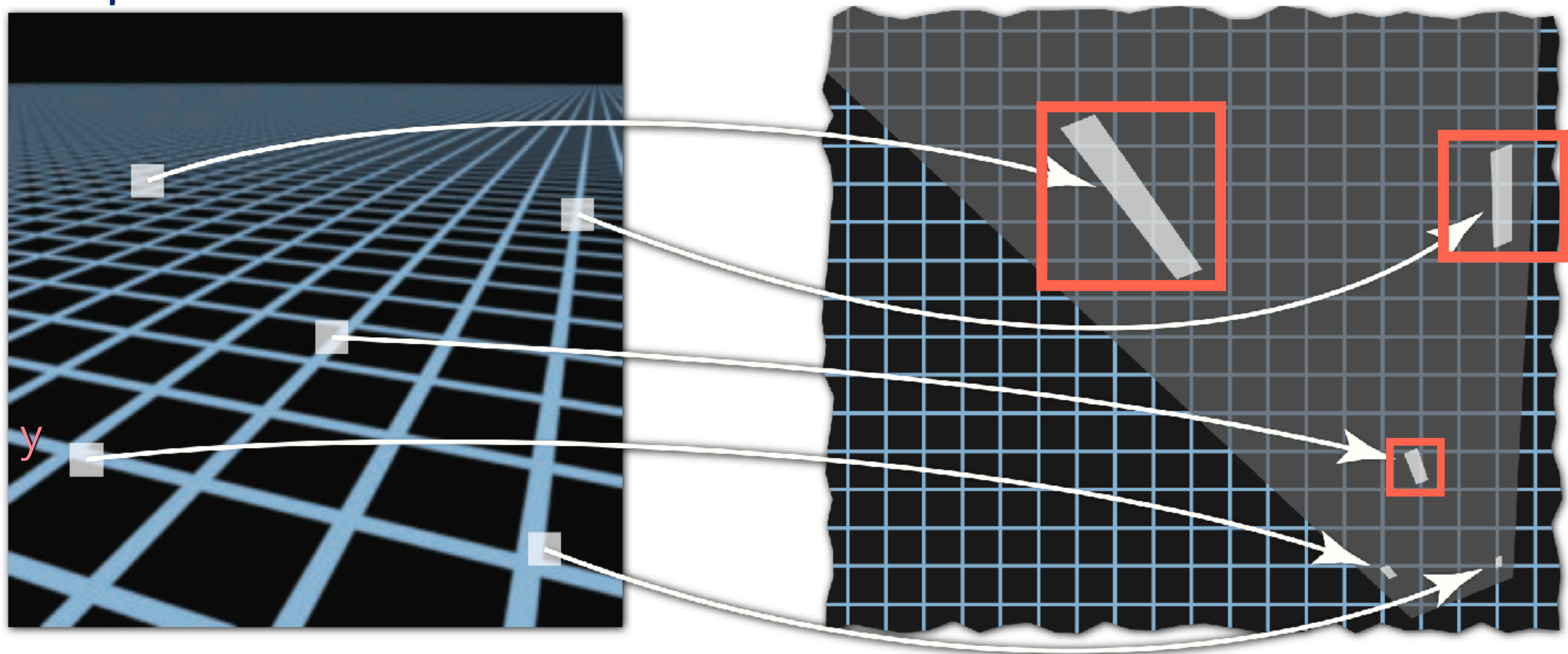


Pixel to Texel Mapping

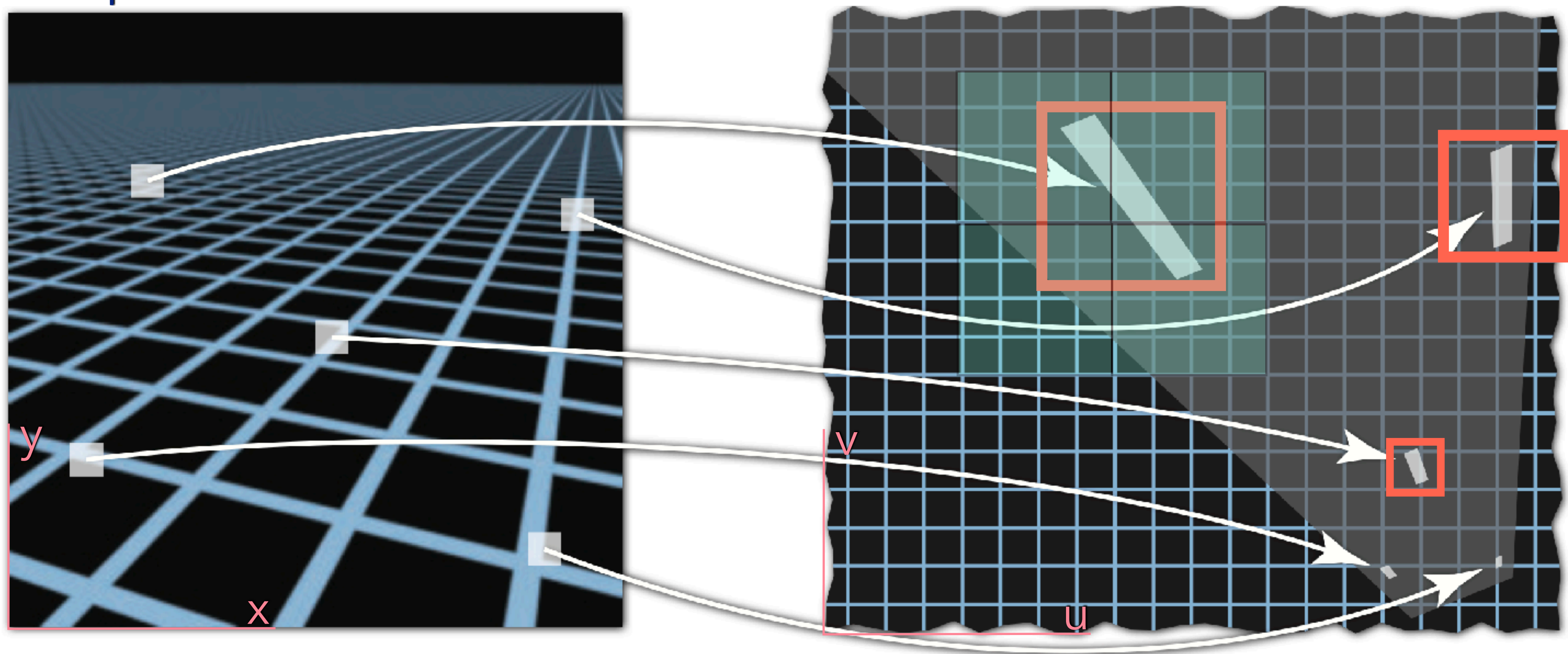


- One pixel per texel?



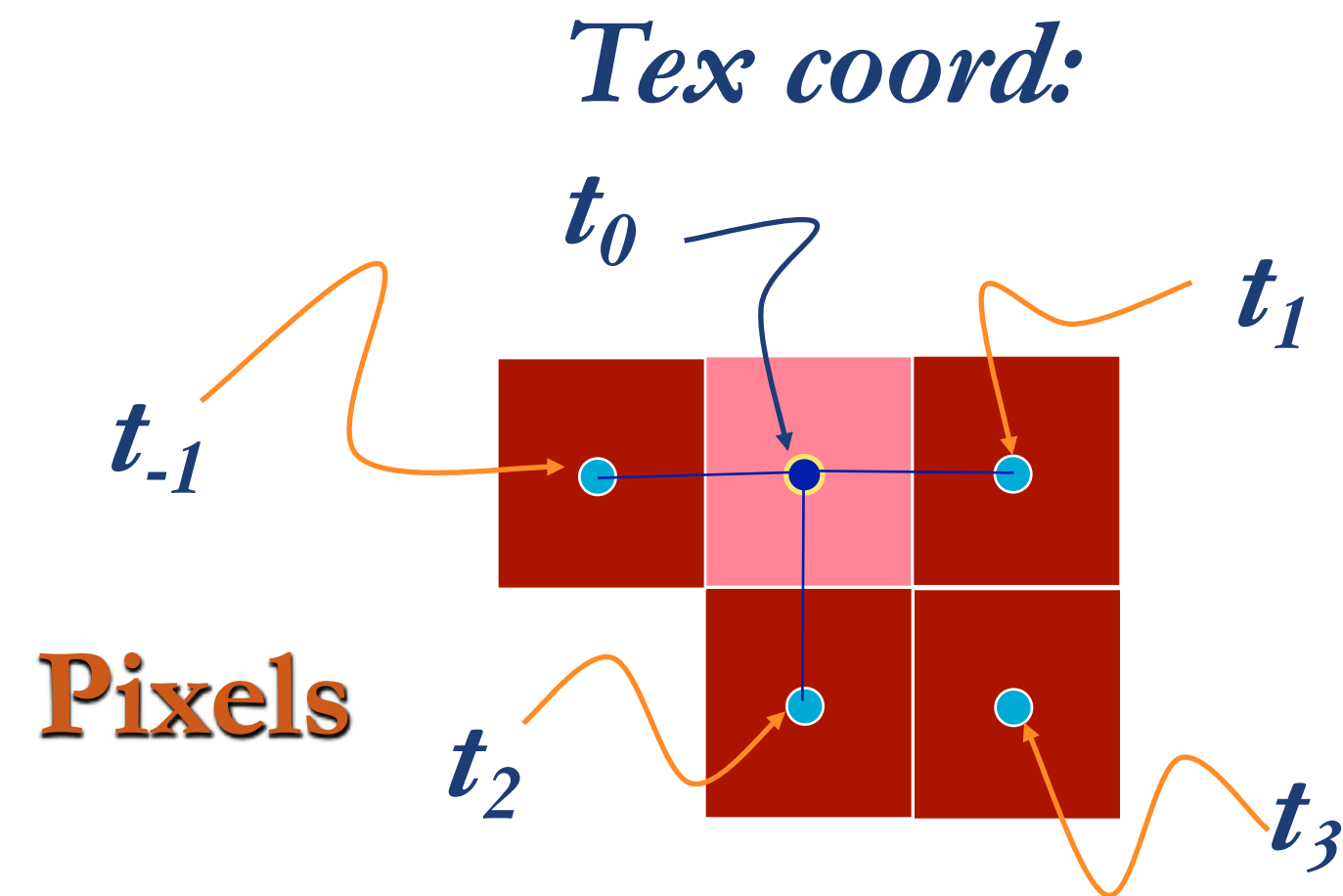
Pixel to Texel Mapping

- One pixel per texel? In which direction?





Derivative



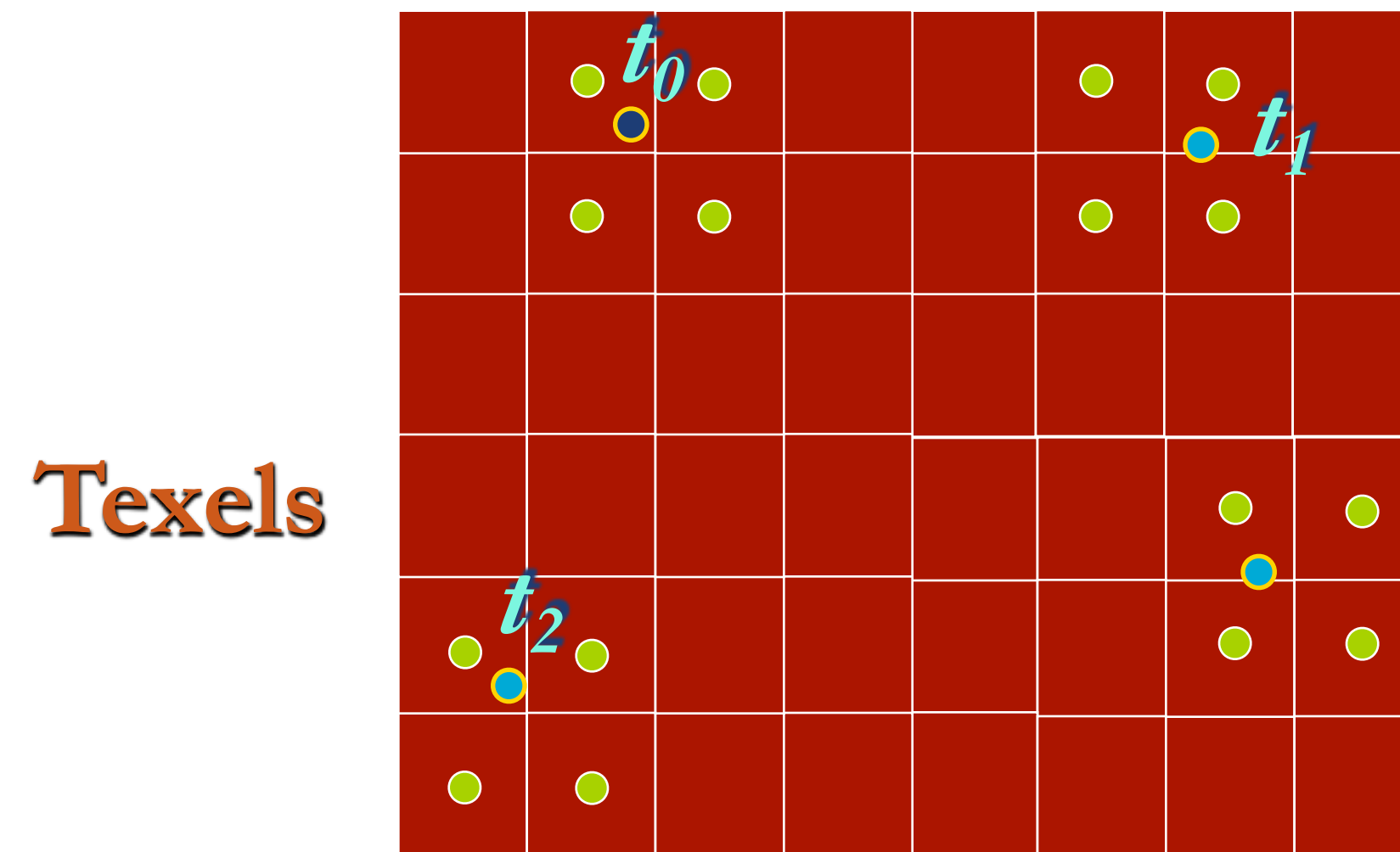
$$t_1 - t_0 = (\Delta u^x, \Delta v^x)$$

$$t_2 - t_0 = (\Delta u^y, \Delta v^y)$$

$$\Delta t = \max(\Delta u^x, \Delta u^y), \max(\Delta v^x, \Delta v^y)$$

$$\Delta u_{pixel} = \max(du/dx, du/dy)$$

$$\Delta v_{pixel} = \max(dv/dx, dv/dy)$$

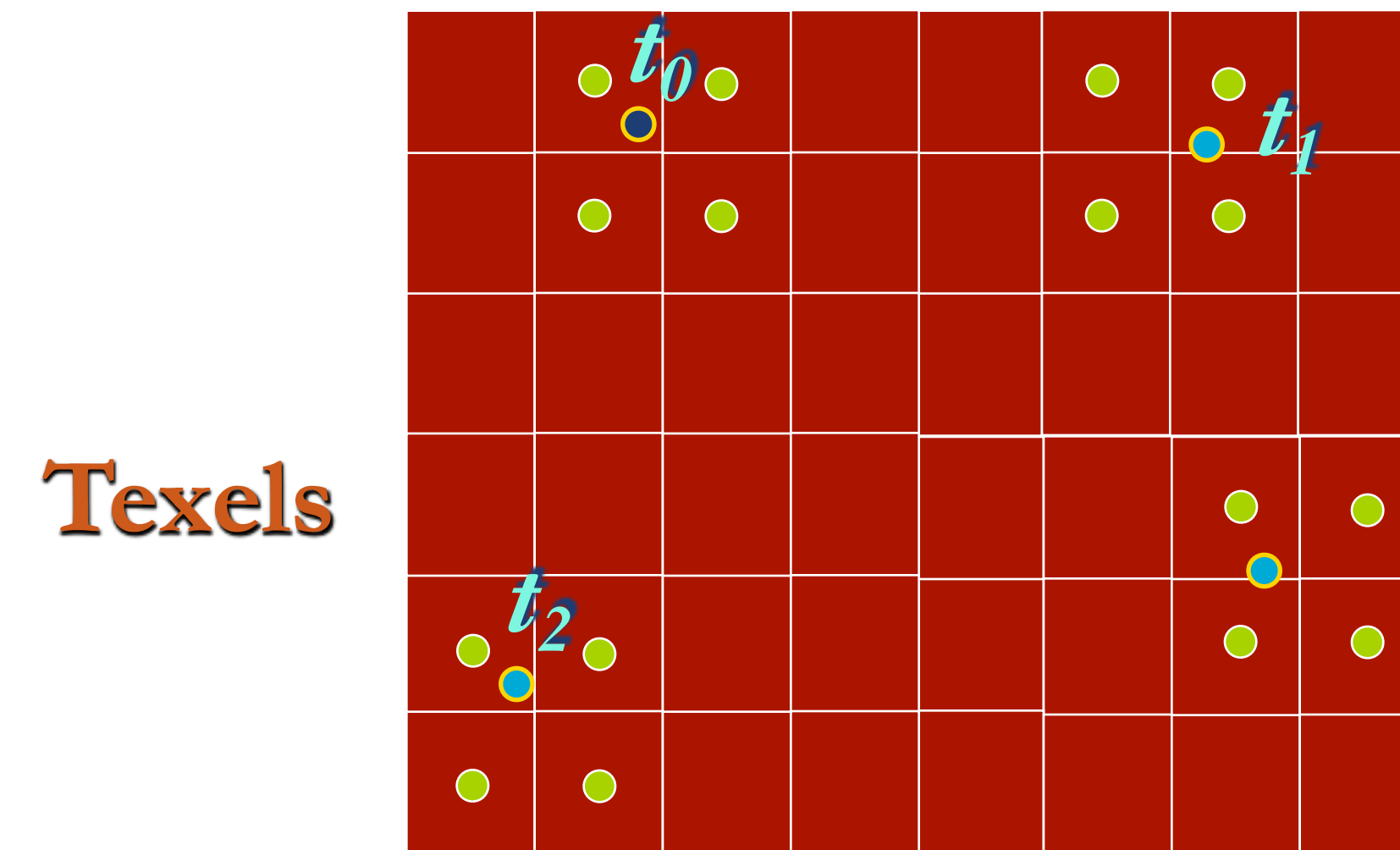
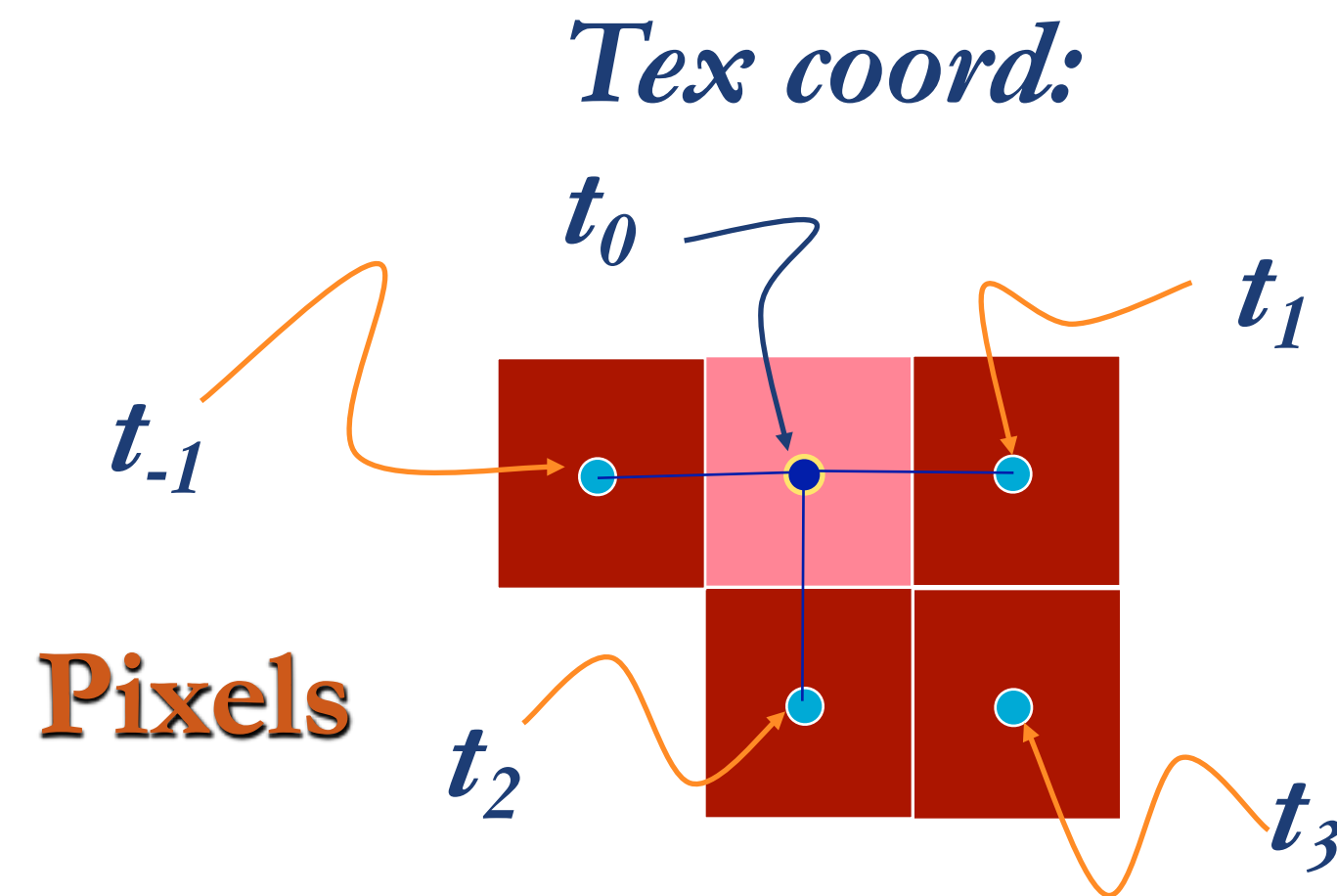


$$W \times H \text{ texture} \Rightarrow \text{texel size} = 1/W \times 1/H$$

Choose a level so that
 Δu and Δv are near the texel size



Derivative



$$t_1 - t_0 = (\Delta u^x, \Delta v^x)$$

$$t_2 - t_0 = (\Delta u^y, \Delta v^y)$$

Need neighbor's
texture coordinate?

