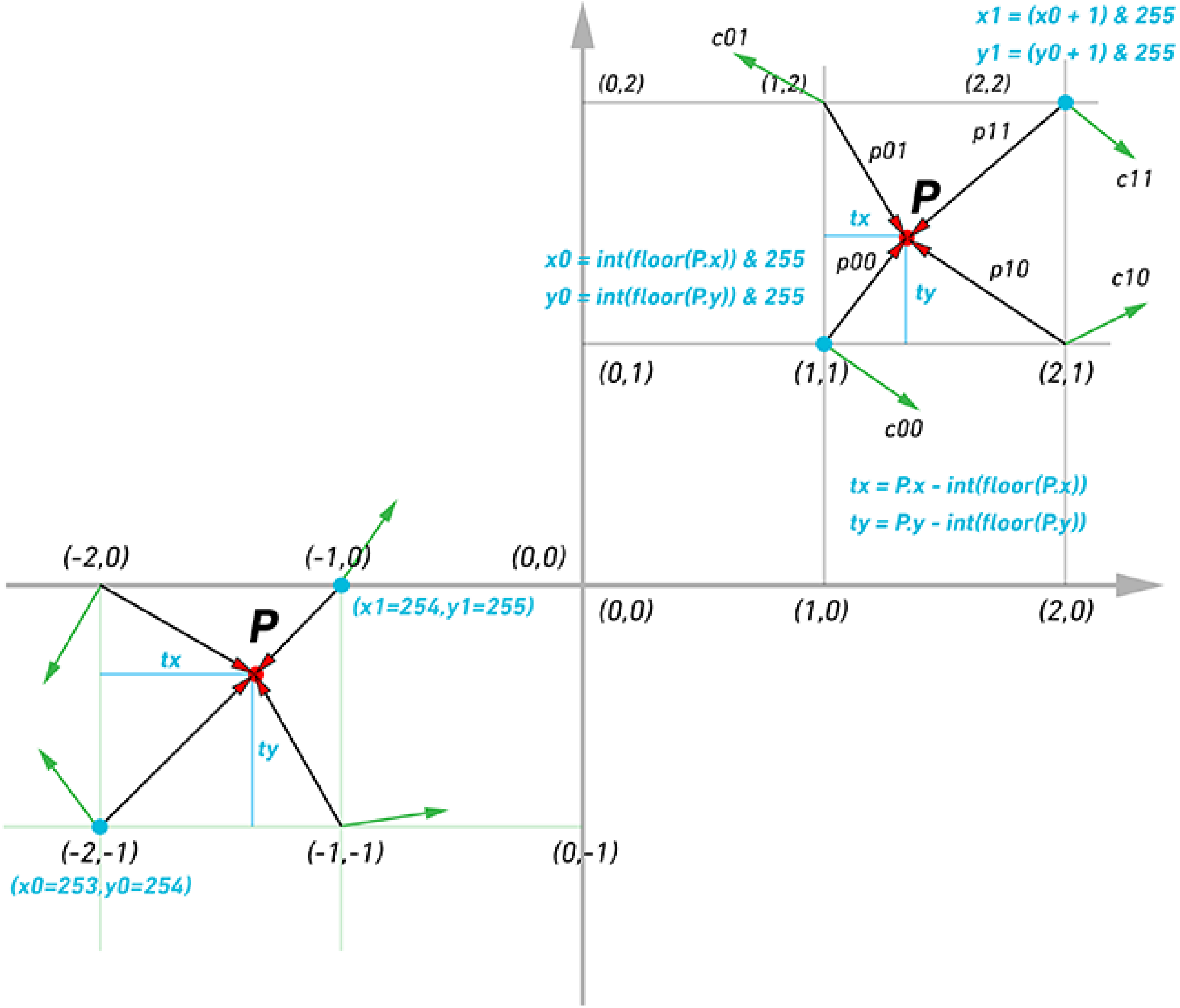
https://en.wikipedia.org/wiki/Perlin_noise

https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise



See details at the link!

