# Computer Graphics I

## Lecture 8: Geometric parameterization & texturing

**Xiaopei LIU**

School of Information Science and Technology
ShanghaiTech University

# What is a texture?

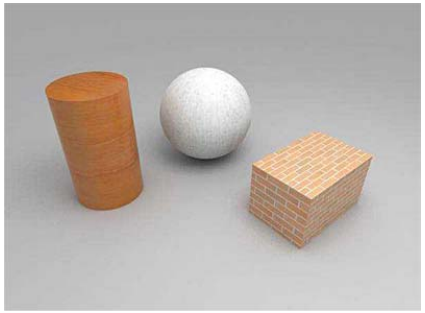
- **How can we add surface details?**




- **Definition**
  - Narrow: surface variations
  - Broad: an image representing surface details

# Why texturing?

- To enrich the patterns on surface



- To reduce computation for geometric details

# Why texturing

- **To provide pre-computed shading**
  - Static scene and lighting

Simple shading

Ambient occlusion texture map

With ambient occlusion

# How can we achieve?

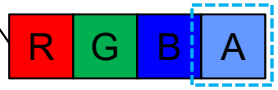
- **Texture mapping**
  - A method for defining high frequency surface detail or color information on a 3D model
  - Pioneered by Edwin Catmull in 1974

- **Texture map**
  - An image applied (mapped) to the surface of a shape or polygon
  - Can be a bitmap image or a procedural texture
  - Can have 1~3 dimensions, but 2D is mostly common

# Texture Map Storage

- **The RGB triplet for each pixel**
  - Sometimes the order may be reversed
  - Sometimes transparency (A) may be included

| Image Data PixelArray [x,y] | | | | | |
|---|---|---|---|---|---|
| Pixel[0,h-1] | Pixel[1,h-1] | Pixel[2,h-1] | ... | Pixel[w-1,h-1] | Padding |
| Pixel[0,h-2] | Pixel[1,h-2] | Pixel[2,h-2] | ... | Pixel[w-1,h-2] | Padding |
| | | | ⋮ | | |
| Pixel[0,9] | Pixel[1,9] | Pixel[2,9] | ... | Pixel[w-1,9] | Padding |
| Pixel[0,8] | Pixel[1,8] | Pixel[2,8] | ... | Pixel[w-1,8] | Padding |
| Pixel[0,7] | Pixel[1,7] | Pixel[2,7] | ... | Pixel[w-1,7] | Padding |
| Pixel[0,6] | Pixel[1,6] | Pixel[2,6] | ... | Pixel[w-1,6] | Padding |
| Pixel[0,5] | Pixel[1,5] | Pixel[2,5] | ... | Pixel[w-1,5] | Padding |
| Pixel[0,4] | Pixel[1,4] | Pixel[2,4] | ... | Pixel[w-1,4] | Padding |
| Pixel[0,3] | Pixel[1,3] | Pixel[2,3] | ... | Pixel[w-1,3] | Padding |
| Pixel[0,2] | Pixel[1,2] | Pixel[2,2] | ... | Pixel[w-1,2] | Padding |
| Pixel[0,1] | Pixel[1,1] | Pixel[2,1] | ... | Pixel[w-1,1] | Padding |
| Pixel[0,0] | Pixel[1,0] | Pixel[2,0] | ... | Pixel[w-1,0] | Padding |

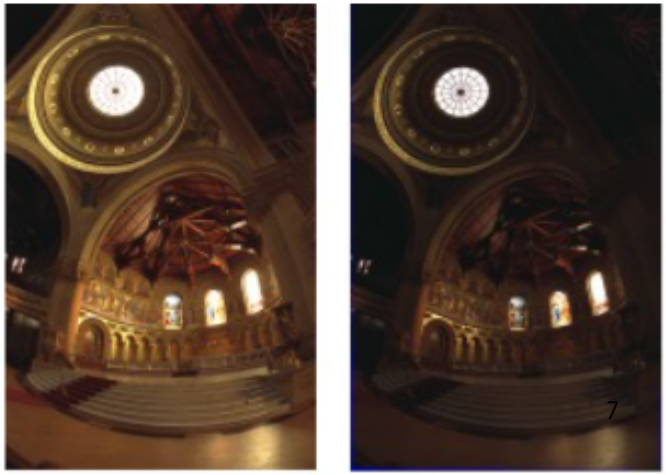


# High dynamic range image

- Problem: ratio of brightness object to darkest object in real-world scenes can be quite large
  - Human eye can discern ratio of 100,000:1 (even more if accounting for adaptation)
- High-dynamic range (HDR) image: encodes large range of luminance (or lightness) values
  - Common format: 16-bits per channel EXR (see environment maps in Asst. 3)
- Modern camera senses can only sense much narrower range of luminances (e.g., 12-bit pixels)
- But most modern displays can only display a much narrower range of luminances
  - Luminance of white pixel / luminance of black pixel for a high-end LCD TV ~ 3000:1 *