Rasterizer computes.

$$\Delta t = \max(\Delta u^x, \Delta u^y), \max(\Delta v^x, \Delta v^y) \quad (\text{see } dFdx, dFdy)$$

$$\Delta u_{pixel} = \max(du/dx, du/dy)$$

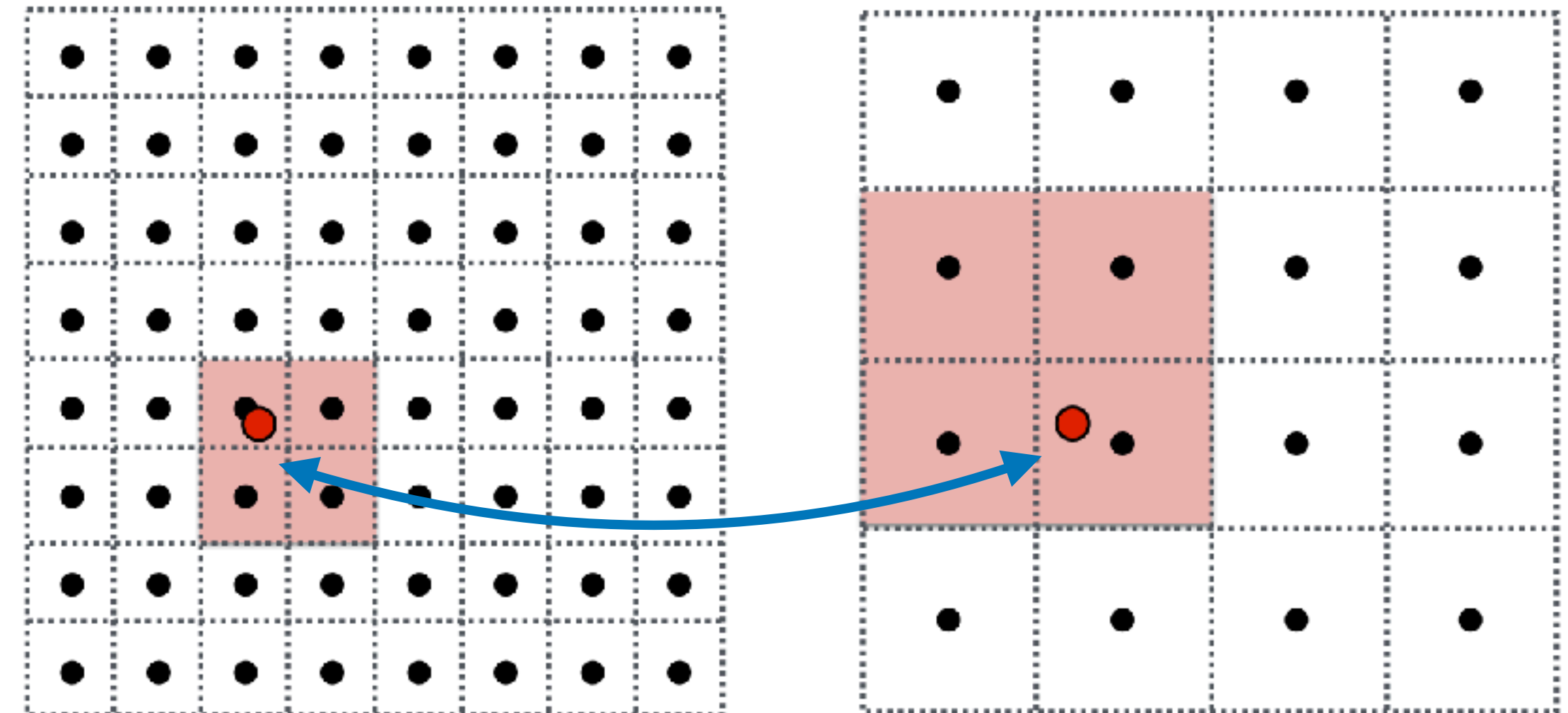
$$\Delta v_{pixel} = \max(dv/dx, dv/dy)$$

$$W \times H \text{ texture} \Rightarrow \text{texel size} = 1/W \times 1/H$$

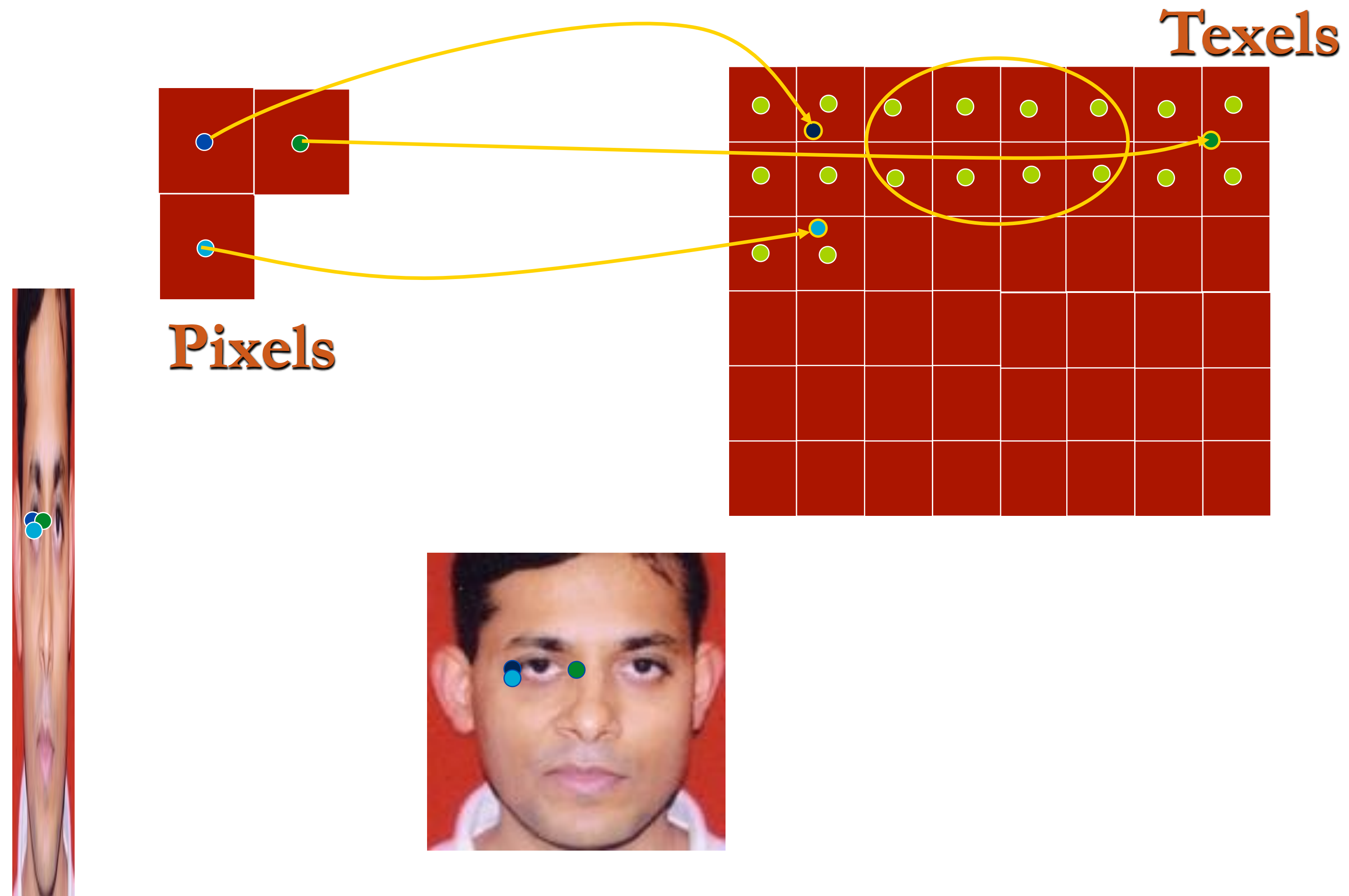
Choose a level so that
 Δu and Δv are near the texel size

Fractional LOD

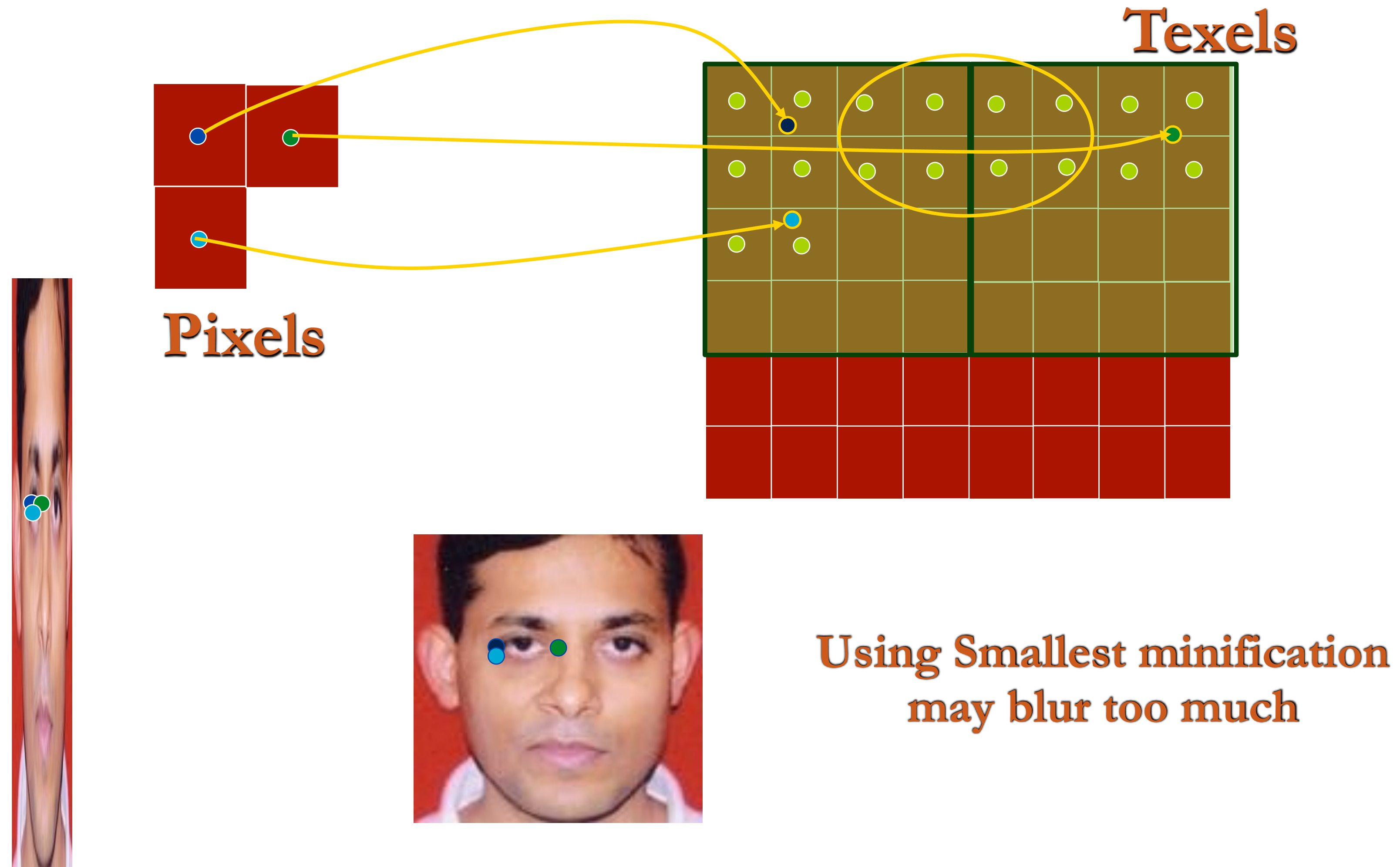
- Basic mipmapping can produce “jumps” in texture detail where LOD changes
- Trilinear filtering: interpolate between results of two adjacent mipmap levels
 - ➔ Bilinear interpolation at level $[k]$
 - ➔ Bilinear interpolation at level $[k]+1$
 - ➔ Linear interpolation between them