https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise

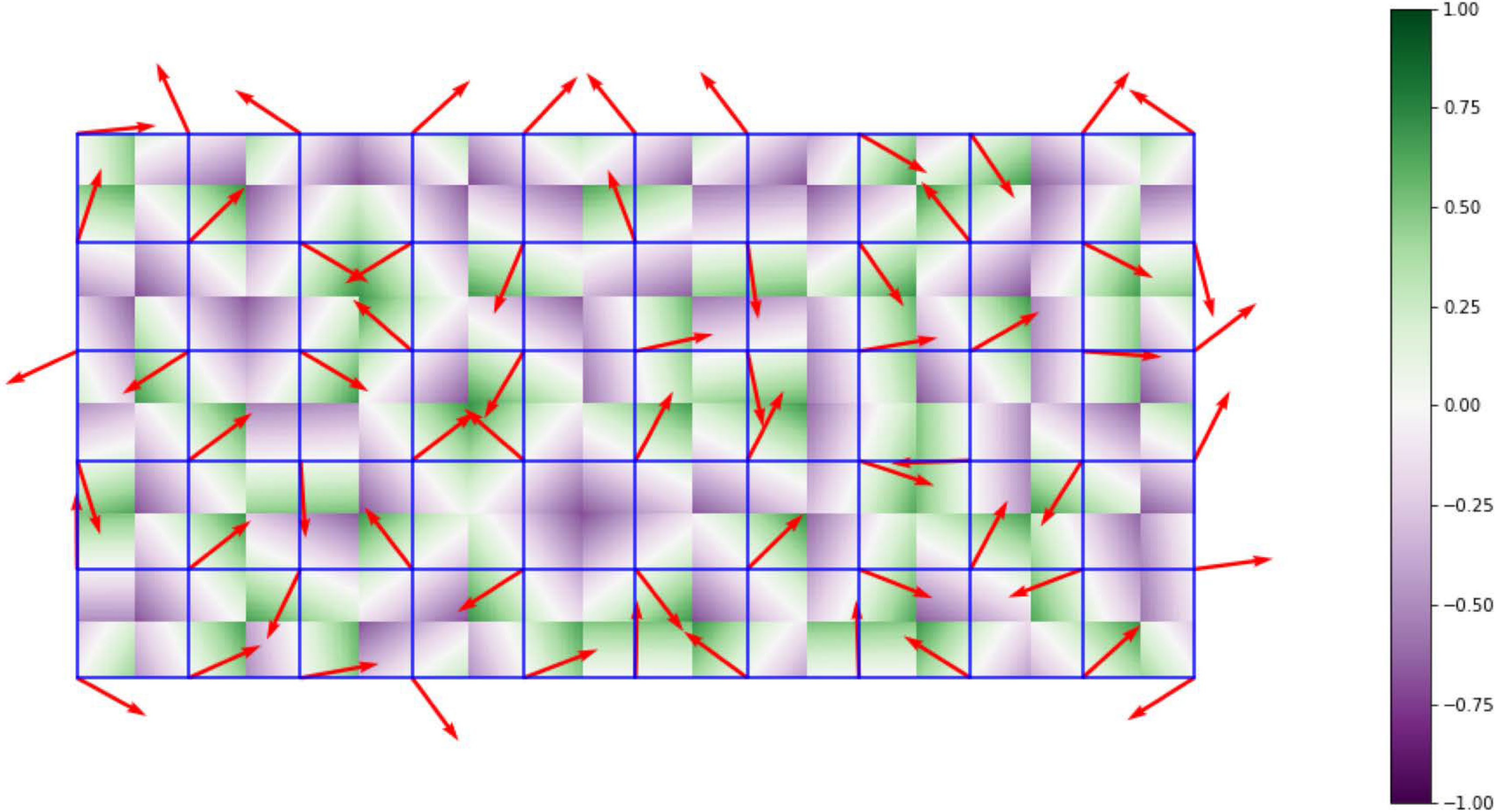


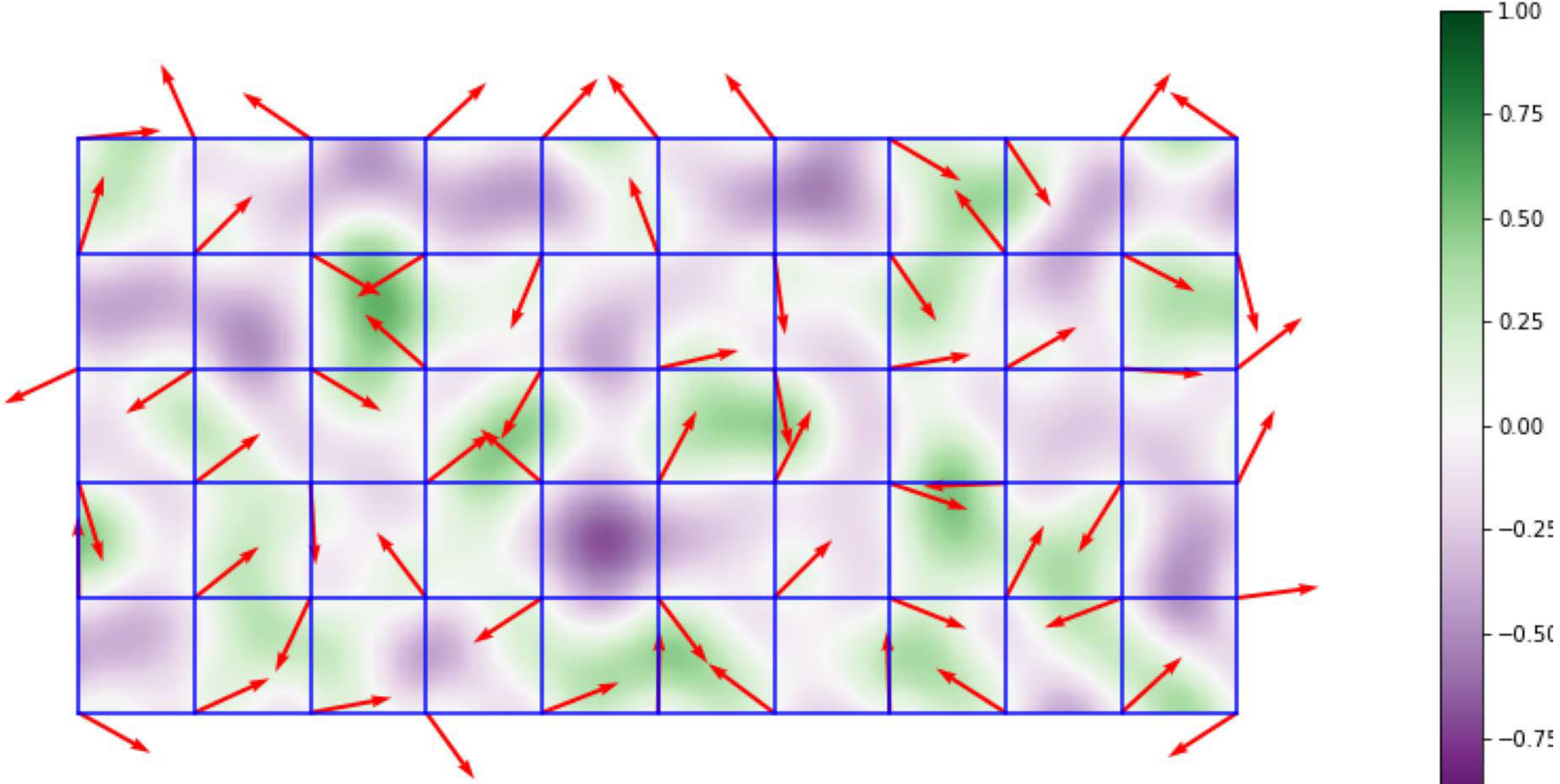
Compute directions between the position of each corner of the cell to the point P at the position of which we wish to evaluate the noise function.