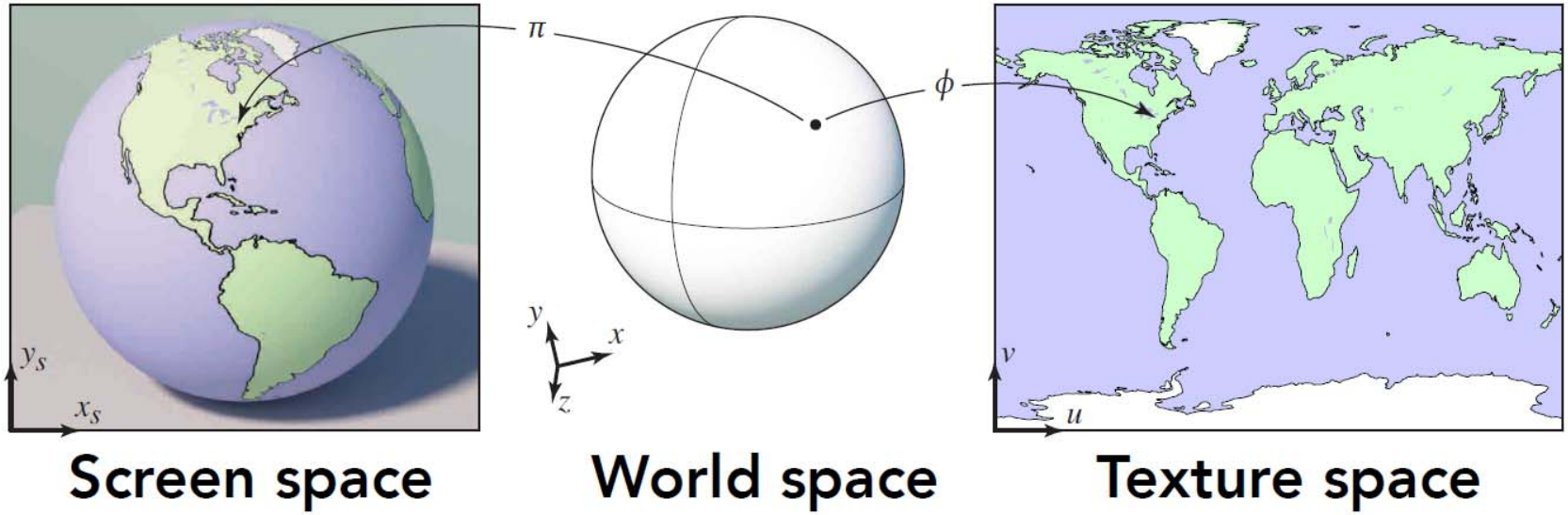
Overexposed (loss of detail in brightest areas since they are clamped to 1)

Underexposed (detail remains in brightest areas, but large regions of image clamped to 0)