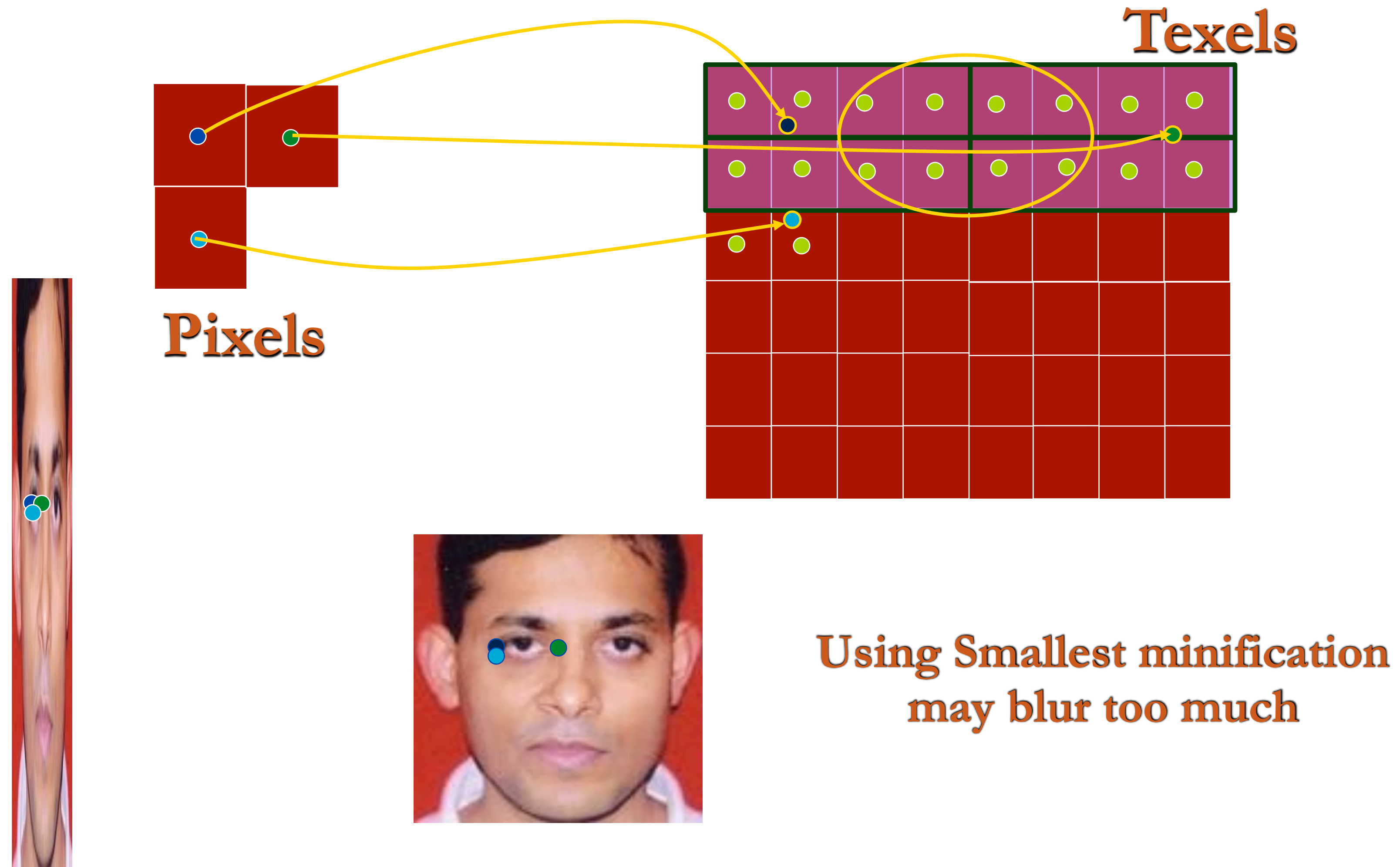
Anisotropic Filter



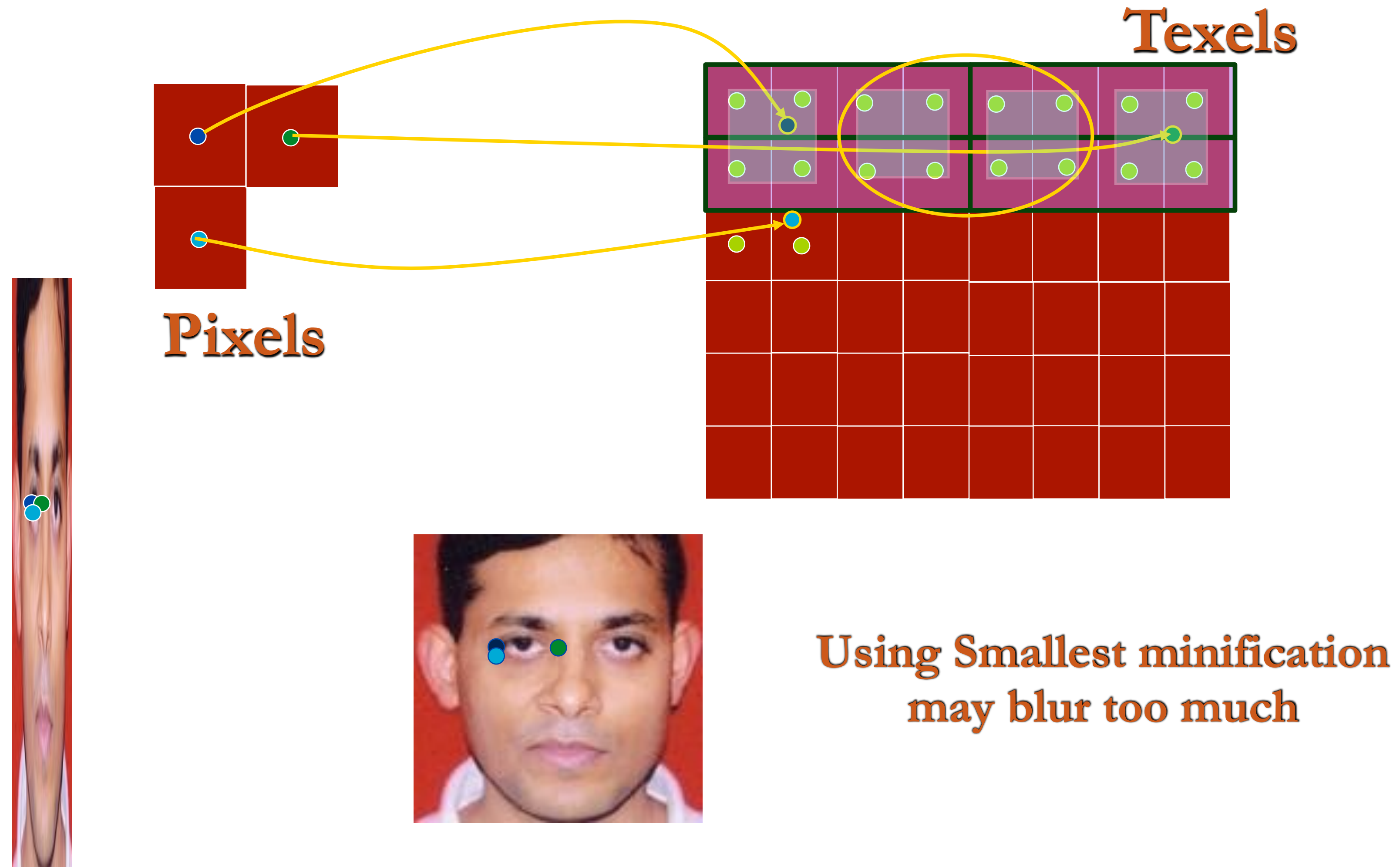
Anisotropic Filter



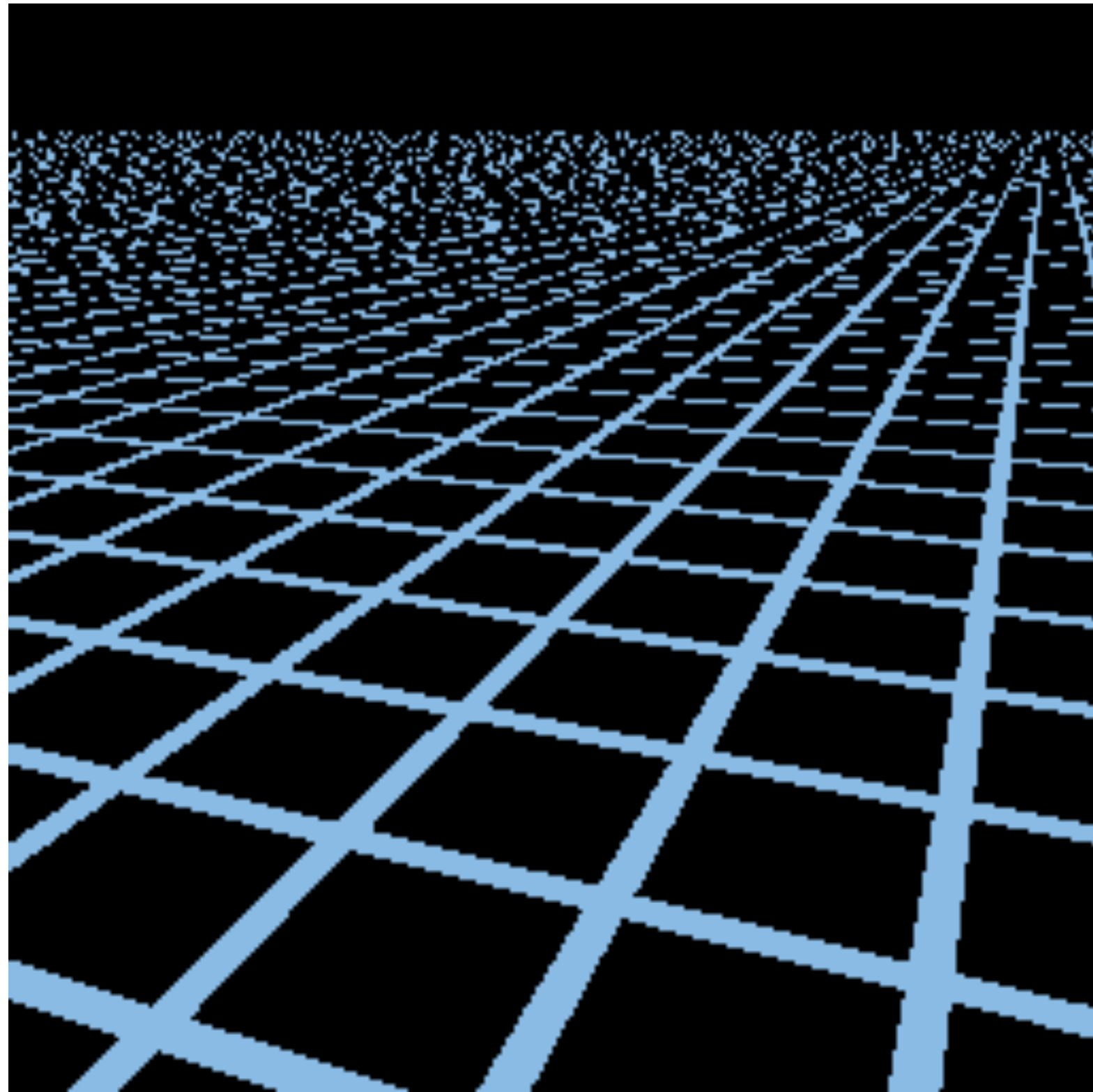
Anisotropic Filter



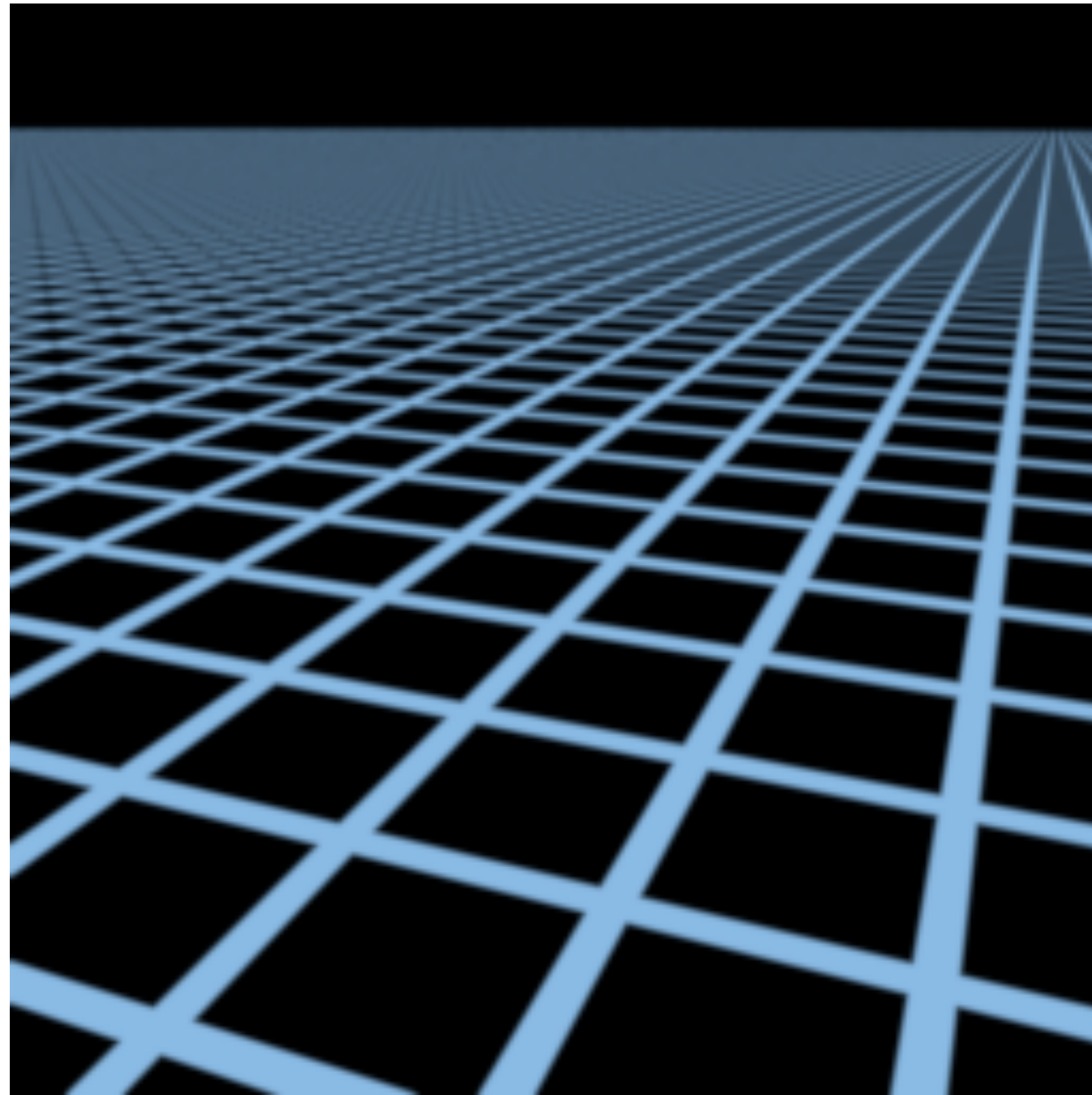
Anisotropic Filter



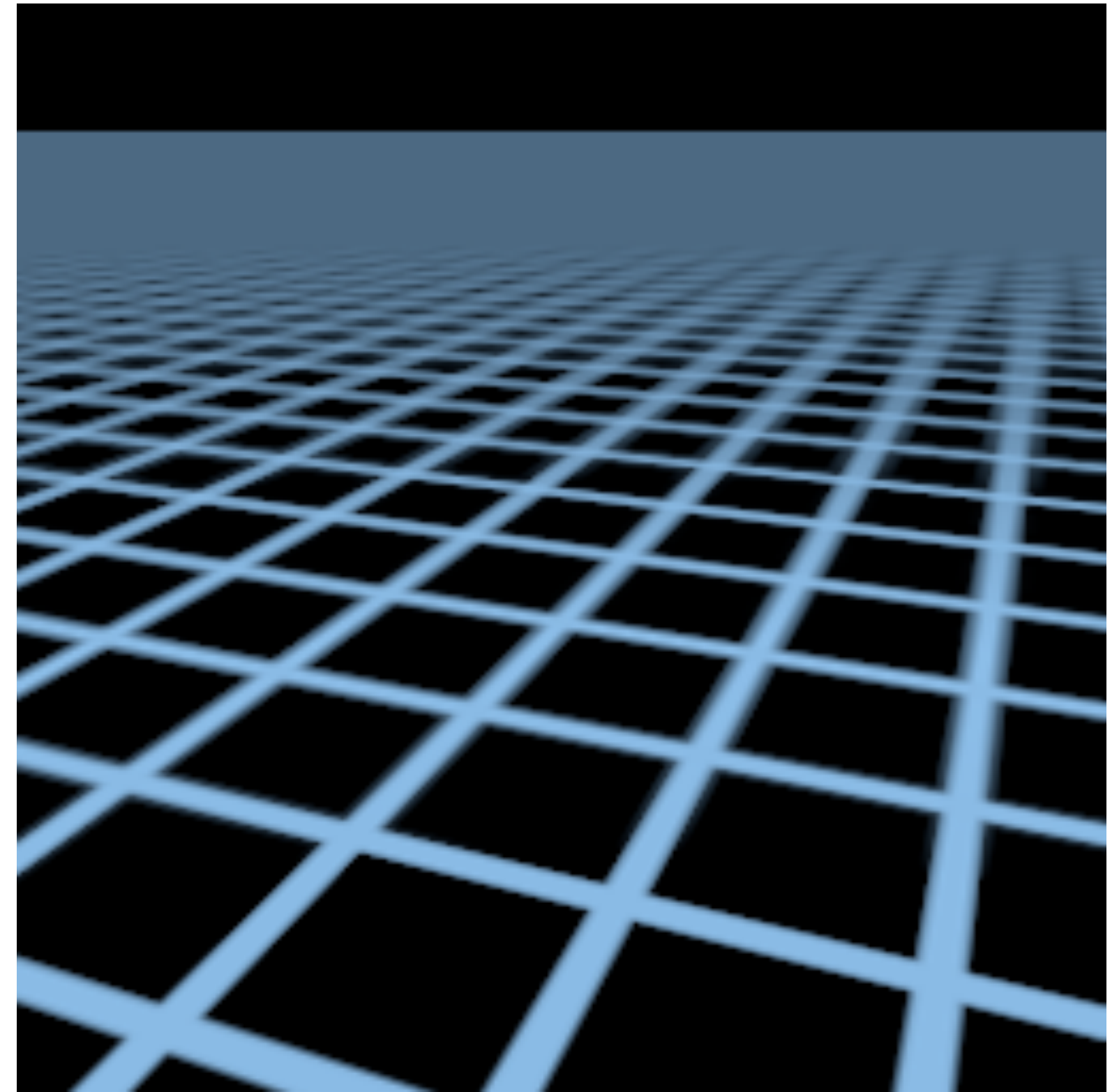
Mip-Map Filtering Examples



Point sampling

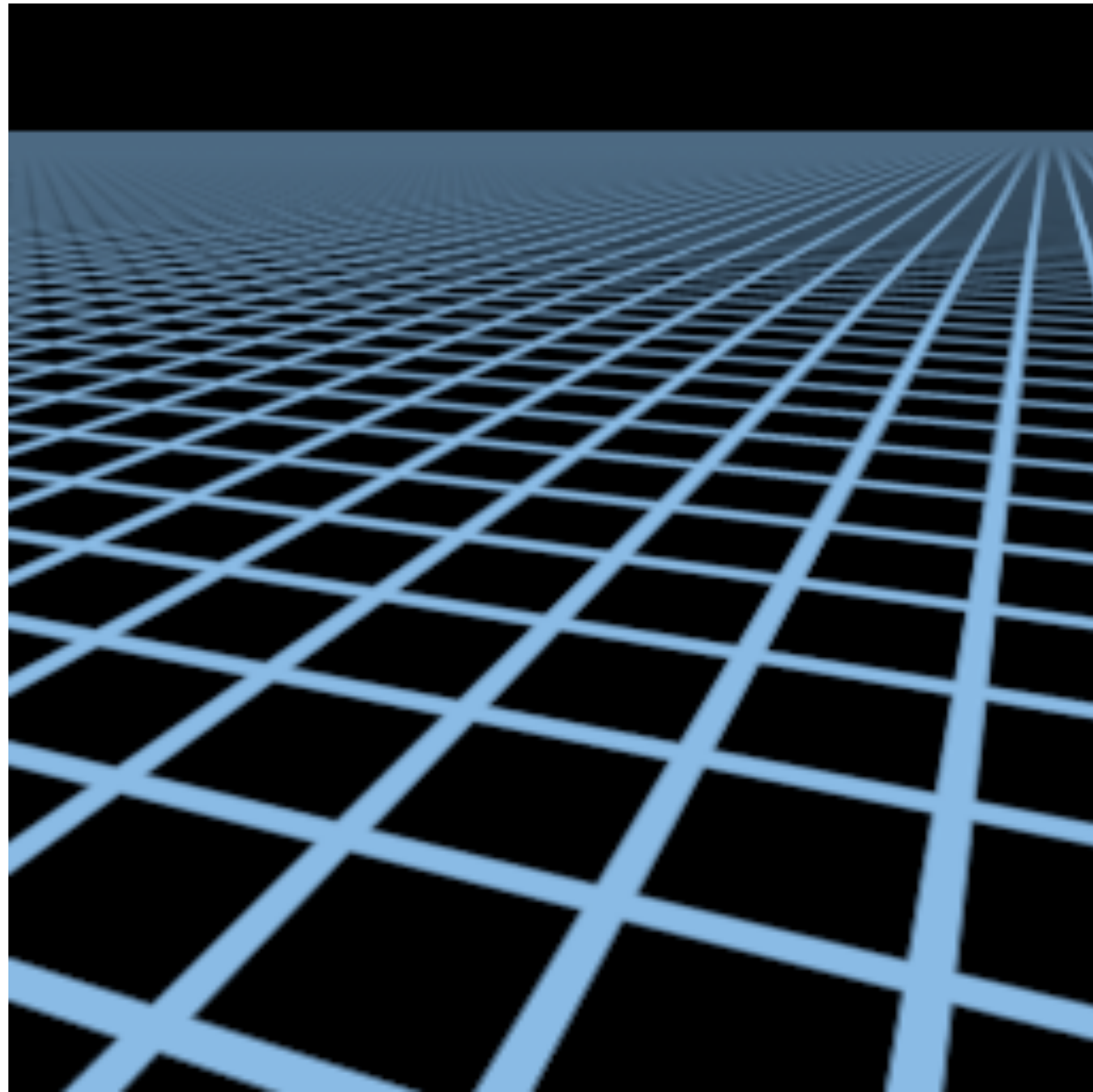


Super-sampling 512x

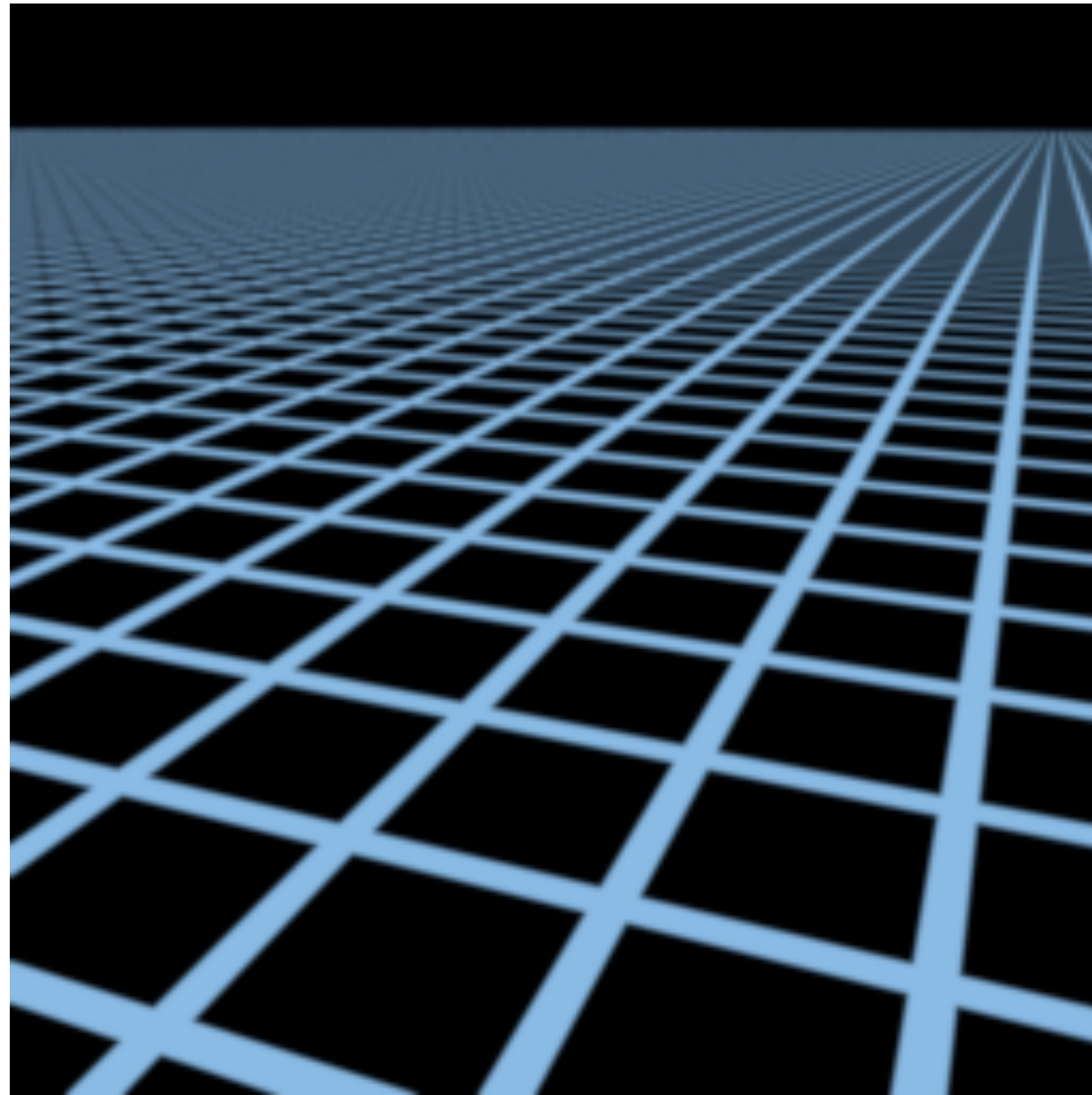


Mipmap with trilinear filtering

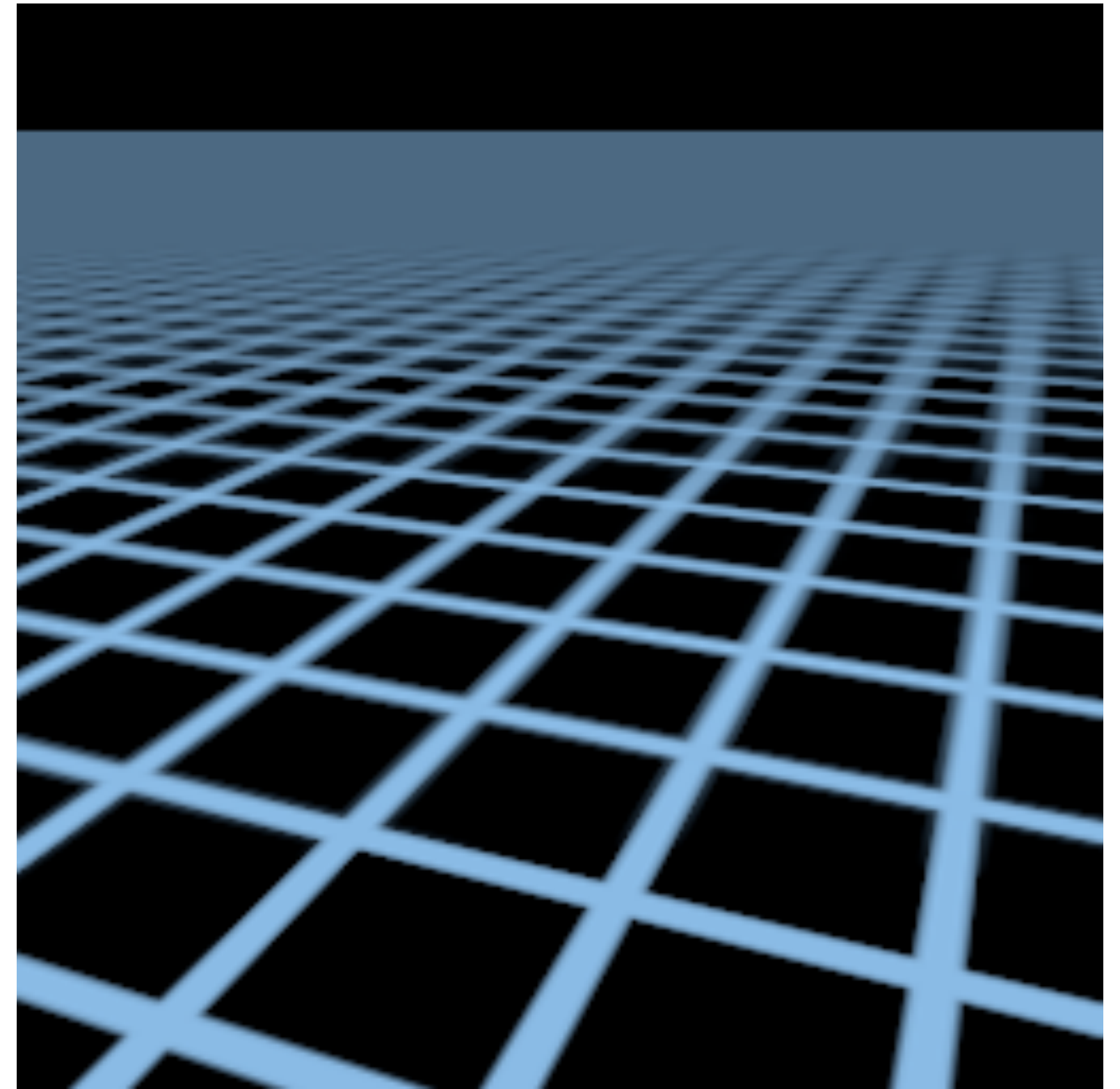
Mip-Map Filtering Examples



Elliptical weighted average

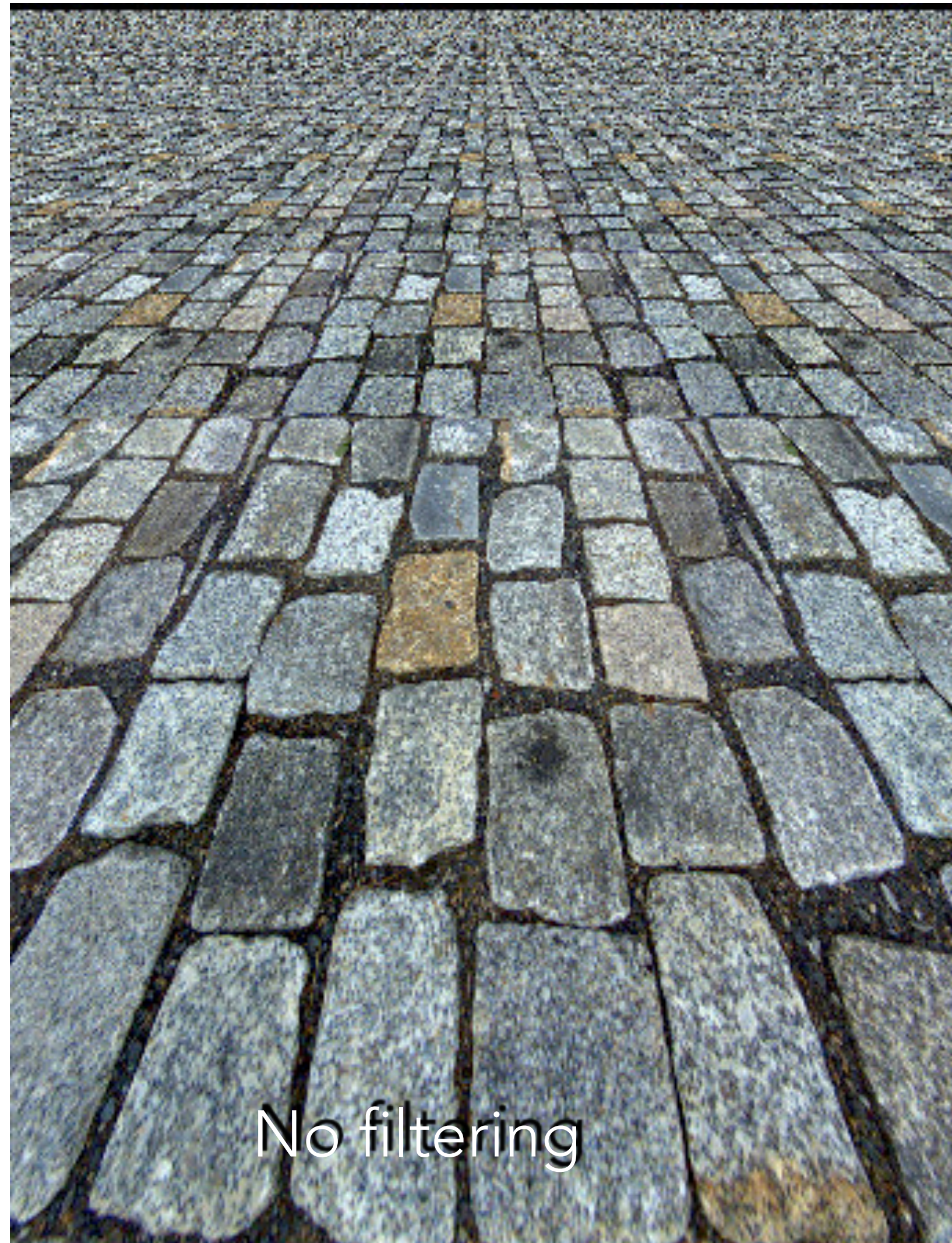
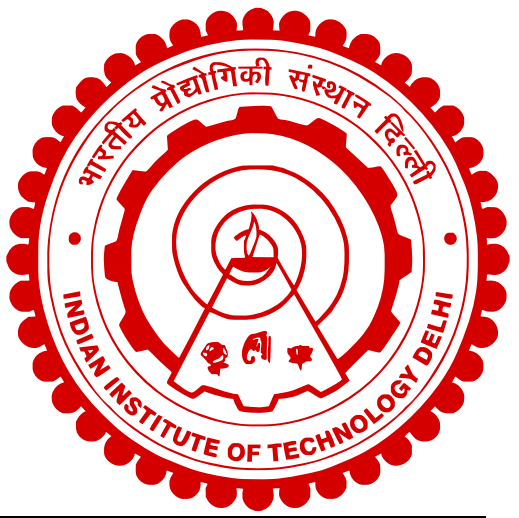


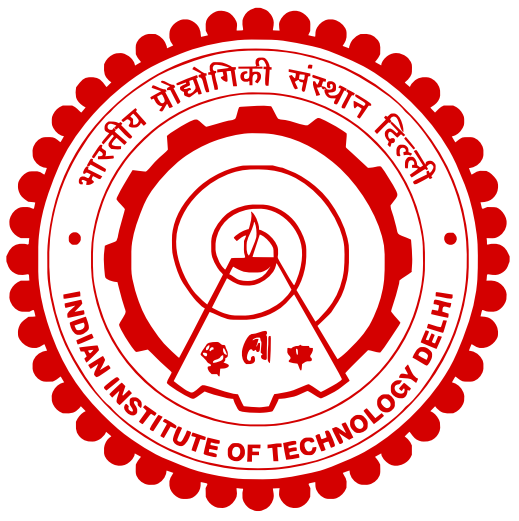
Super-sampling 512x



Mipmap with trilinear filtering

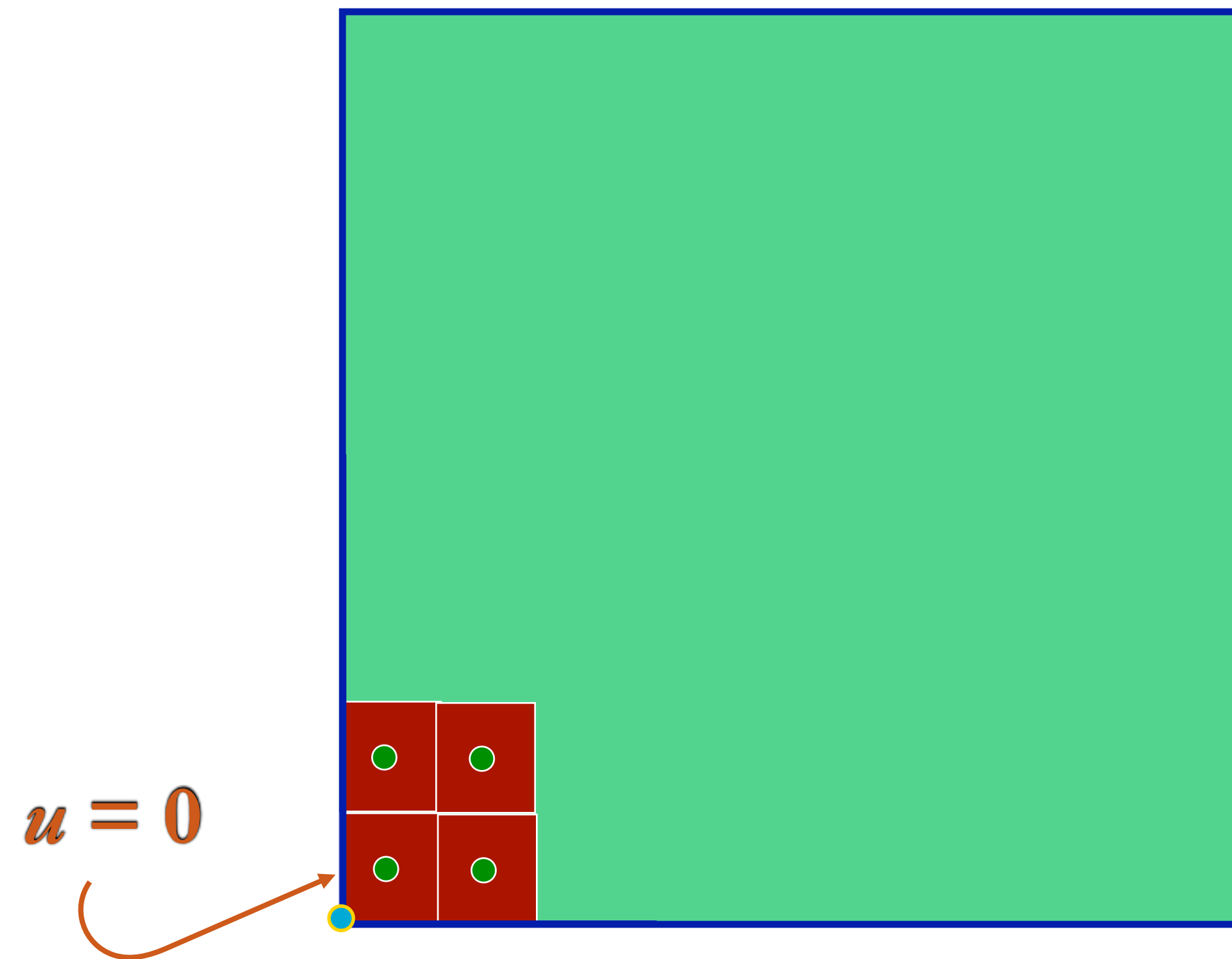
Anisotropic Filtering Example





Texture Border

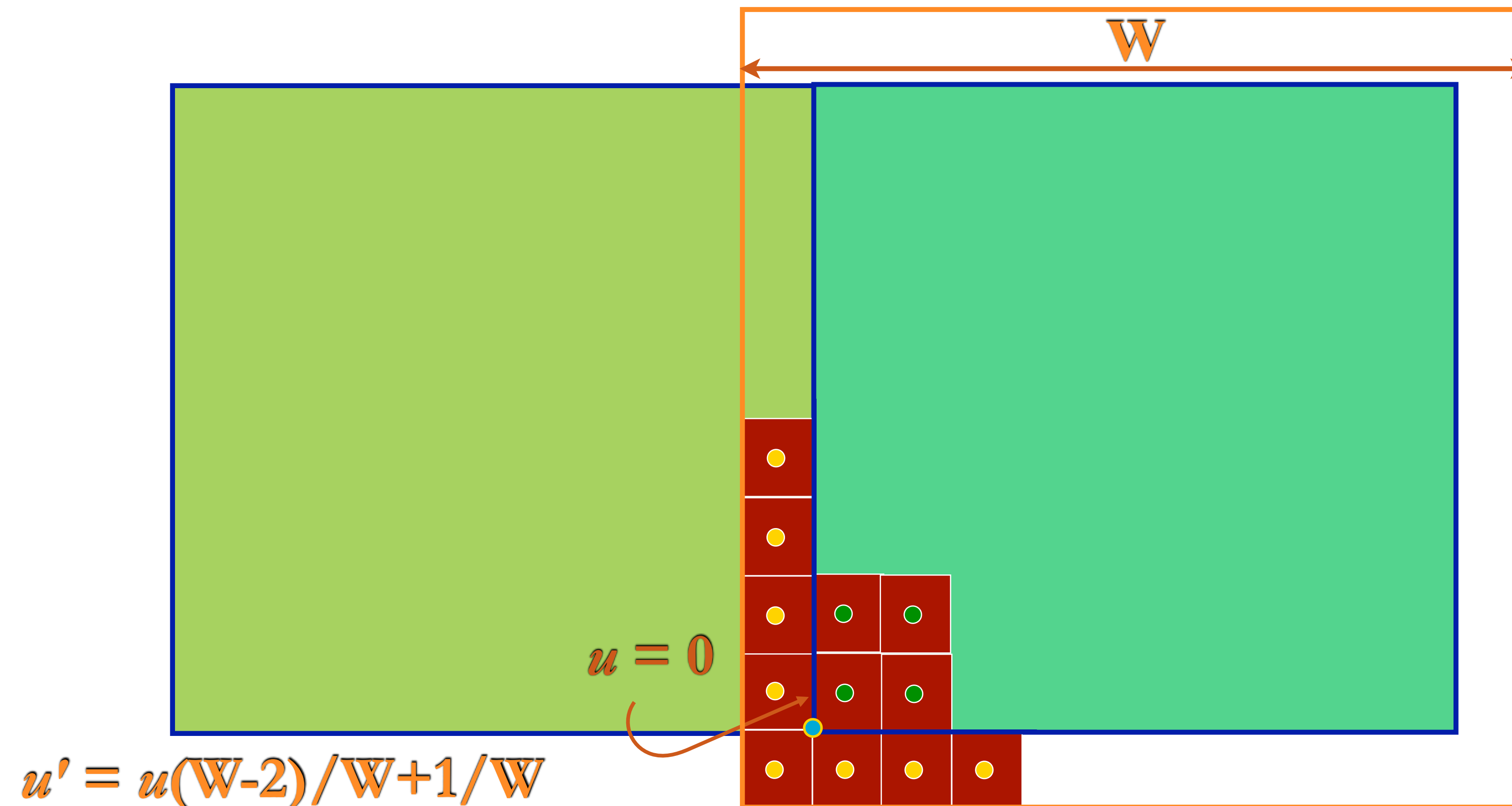