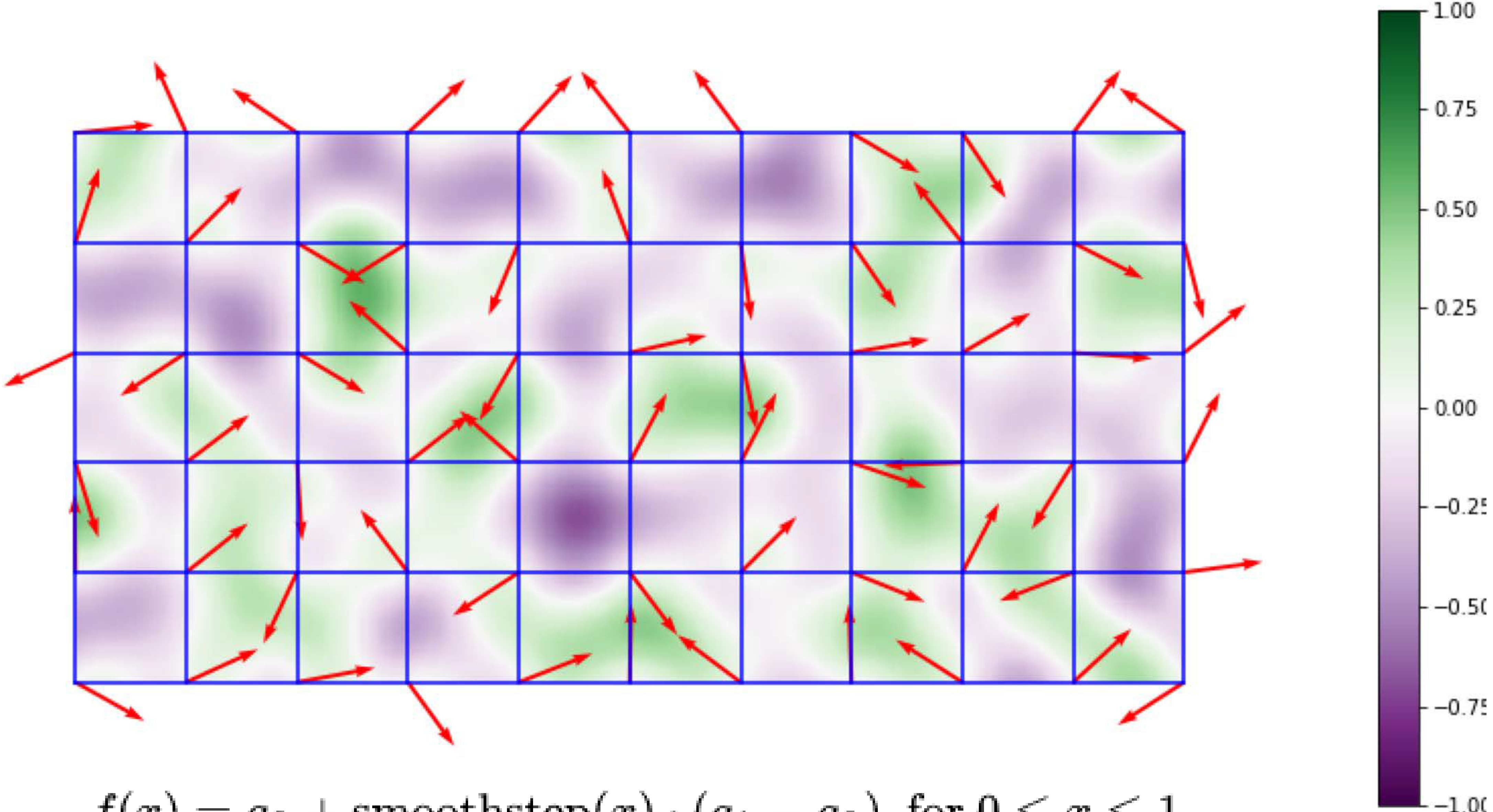
Take dot product between the gradient at the corner of a cell and the vector from that corner to P