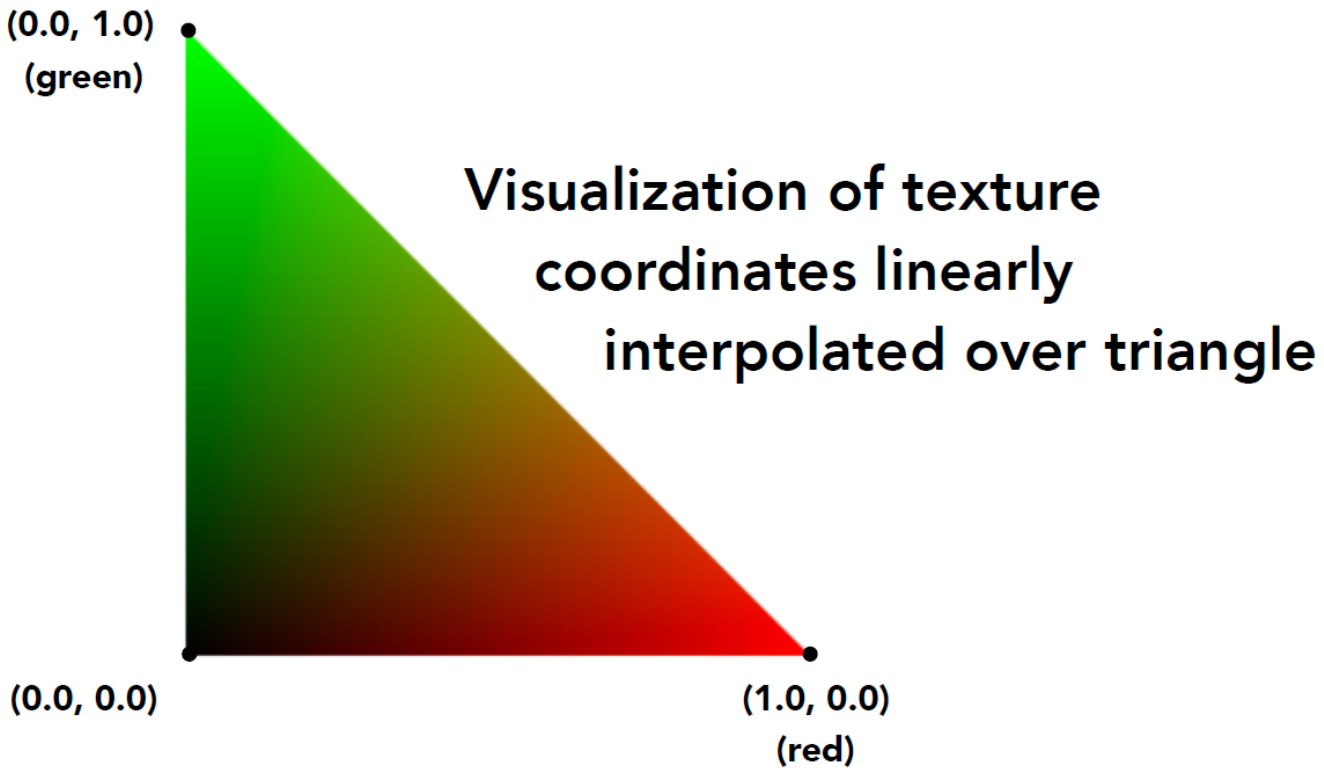
# Texture coordinate mapping

- **Surface lives in 3D world space**
  - Every point also has a place where it goes in the image and in the texture