- Extra texel needed for linear interpolation

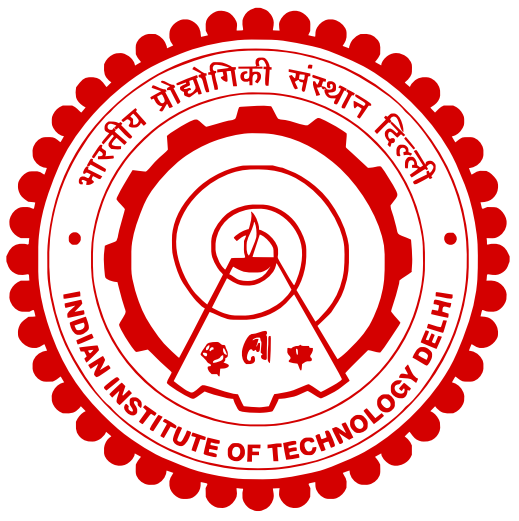




Texture Border

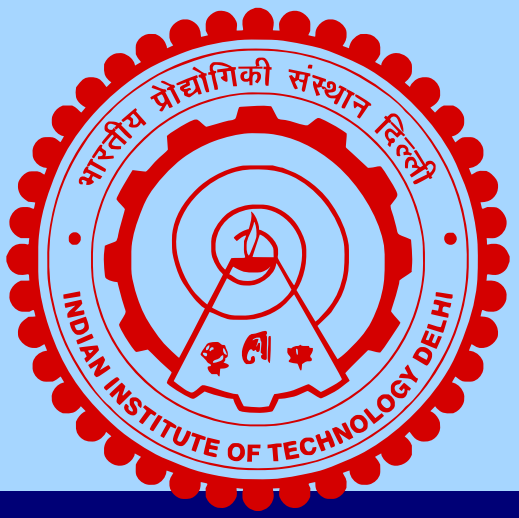
- Extra texel needed for linear interpolation





Review: Textures in OpenGL

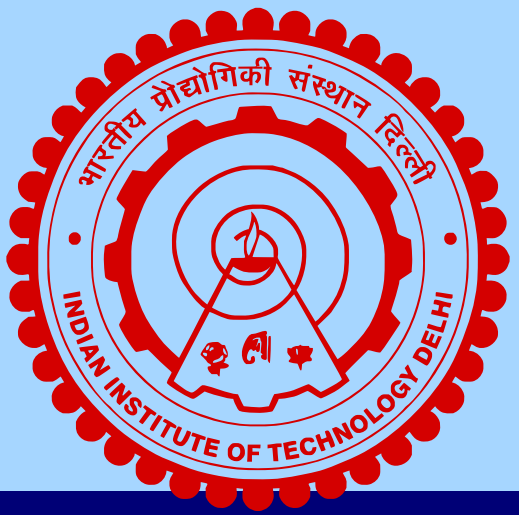
- Texture = Object containing "images" (1 or more)
 - ➔ Has an ID/name for reference on the client
 - ➔ Encapsulates: Data storage and sampling (how it is used)
- Texture has a format
 - ➔ 1D/2D/3D organization, Dimensions, data format
 - Specialized types also exist, e.g., cube map, texture array, multisample texture, etc.
 - ➔ Upto 4 components per sample + some data format of each component
- Client can define this data
- Shaders can Read or Write (not both for the same texture)
- Shaders look up with texture coordinates $[u, v, w]$ (interpolated from vertex tex-coord)
 - ➔ Also an LOD [MIP-MAP *level of detail*]