https://en.wikipedia.org/wiki/Perlin_noise

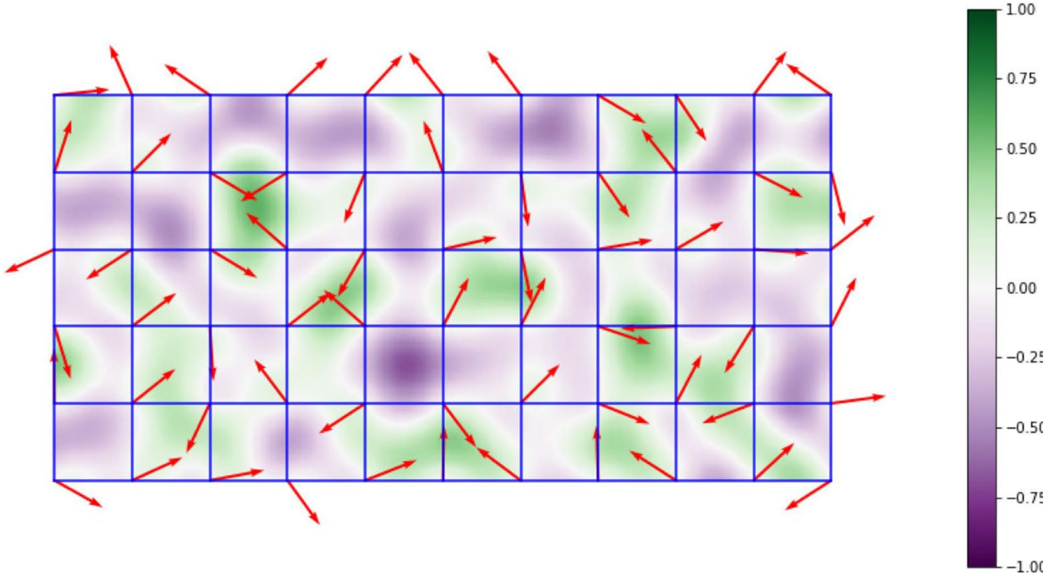
https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise