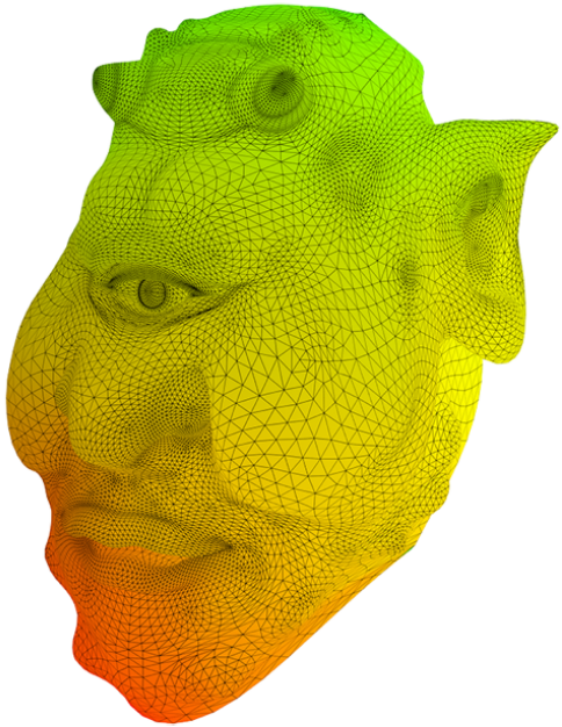
# 2D texture mapping

- **Each surface point is assigned a texture coordinate (u,v)**

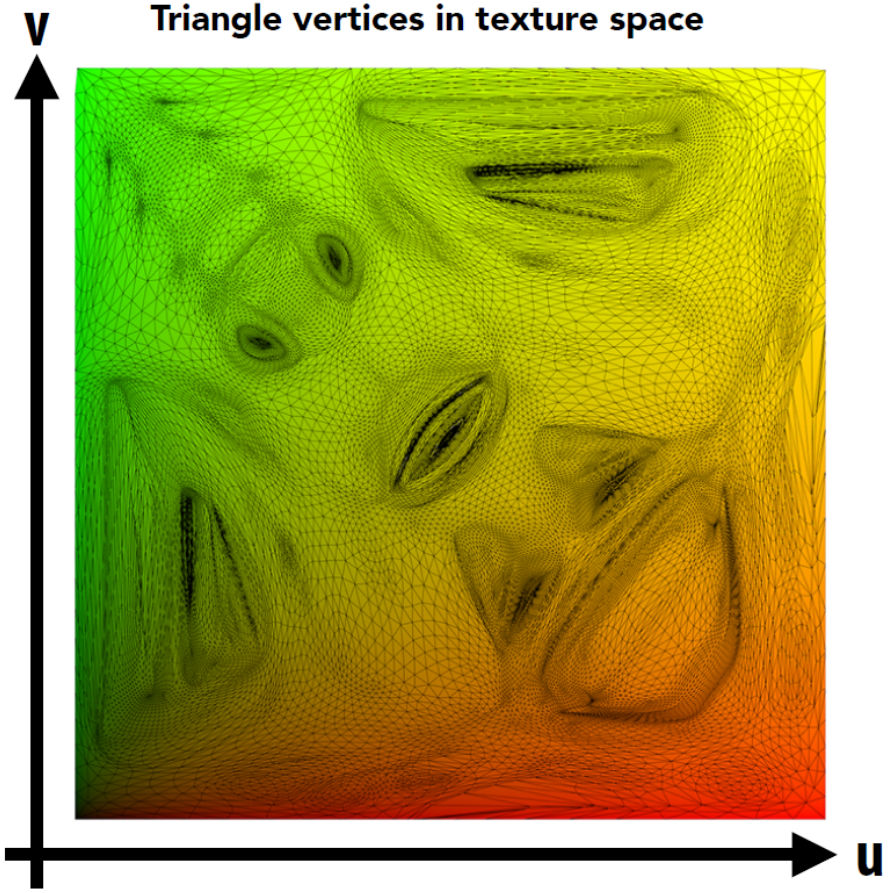


# Visualization of texture coordinates

Visualization of texture coordinates

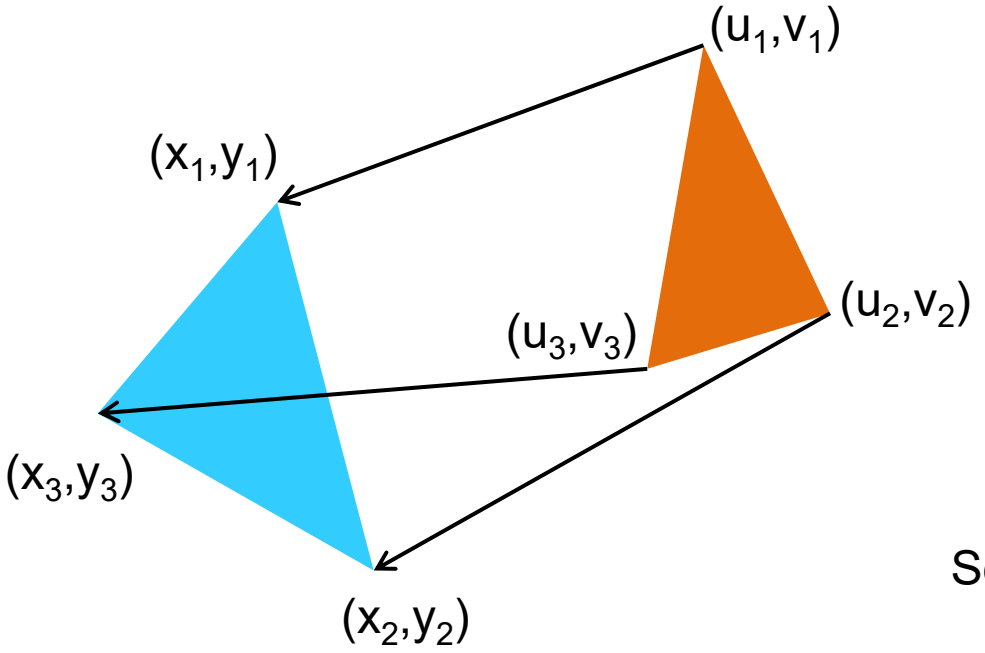


Triangle vertices in texture space



# 2D texture mapping

- **How to perform texture mapping?**
  - Construct the texture projection matrix
  - Mapping between projected vertices and texture coordinates