Example

SHADER

```
uniform sampler2D color_sampler;  
uniform sampler2D normal_sampler;  
out vec3 color;  
  
void main() {  
    vec3 tcolor = texture(color_sampler, UVcolor).rgb;  
    vec3 normal = normalize(texture(normal_sampler, UVnormal).rgb*2.0 - 1.0);  
    color = shade(tcolor, normal, ..);  
}
```



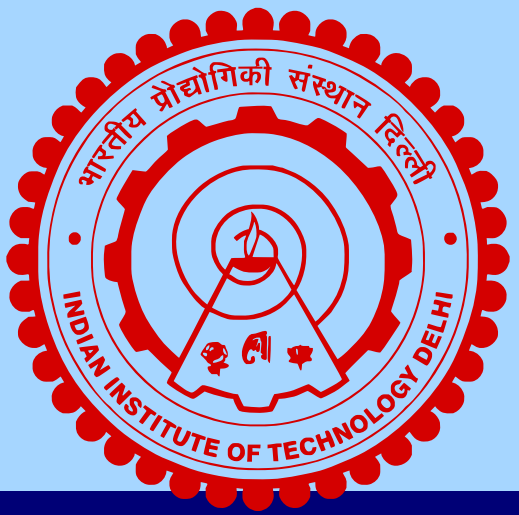
Example

```
uniform sampler2D color_sampler;  
uniform sampler2D normal_sampler;  
out vec3 color;
```

SHADER

```
glUseProgram(program); // Created and bound earlier  
GLint colorSampler = glGetUniformLocation(program, "color_sampler");  
GLint normSampler = glGetUniformLocation(program, "normal_sampler");  
glUniform1i(colorSampler, 0); // Associate Texture unit 0 with this sampler  
glUniform1i(normSampler, 5); // Associate Texture unit 5 with this sampler  
  
glActiveTexture(GL_TEXTURE0); // We will now set up Texture unit 0  
glBindTexture(GL_TEXTURE_2D, colorMap); // colorMap set up earlier  
glBindSampler(0, linearSampler); // linearSampler setup earlier  
glActiveTexture(GL_TEXTURE0 + 5); // Same as Constant GL_TEXTURE5  
glBindTexture(GL_TEXTURE_2D, normalMap); // normalMap set up earlier  
glBindSampler(5, linearSampler);  
  
render();
```

CLIENT



Example

```
uniform sampler2D color_sampler;  
uniform sampler2D normal_sampler;  
out vec3 color;
```

SHADER

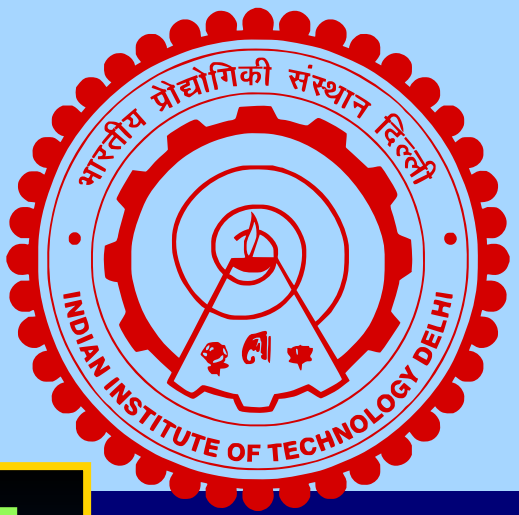
```
glUseProgram(program); // Created and bound earlier  
GLint colorSampler = glGetUniformLocation(program, "color_sampler");  
GLint normSampler = glGetUniformLocation(program, "normal_sampler");  
glUniform1i(colorSampler, 0); // Associate Texture unit 0 with this sampler  
glUniform1i(normSampler, 5); // Associate Texture unit 5 with this sampler
```

CLIENT

```
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, colorMap);  
glBindSampler(0, linearSampler);  
glActiveTexture(GL_TEXTURE5);  
glBindTexture(GL_TEXTURE_2D, normalMap);  
glBindSampler(5, linearSampler);  
  
render();
```

```
glGenTextures(1, &colorMap);  
glBindTexture(GL_TEXTURE_2D, colorMap);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB,  
              GL_UNSIGNED_BYTE, data); // also see glTexStorage2D  
glGenTextures(1, &normalMap);  
glBindTexture(GL_TEXTURE_2D, normalMap);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB, GL_FLOAT, data);  
e.g., GL_LINEAR_MIPMAP_LINEAR  
glGenSamplers(1, &linearSampler);  
glSamplerParameteri(linearSampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR);  
glSamplerParameteri(linearSampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

INITIALIZE



Example

```
uniform sampler2D color_sampler;  
uniform sampler2D normal_sampler;  
out vec3 color;
```

```
glUseProgram(program); // Created and bound earlier  
GLint colorSampler = glGetUniformLocation(program, "color_sampler");  
GLint normSampler = glGetUniformLocation(program, "normal_sampler");  
glUniform1i(colorSampler, 0); // Associate Texture unit 0 with this sampler  
glUniform1i(normSampler, 5); // Associate Texture unit 5 with this sampler
```

```
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, colorMap);  
glBindSampler(0, linearSampler);  
glActiveTexture(GL_TEXTURE5);  
glBindTexture(GL_TEXTURE_2D, normalMap);  
glBindSampler(5, linearSampler);  
  
render();
```

SHADER

ALSO SEE

```
glGenerateTextureMipmap(textureID);  
glSamplerParameterf(samplerID,  
    GL_TEXTURE_MAX_ANISOTROPY, 4.0f);  
textureProj(sampler, projTexCoord);  
textureLod(sampler, texCoord, lod);  
textureGrad(sampler, texCoord, dtdx, dtdy);
```

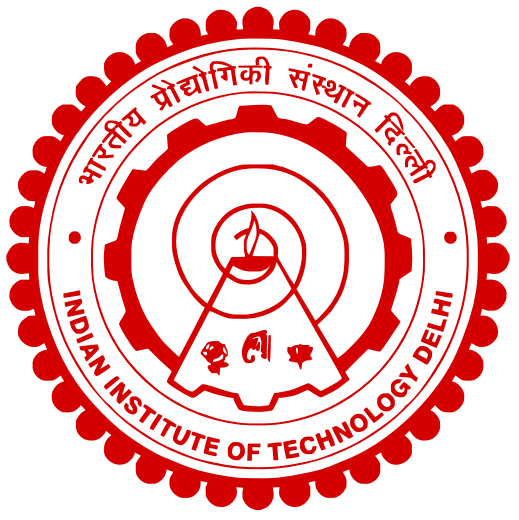
```
glGenTextures(1, &colorMap);  
glBindTexture(GL_TEXTURE_2D, colorMap);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB,  
    GL_UNSIGNED_BYTE, data); // also see glTexStorage2D  
glGenTextures(1, &normalMap);  
glBindTexture(GL_TEXTURE_2D, normalMap);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB, GL_FLOAT, data);  
  
glGenSamplers(1, &linearSampler);  
glSamplerParameteri(linearSampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR);  
glSamplerParameteri(linearSampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

LOD

Border

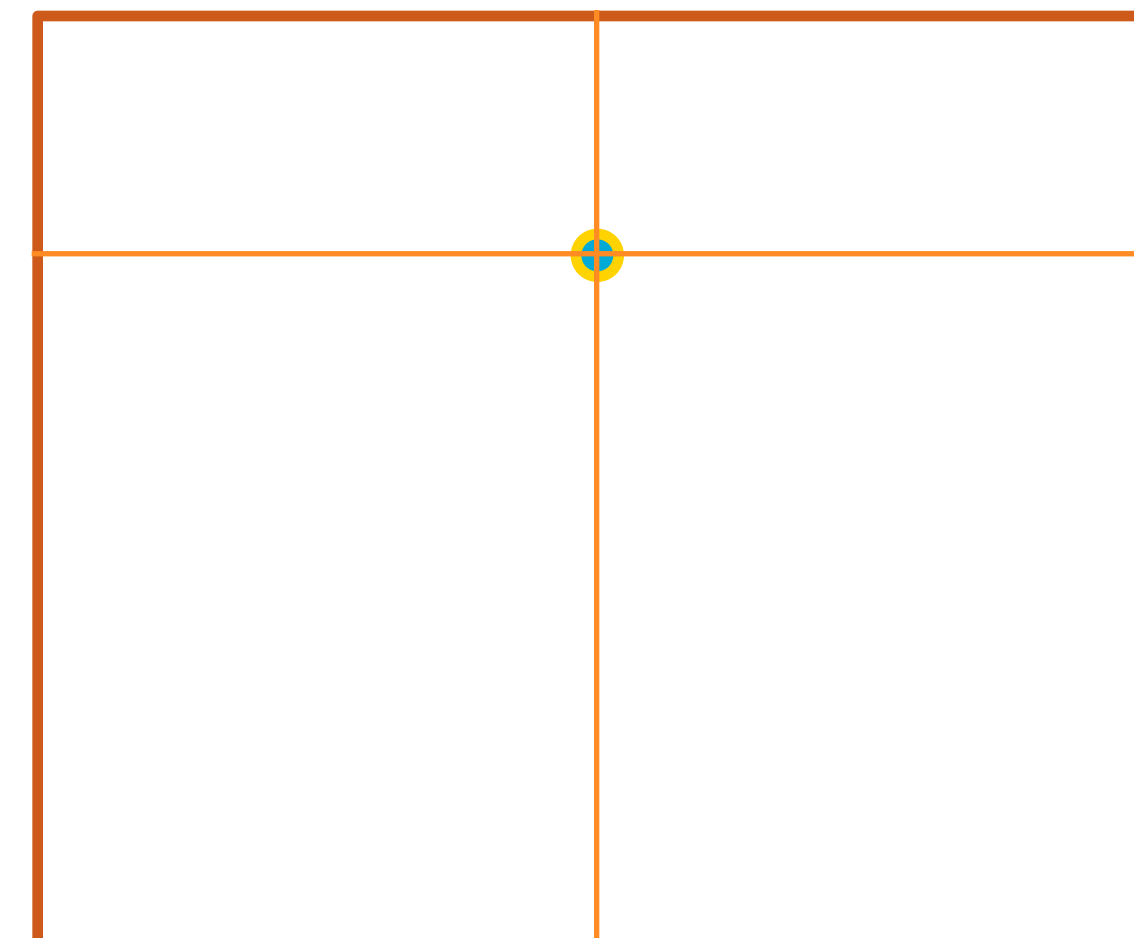
INITIALIZE

e.g., GL_LINEAR_MIPMAP_LINEAR

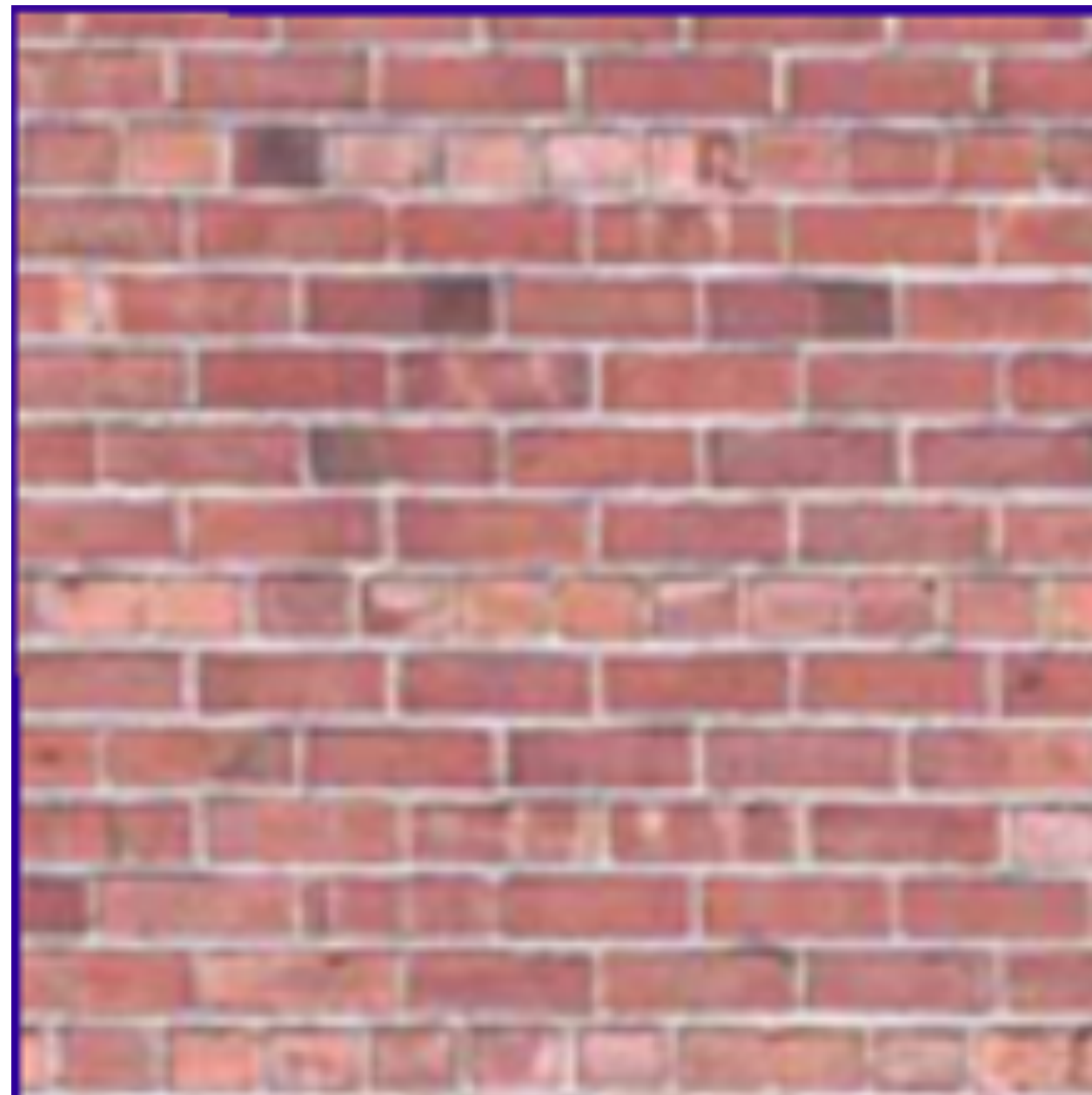


Other Texture Usage

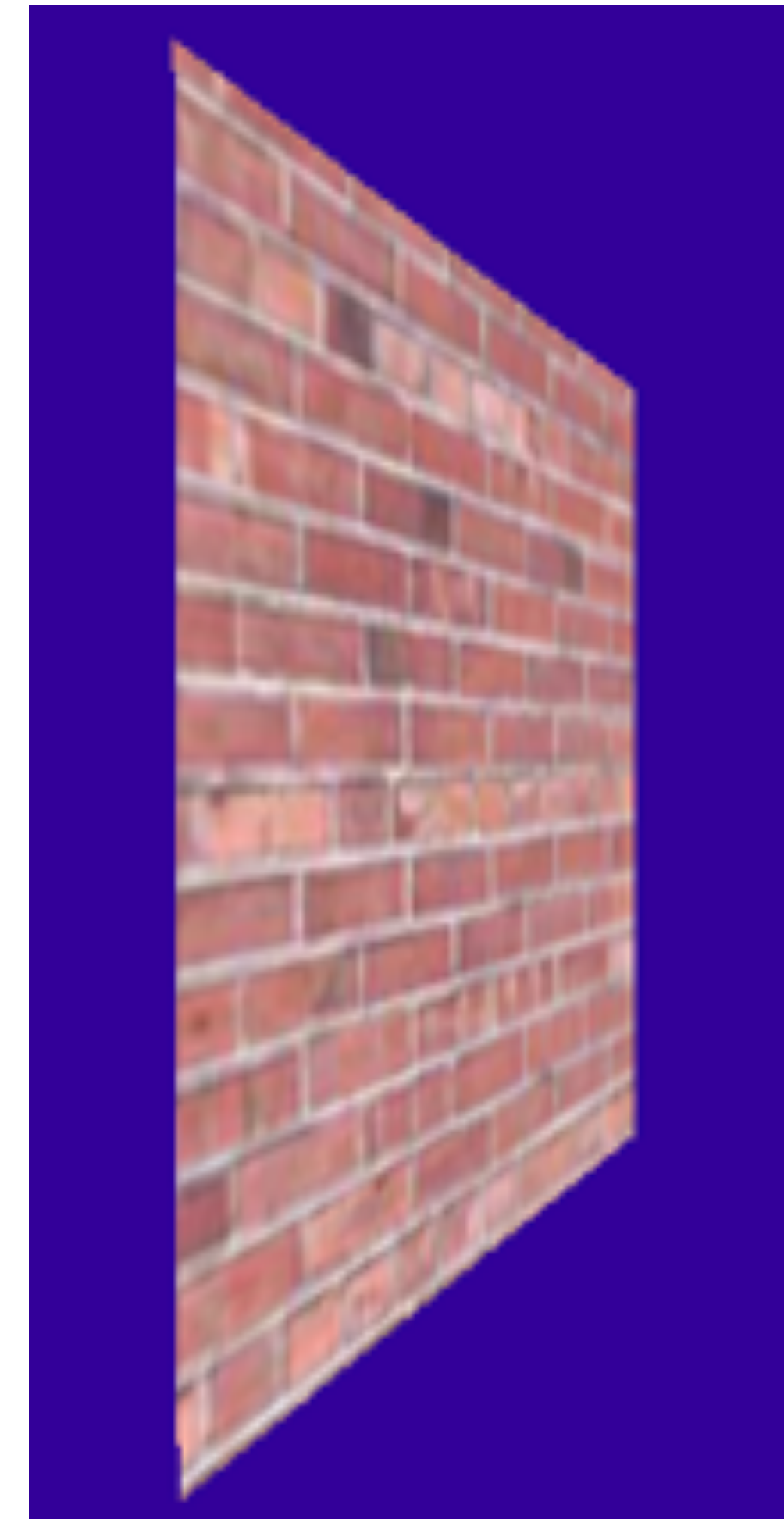
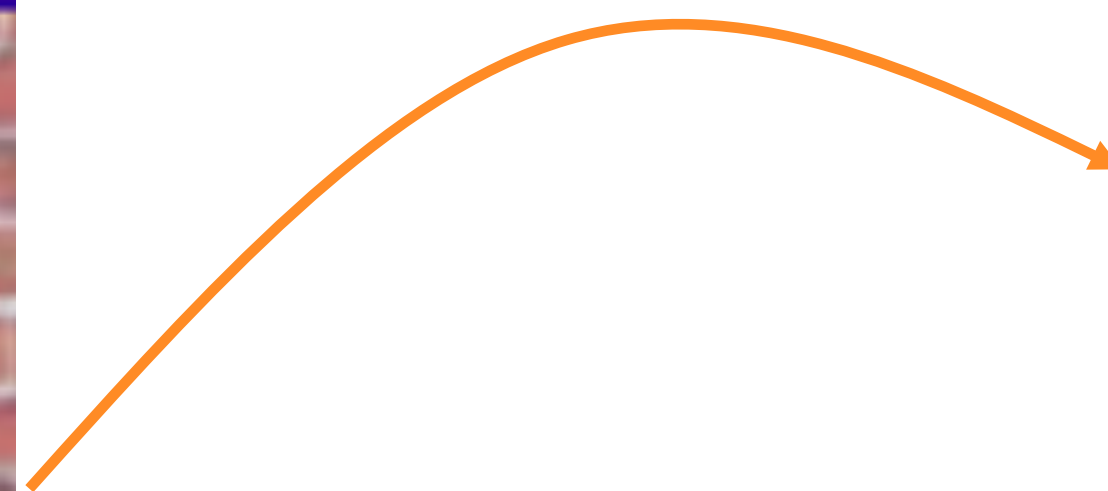
- Alpha Map
 - ➡ Transparent texture
- Light Map
 - ➡ Irradiance is independent of viewer
 - ➡ How do you generate it?
 - Texel to Object space reverse-mapping
- Displacement Map
- Bump Map
- Environment Map
- Point Sprites