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

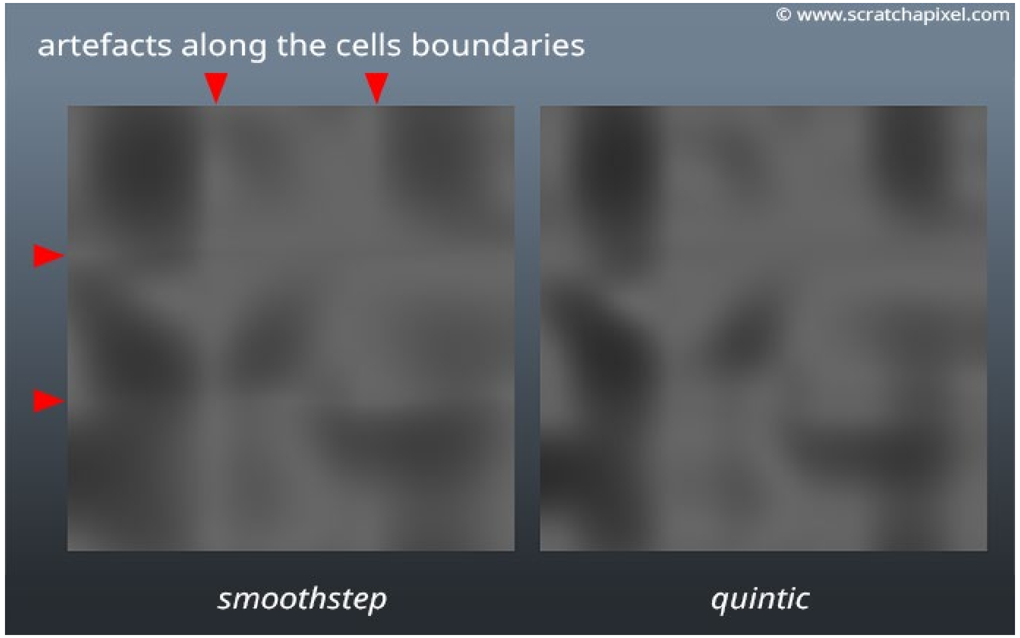

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



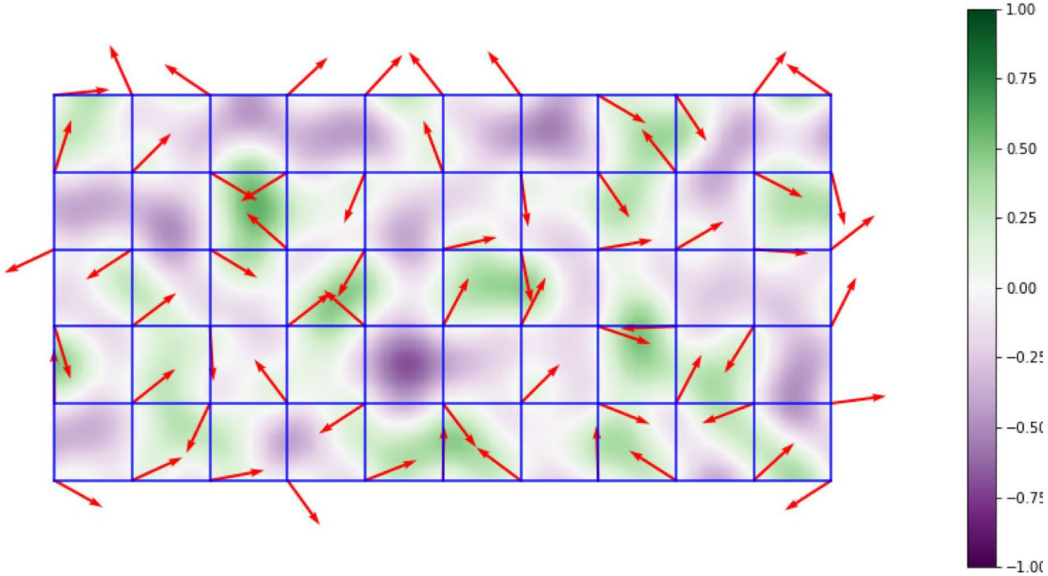
$$f(x) = a_0 + \mathrm{smoothstep}(x) \cdot (a_1 - a_0) \;\; \text{for } 0 \leq x \leq 1$$



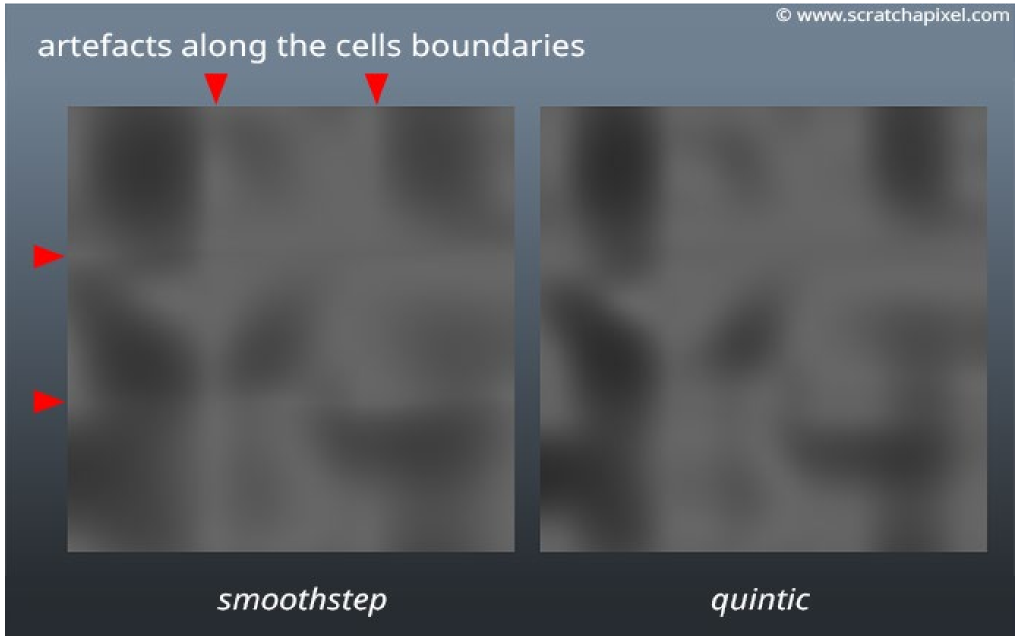
$$\text{smoothstep}(x) = S_1(x) = \begin{cases} 0 & x \le 0 \\ 3x^2 - 2x^3 & 0 \le x \le 1 \\ 1 & 1 \le x \end{cases}$$