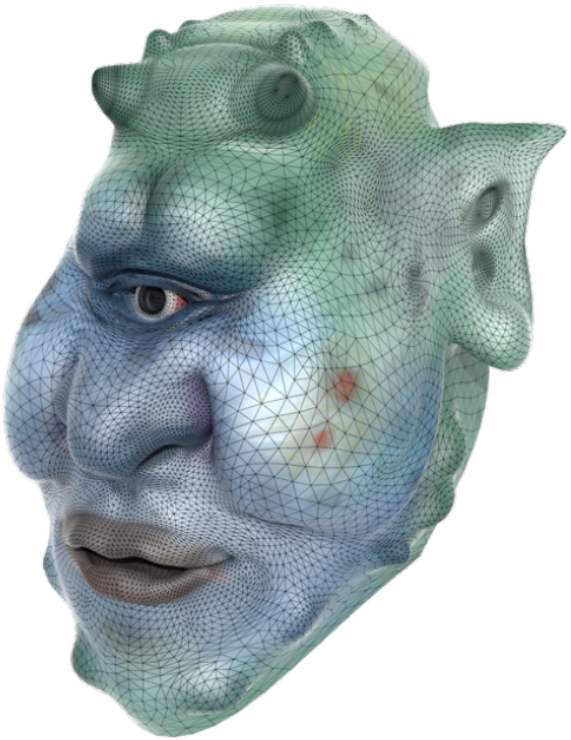

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

0 0 1

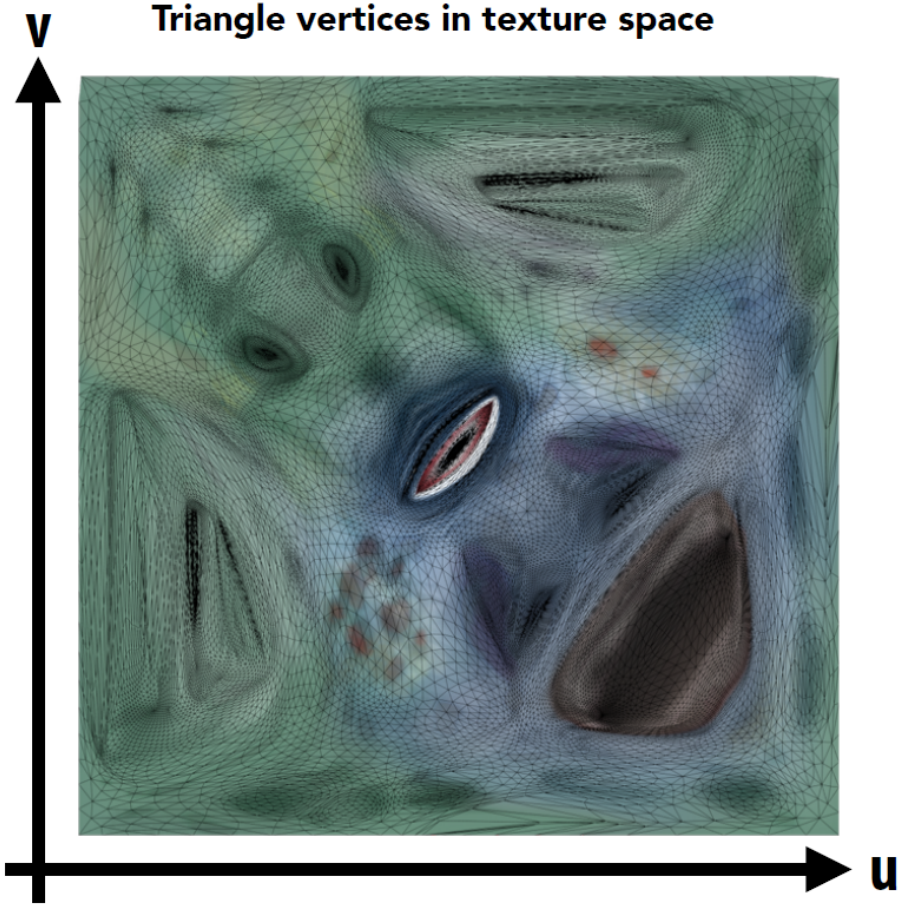
Solve a linear system for a triangle pair