Diffuse Color Lookup

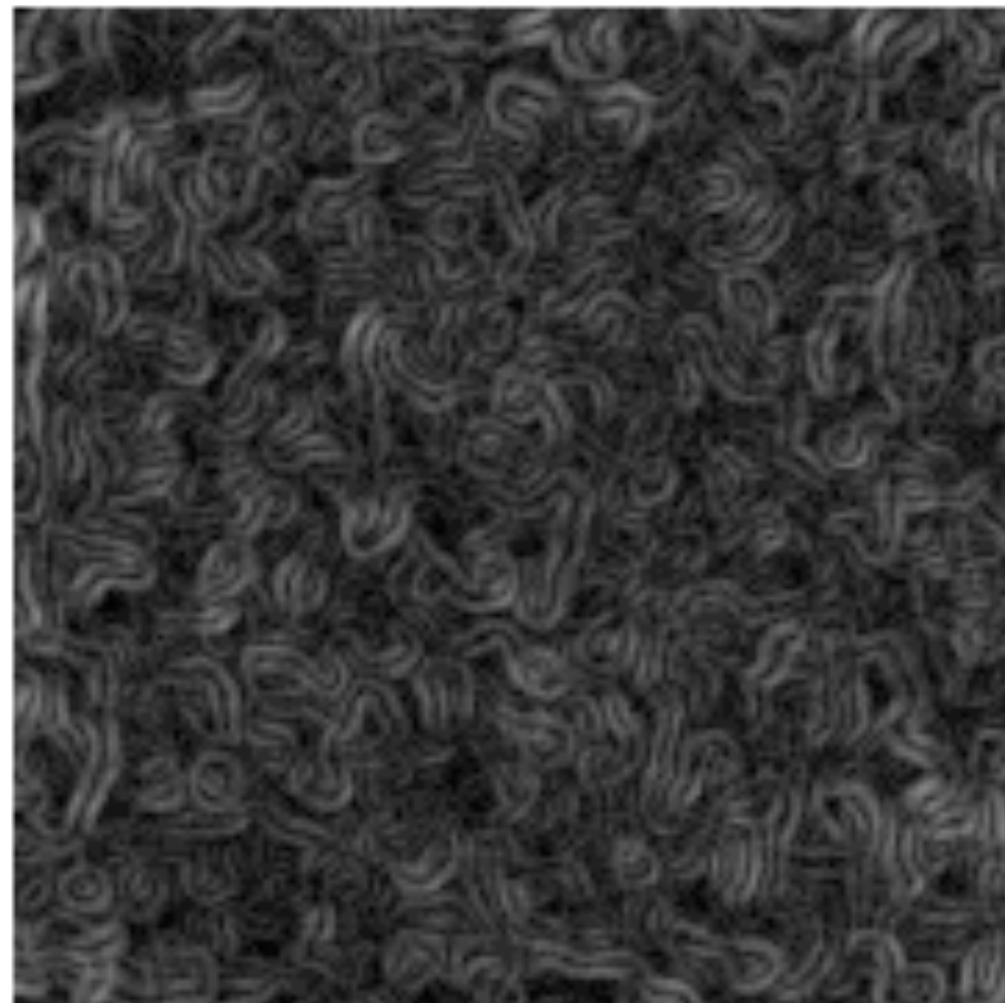


Texture

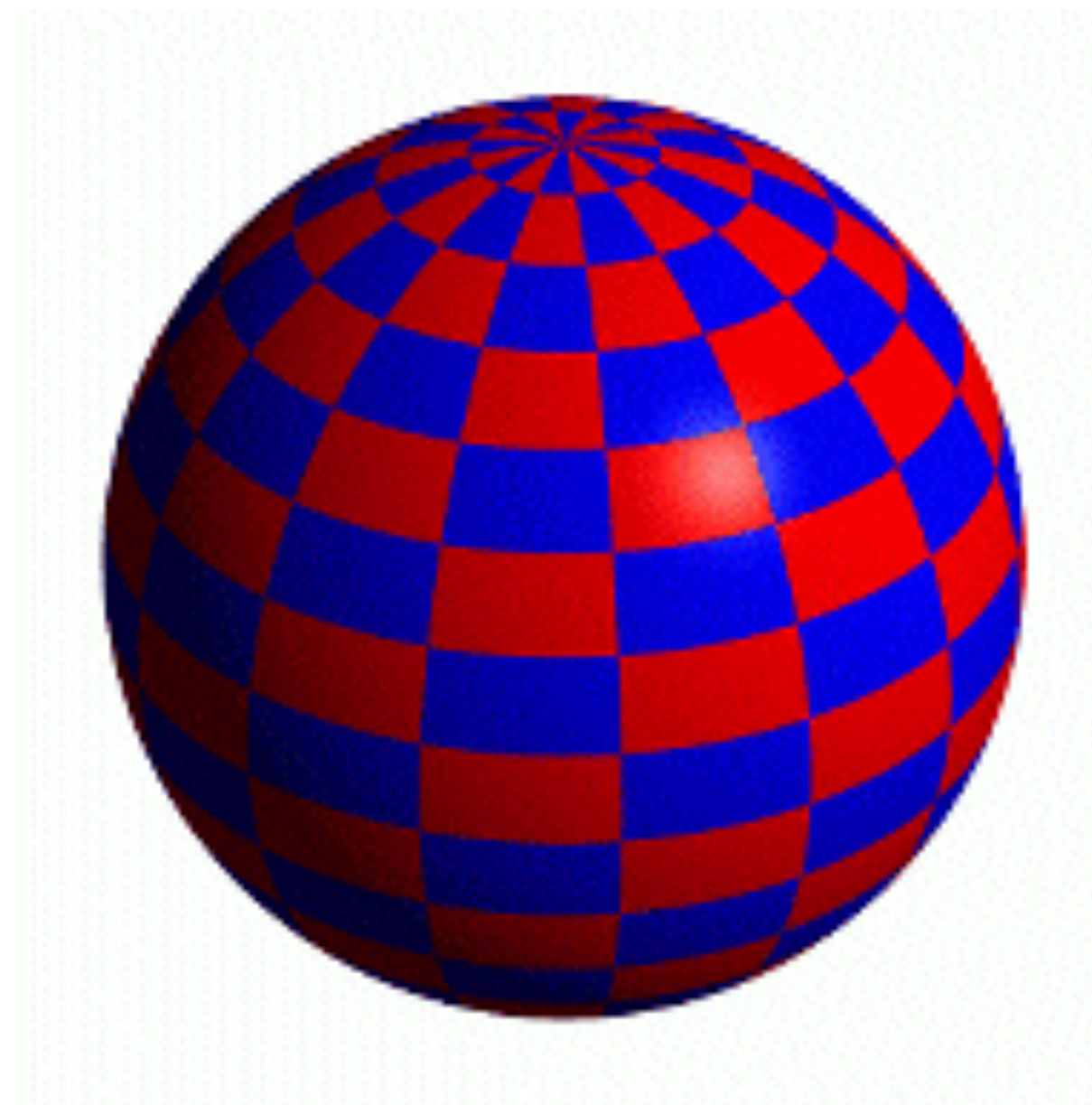


Textured Surface

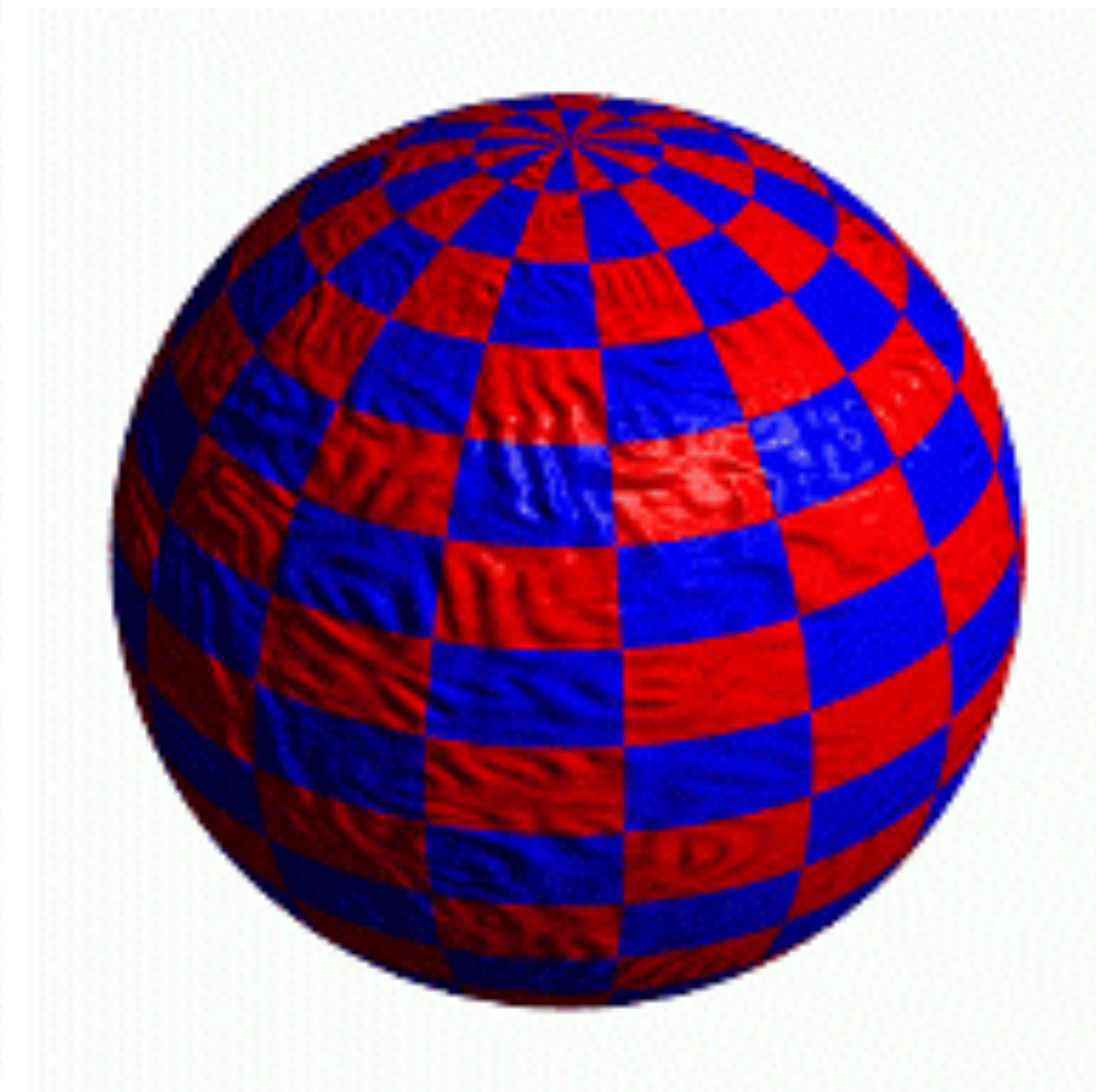
Bump Mapping



Texture



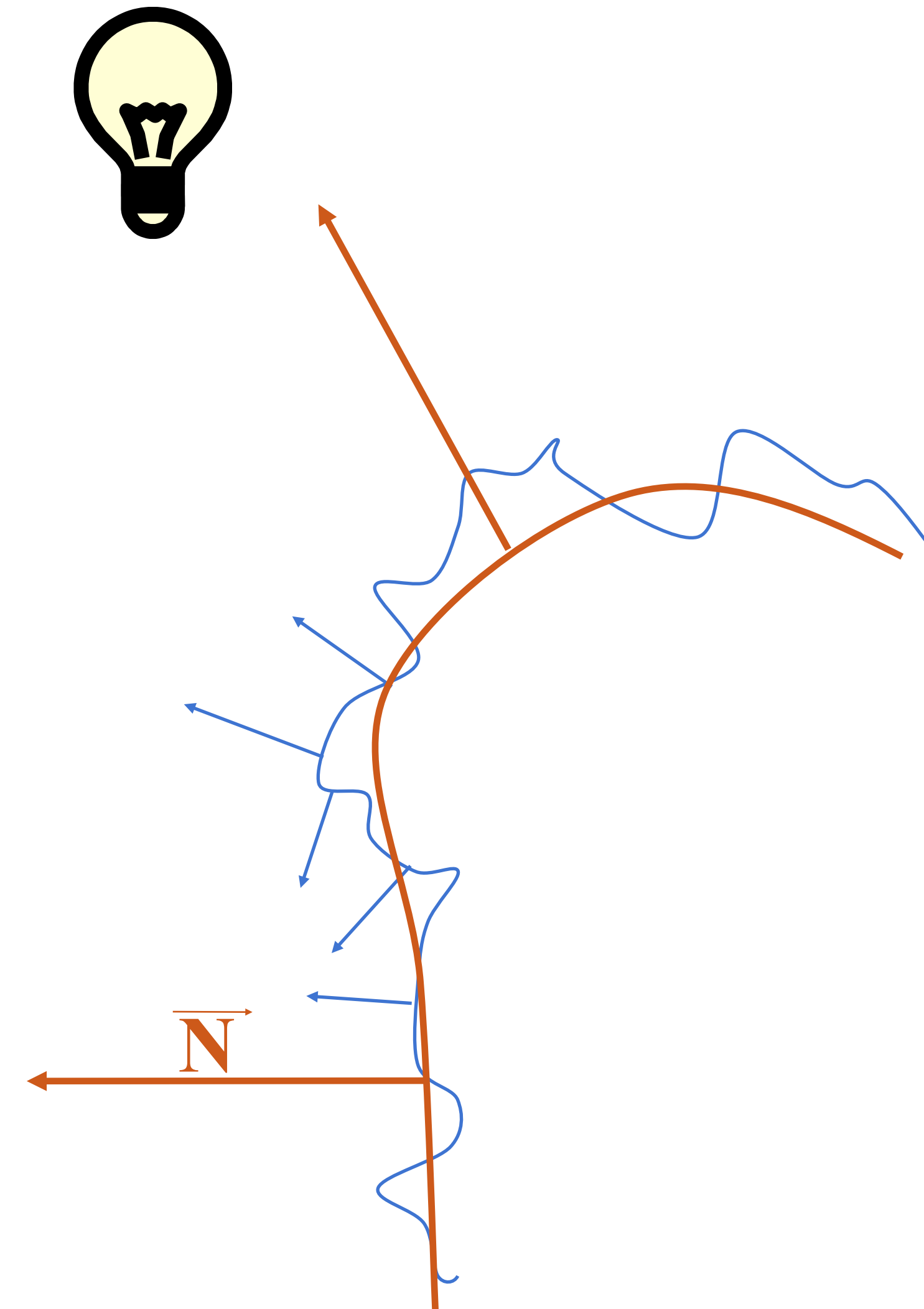
Surface



Textured Surface

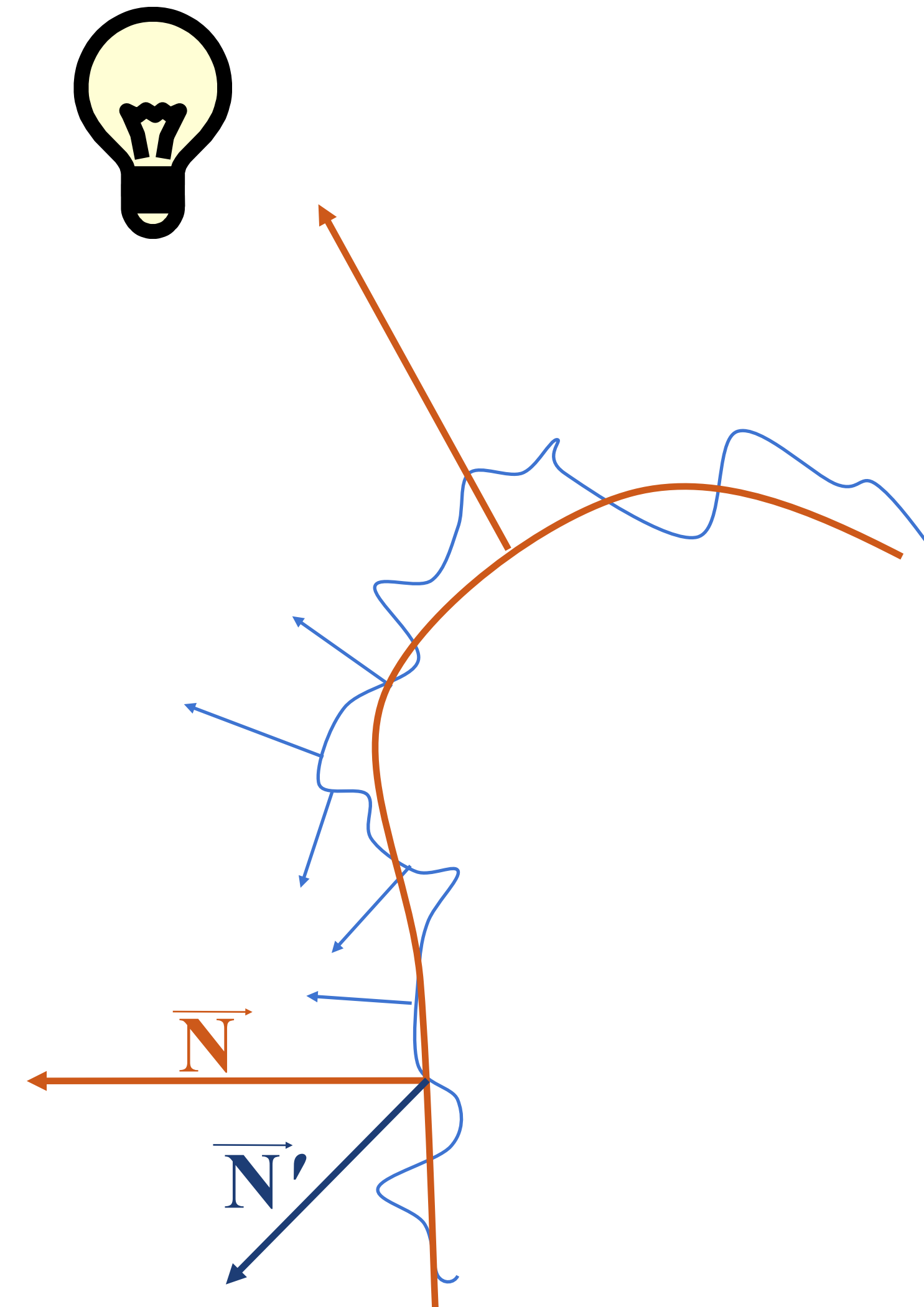
Bump Mapping

- Physically displace
 - ➔ Generate many triangles
- Simulate
 - ➔ Only change the normal
 - ➔ Normal affects illumination
- Works well for small bumps



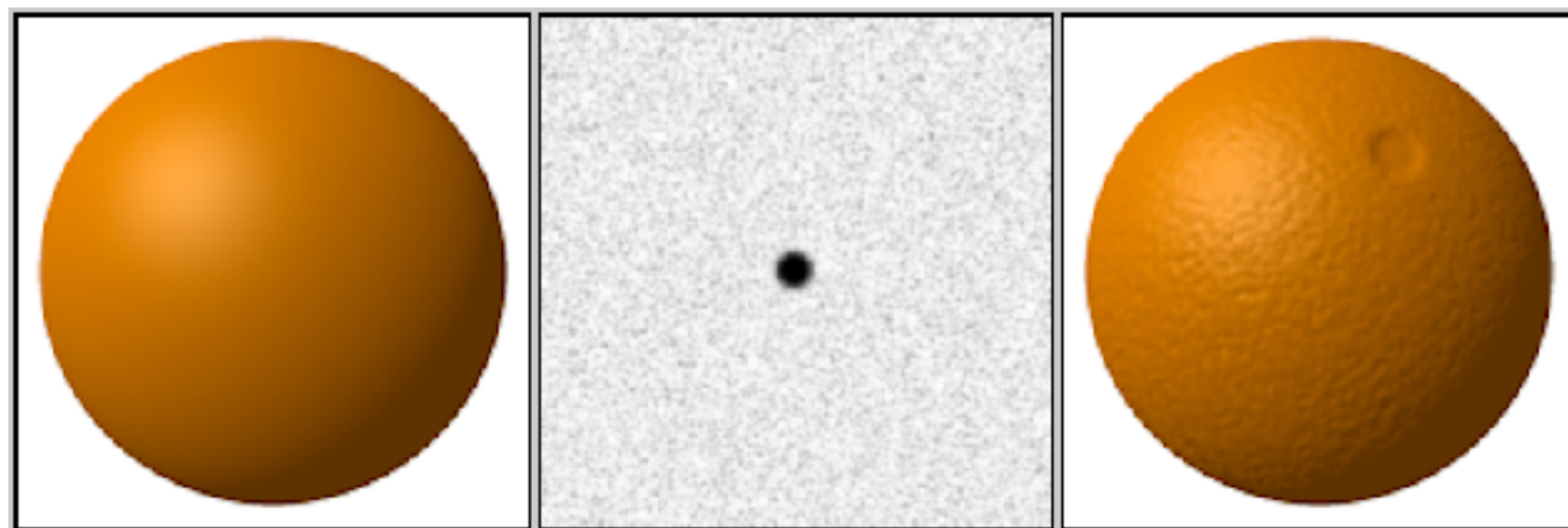
Bump Mapping

- Physically displace
 - ➔ Generate many triangles
- Simulate
 - ➔ Only change the normal
 - ➔ Normal affects illumination
- Works well for small bumps