https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/improved-perlin-noise



$$\mathrm{smoothstep}(x) = S_1(x) = \begin{cases} 0 & x \le 0 \\ 3x^2 - 2x^3 & 0 \le x \le 1 \\ 1 & 1 \le x \end{cases}$$



$$6t^5 - 15t^4 + 10t^3 .$$

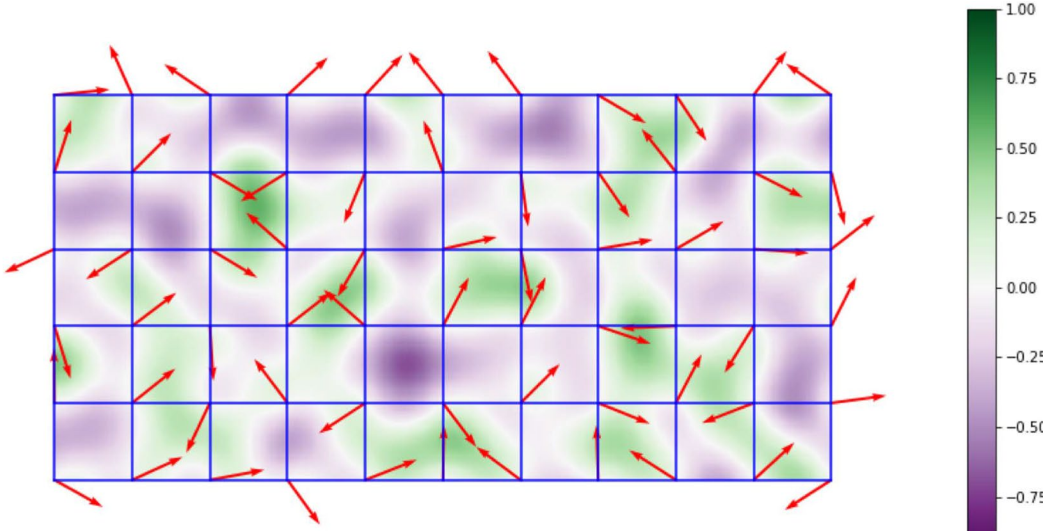
Its first order derivative is:

$$30t^4 - 60t^3 + 30t^2 .$$

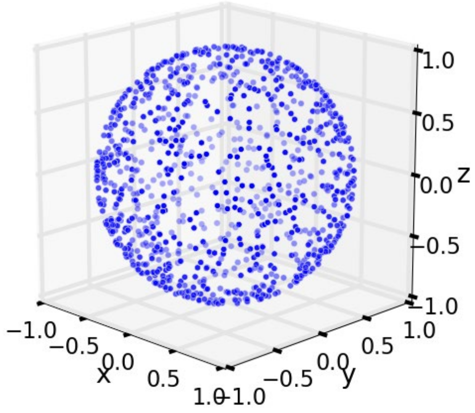
And its second order derivative is:

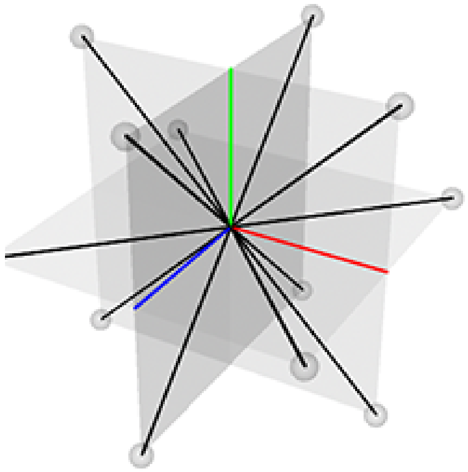
$$120t^3 - 180t^2 + 60t.$$

https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/improved-perlin-noise



$$\text{smoothstep}(x) = S_1(x) = \begin{cases} 0 & x \le 0 \\ 3x^2 - 2x^3 & 0 \le x \le 1 \\ 1 & 1 \le x \end{cases}$$





https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/improved-perlin-noise

Let’s go over the midterm