# Image texture applied to a surface

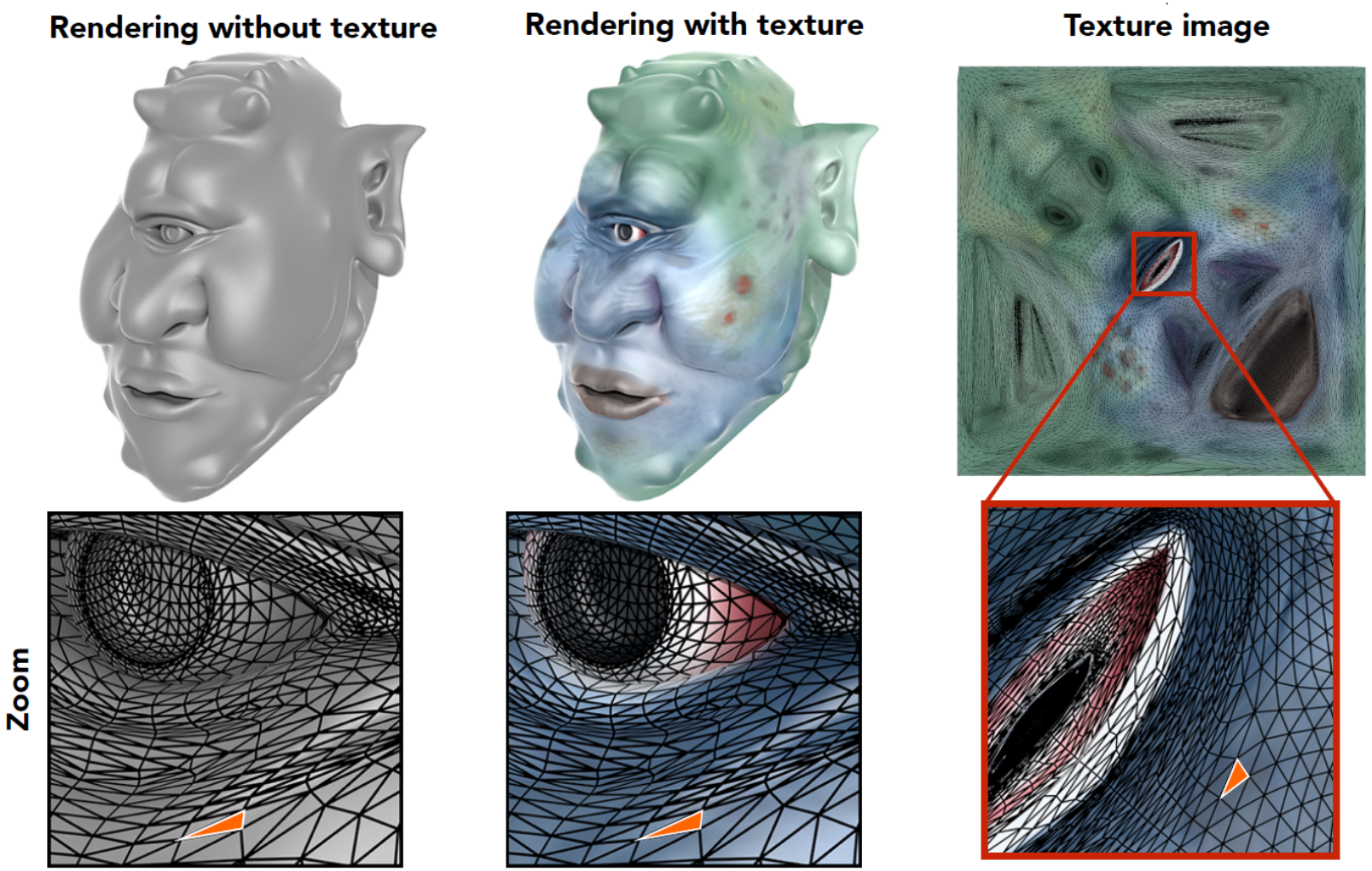
Rendered result



Triangle vertices in texture space



# Image texture applied to a surface

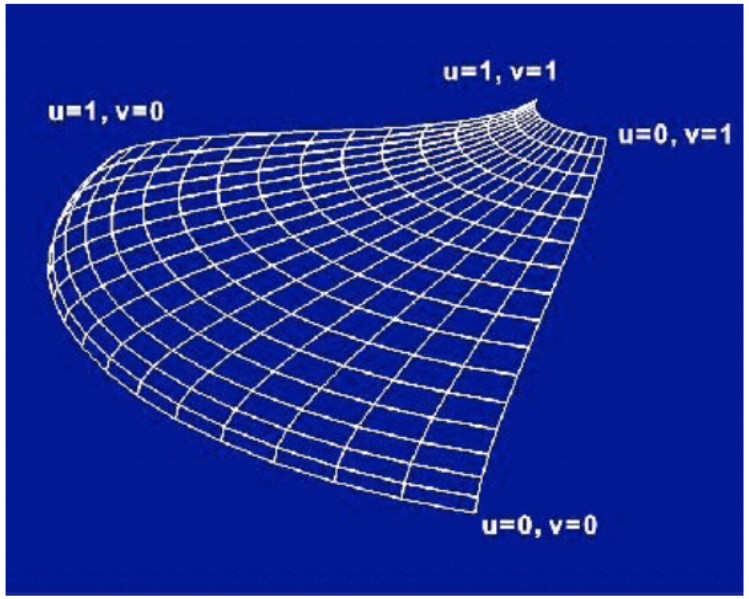


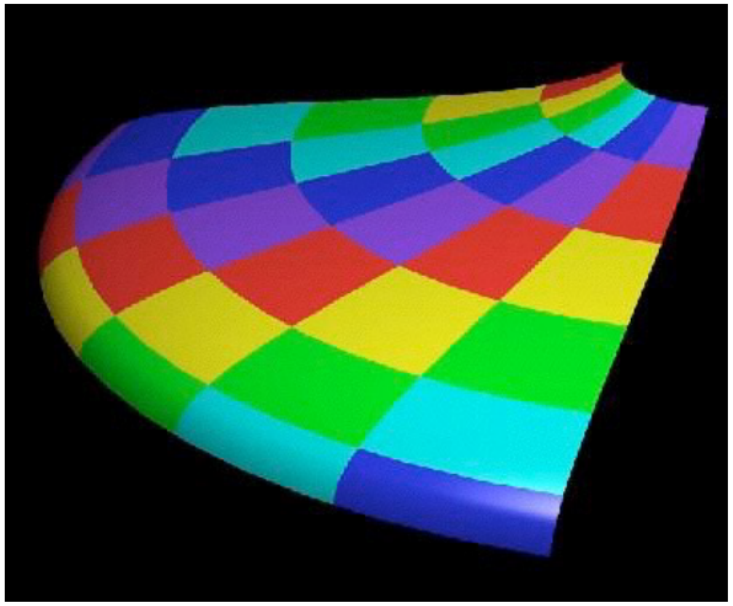
# Surface parameterization

- **Determine the mapping (projection)**
  - Between any point on surface and point on texture
- **Texture coordinate functions**
  - Specify how to do projection
  - Projection from different domains