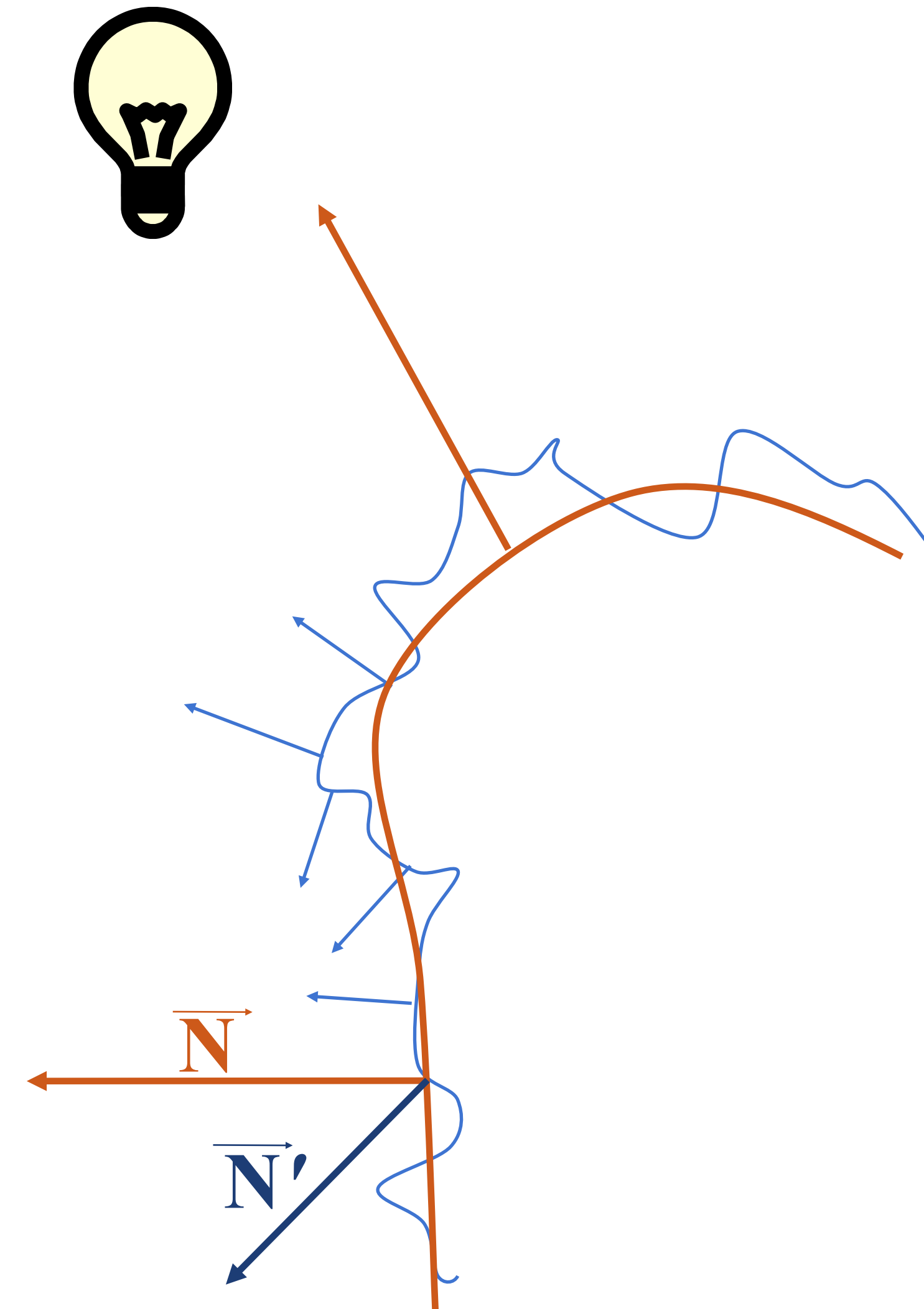


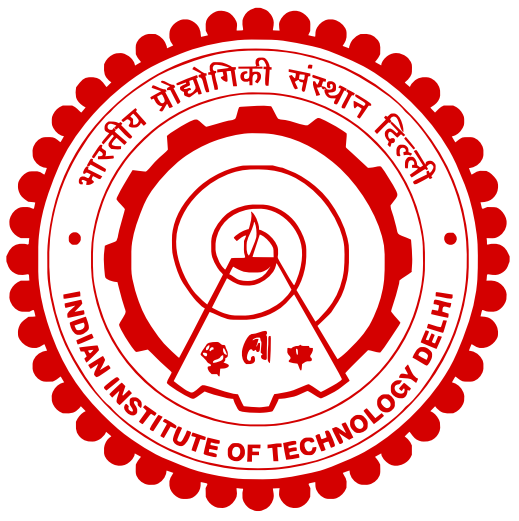
Bump Mapping

- Physically displace
 - ➔ Generate many triangles
- Simulate
 - ➔ Only change the normal
 - ➔ Normal affects illumination
- Works well for small bumps



[Wikipedia]





Bump Mapping

- Assume

- ➡ 'Bump' to the surface is small
- ➡ In the direction of the normal
- ➡ **aka** Height-map

- Simulation:

$$\vec{P'} = \vec{P} + b_P \vec{N}$$

$$\vec{P'}(u,v) = \vec{P}(u,v) + b(u,v) \vec{N}(u,v)$$



Bump Mapping

- Tangent to the surface $P(u,v)$ at a point (u,v) : P_u, P_v

- The normal is then:

- $N(u,v) = P_u \times P_v$

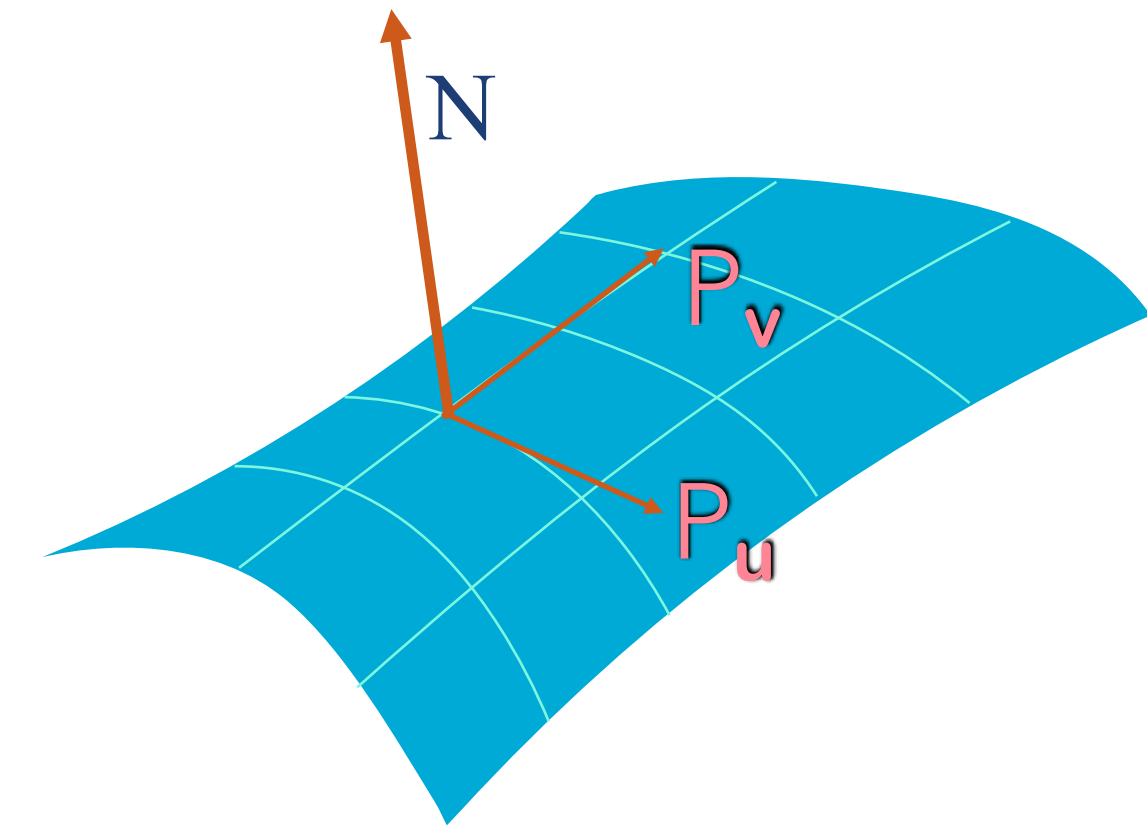
- $P'(u,v) = P(u,v) + b(u,v) N$

- $P'_u = P_u + b_u N + b N_u \approx P'_u = P_u + b_u N$ (if b is small)

- $P'_v = P_v + b_v N + b N_v \approx P'_v = P_v + b_v N$ (if b is small)

- $P'_u \times P'_v = P_u \times P_v + b_u(N \times P_v) - b_v(N \times P_u) + b_u b_v(N \times N) = N + b_u(N \times P_v) - b_v(N \times P_u) \approx N - b_u P_u - b_v P_v$

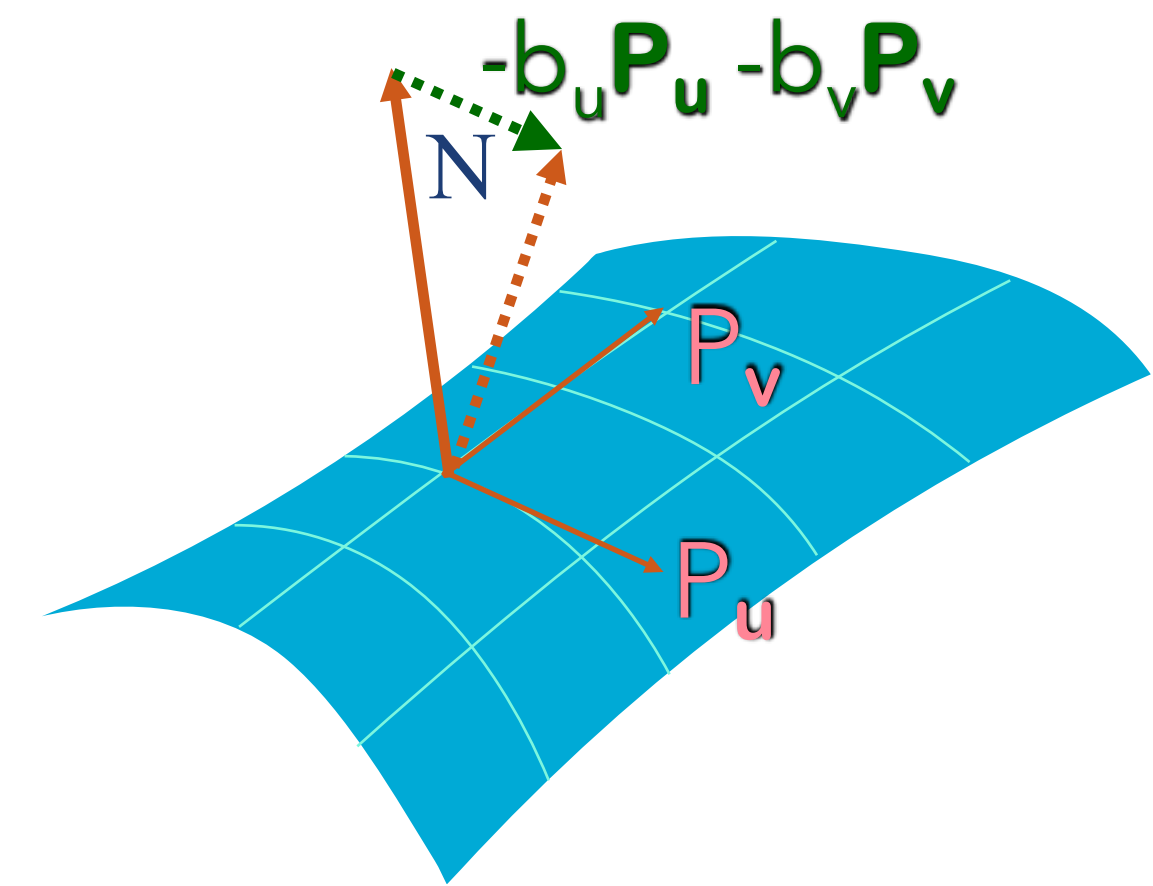
if P_u, P_v are almost perpendicular





Bump Mapping Implementation

- Rasterizer provides derivatives in x and y pixel directions
- May store b_u and b_v as a 2-component texture
 - ➔ Surface derivatives as vectors
 - possibly interpolated from derivatives at vertices
- Remember to Normalize N'



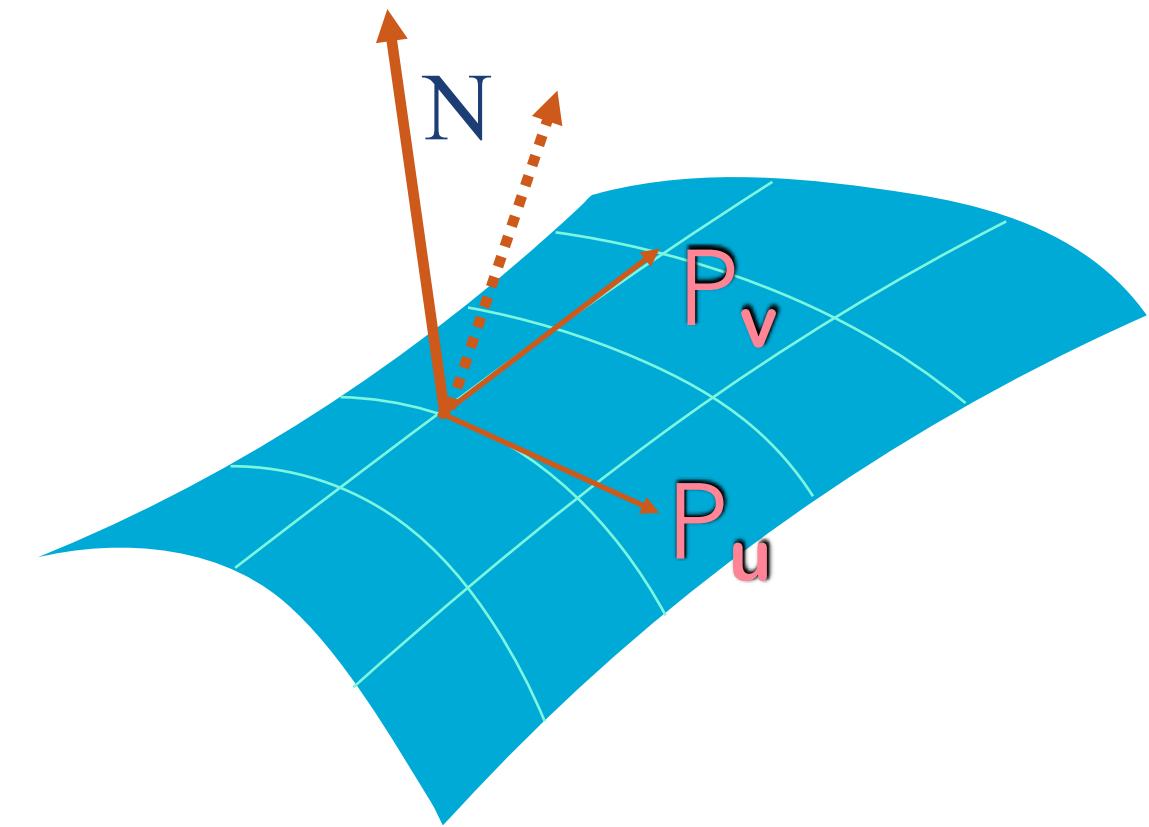


Normal Mapping

- Directly store the new normal per texel
- maybe, from bump map: $N - b_u P_u - b_v P_v$

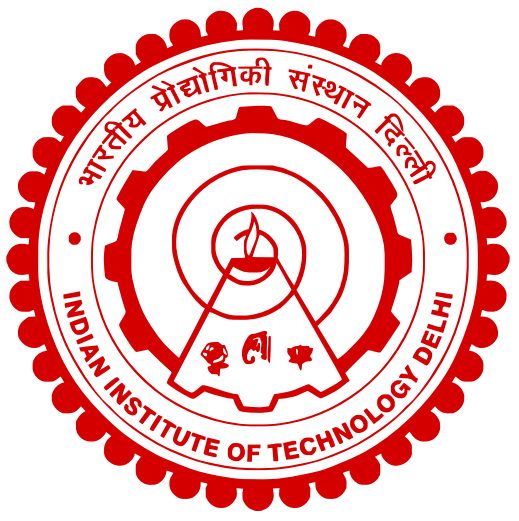
$$= -b_u, -b_v, 1$$

Convenient to store normals in
a surface-local coordinate system

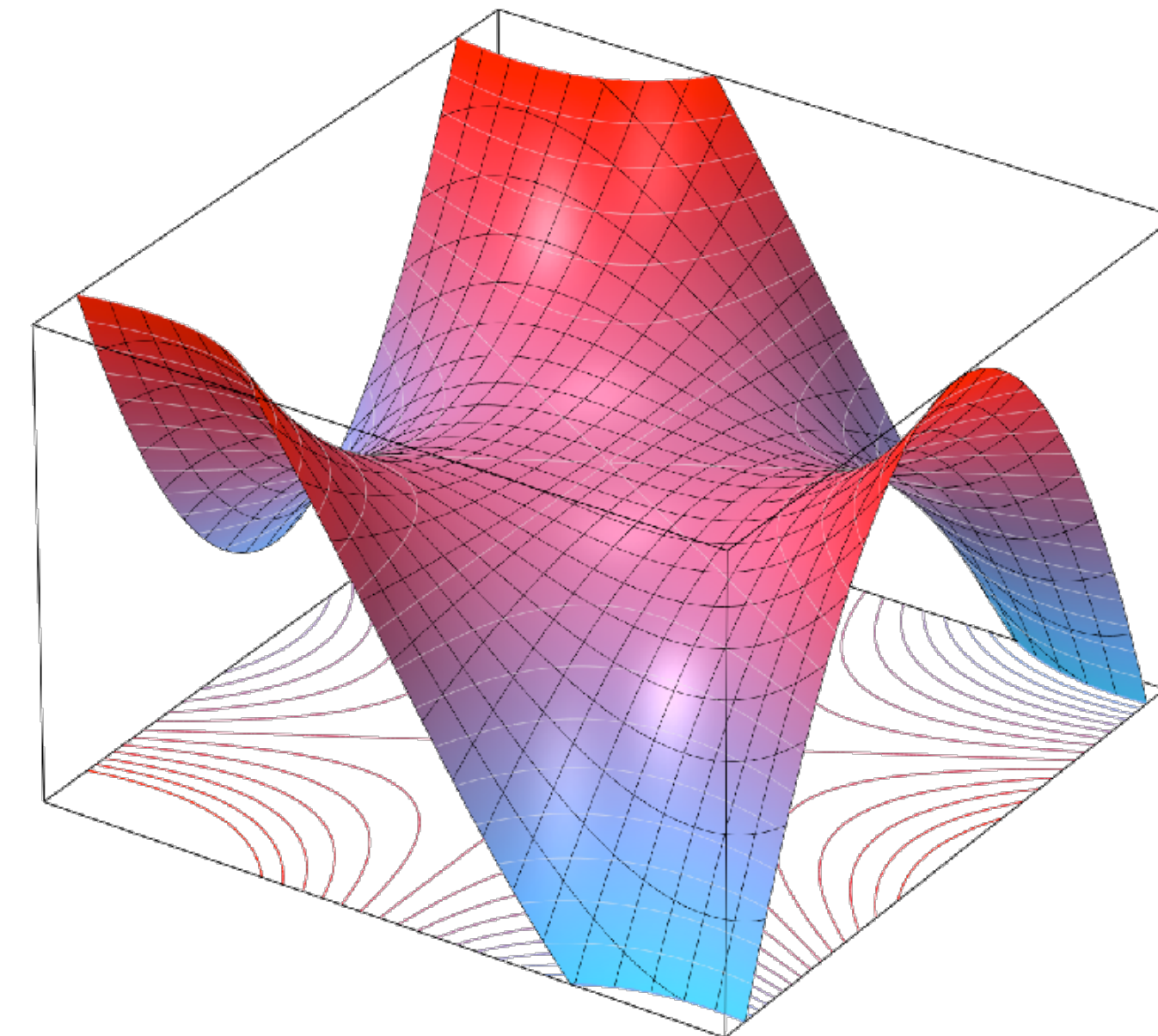
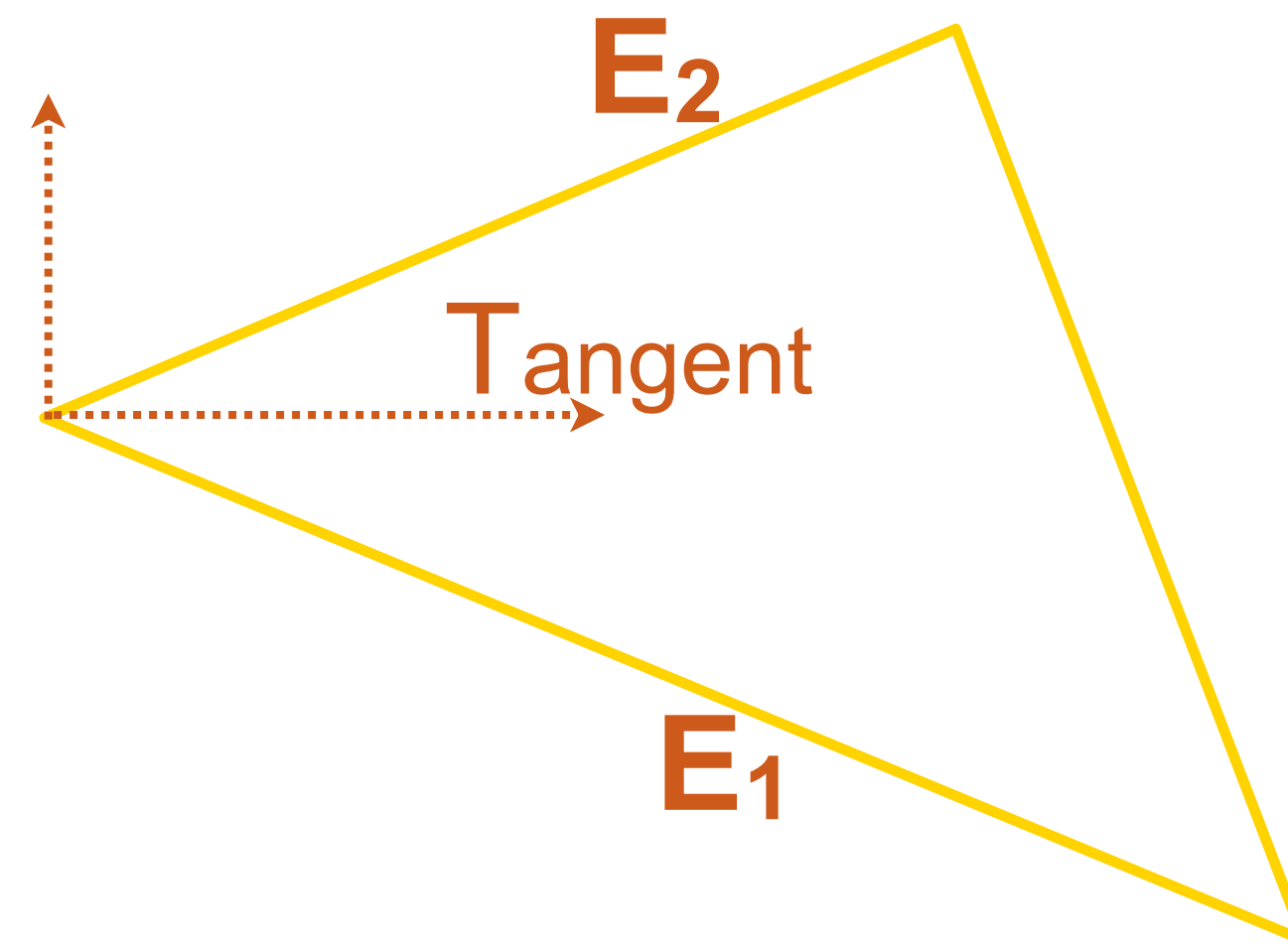


- ➡ Remember to normalize
 - (and range compress)

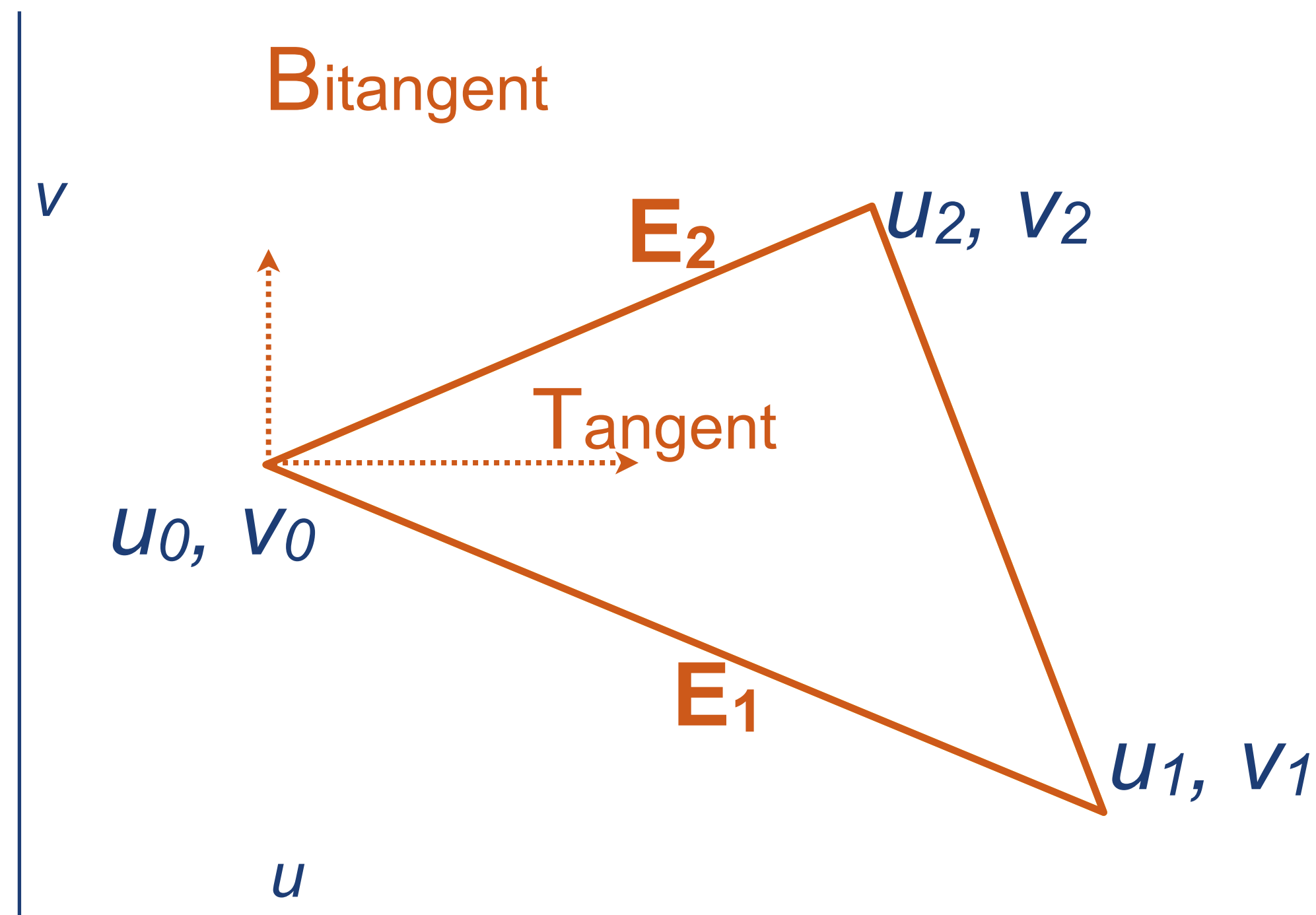
TB(N)



Bitangent



TB(N)



Keep T,B consistent from triangle to triangle



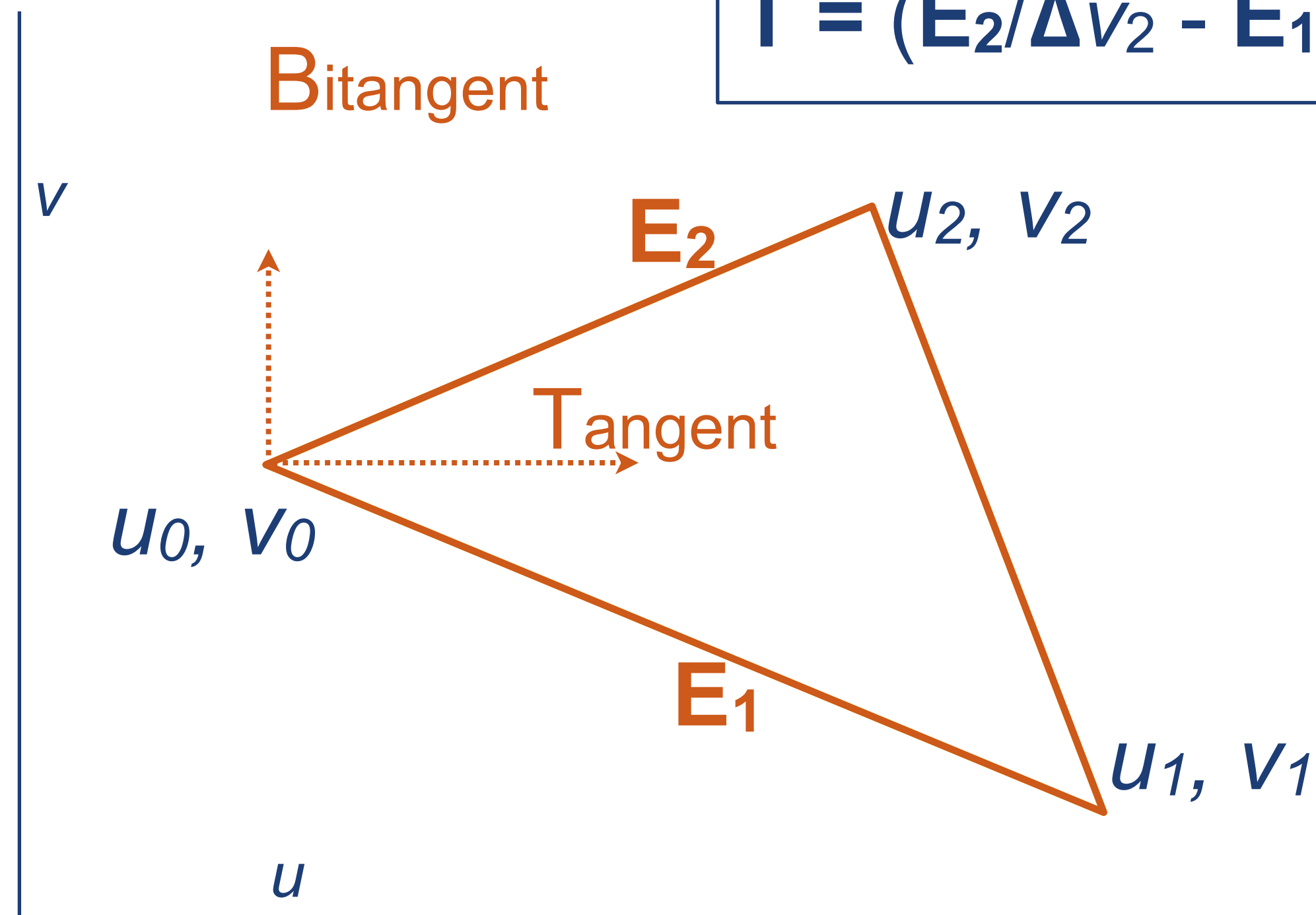
TB(N)

$$\mathbf{E}_2 = (u_2 - u_0)\mathbf{T} + (v_2 - v_0)\mathbf{B} = \Delta u_2 \mathbf{T} + \Delta v_2 \mathbf{B}$$

$$\mathbf{E}_1 = (u_1 - u_0)\mathbf{T} + (v_1 - v_0)\mathbf{B} = \Delta u_1 \mathbf{T} + \Delta v_1 \mathbf{B}$$

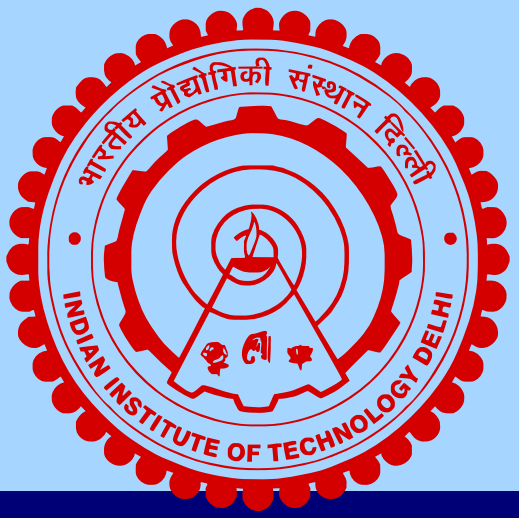
$$\mathbf{B} = (\mathbf{E}_2 / \Delta u_2 - \mathbf{E}_1 / \Delta u_1) / (\Delta v_2 / \Delta u_2 - \Delta v_1 / \Delta u_1)$$

$$\mathbf{T} = (\mathbf{E}_2 / \Delta v_2 - \mathbf{E}_1 / \Delta v_1) / (\Delta u_2 / \Delta v_2 - \Delta u_1 / \Delta v_1)$$



Keep T,B consistent from triangle to triangle

Shade



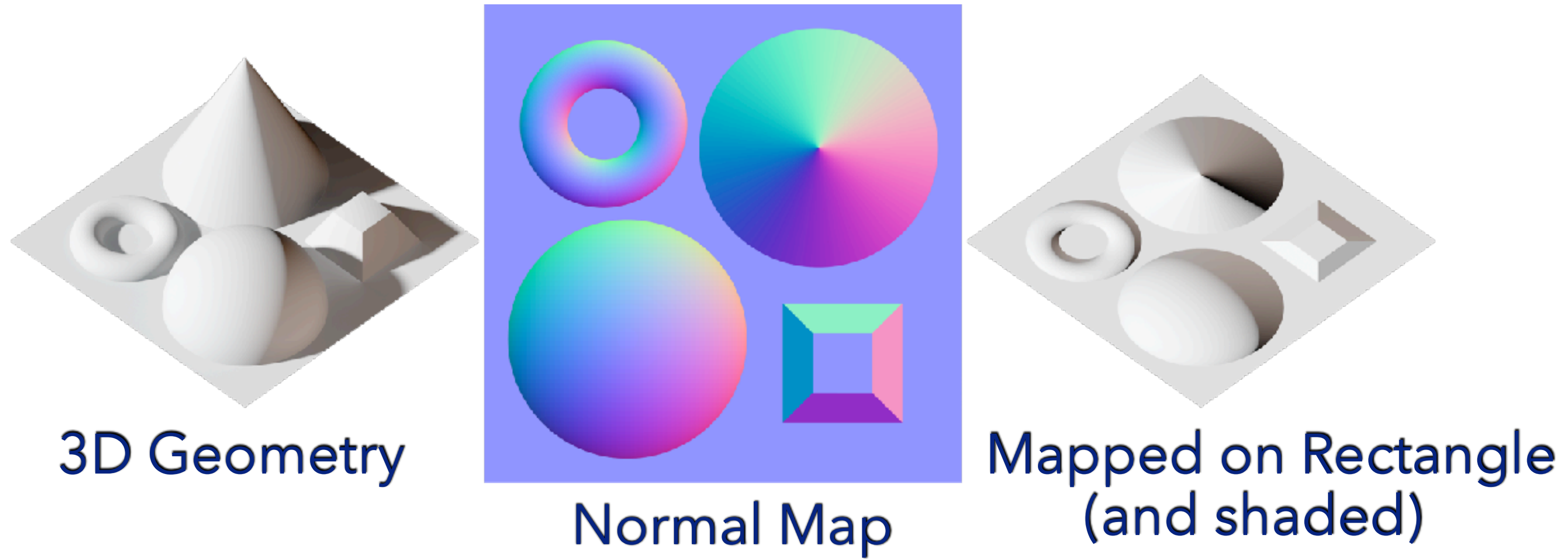
TBN space lighting

```
vec3 MtlDiffuseR = texture(DiffuseTextureSampler, UV).rgb;
vec3 MtlSpecularR = texture(SpecularTextureSampler, UV ).rgb * 0.3;
vec3 n_tbn = normalize(texture(NormalTextureSampler, UV)).rgb*2.0 - 1.0);

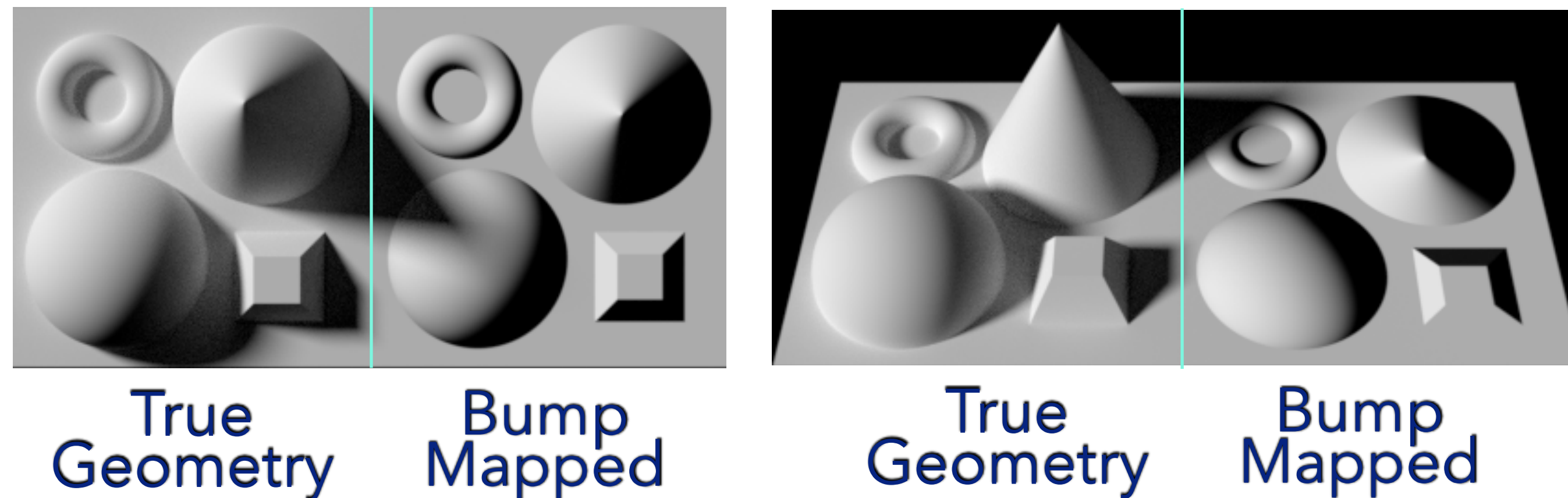
float ldist = length(LightPosition_camera - Position_camera);
float ldistSqr = ldist * ldist; // Square of distance to light
vec3 l_tbn = normalize(LightDirection_tbn); // Direction to the light

float cosTheta = clamp(dot(n_tbn, l_tbn ), 0, 1); // n.l
vec3 E_tbn = normalize(EyeDirection_tbn); // Eye vector
vec3 R_tbn = reflect(-l_tbn, n_tbn); // Ideal direction of reflection
float cosAlphaPow = pow(clamp(dot(E_tbn, R_tbn), 0,1 ), Specularpower);
phongcolor = MtlAmbientColor +
    MtlDiffuseR * LightColor * cosTheta / ldistSqr +
    MtlSpecularR*LightColor * cosAlphapow / ldistSqr;
```

Normal Mapping Example



Rendered
Scenes





Displacement Mapping

- Modifies the surface position
 - ➔ Along surface normal, typically
- Use if Bumps are large
- Get silhouettes, shadows right
- Much more expensive
 - ➔ Tessellate geometry
 - ➔ Supported in modern GPUs