# Examples of Texture Coordinate Functions

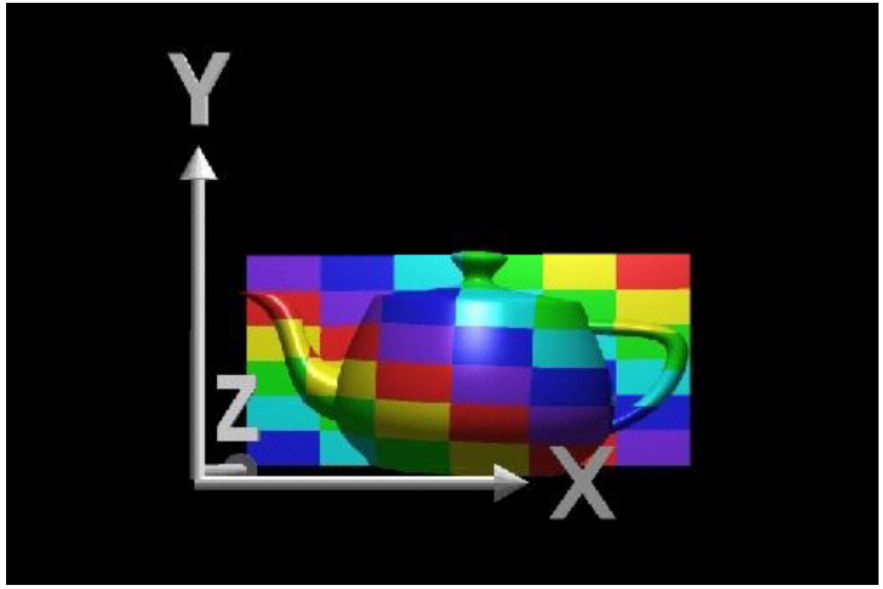
- **A parametric surface (e.g. spline patch)**
  - Use parameter space coordinates as texture coordinates directly

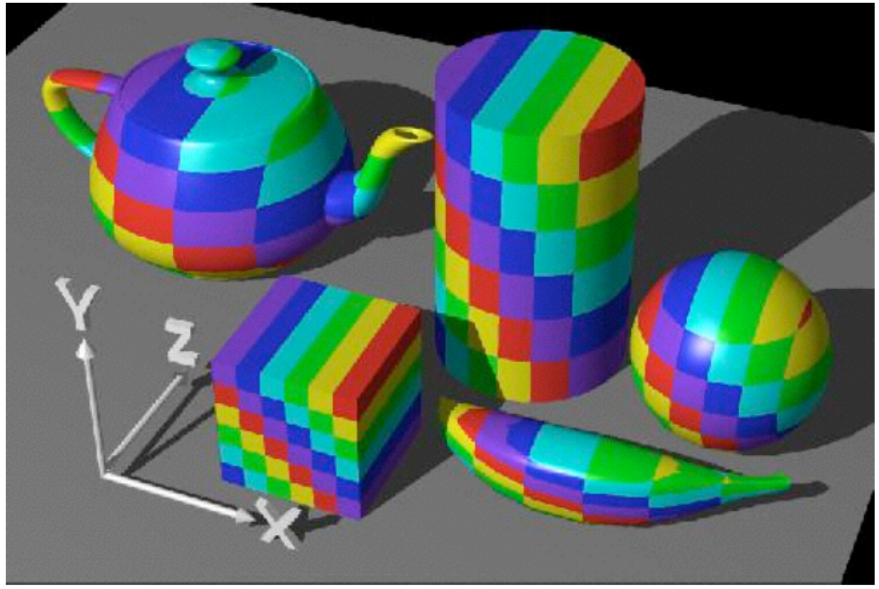


# Examples of texture coordinate functions

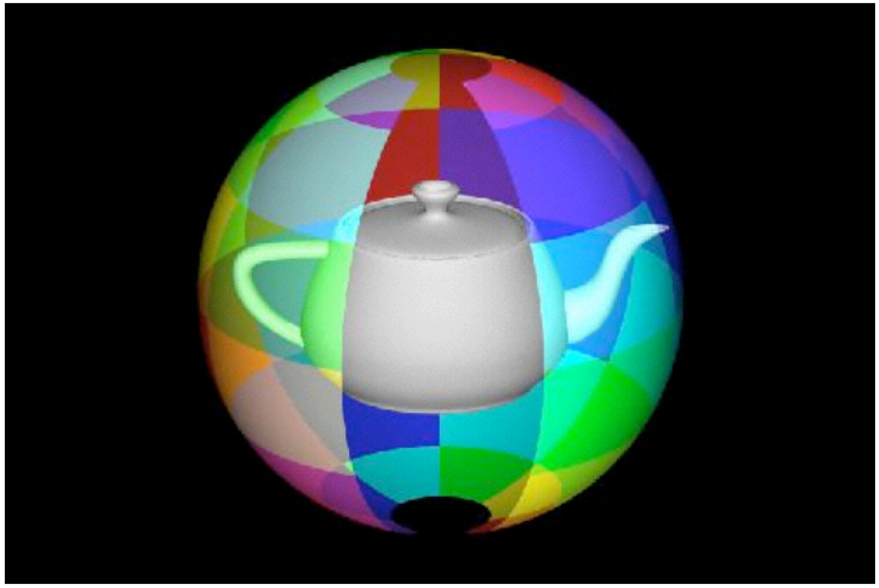
- **Planar projection**
  - Distortion in areas which are not parallel to the texture plane

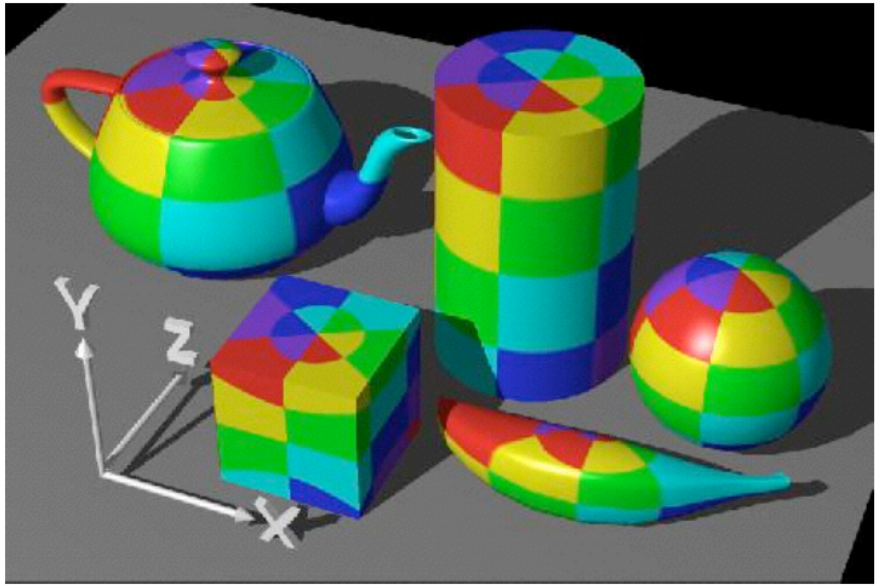


# Examples of texture coordinate functions

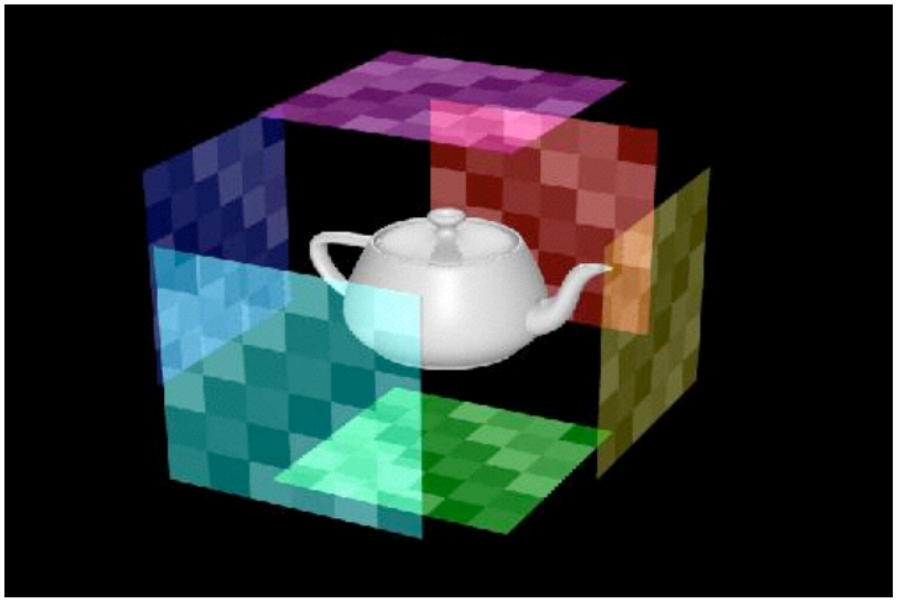
- **Spherical projection**
  - Distortion in areas which are not parallel to the local texture tangent plane

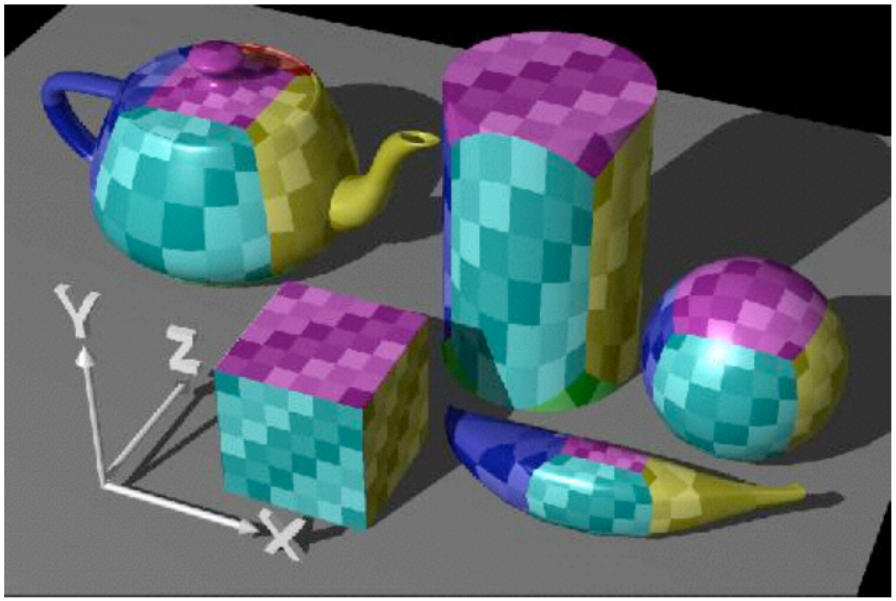




# Examples of texture coordinate functions

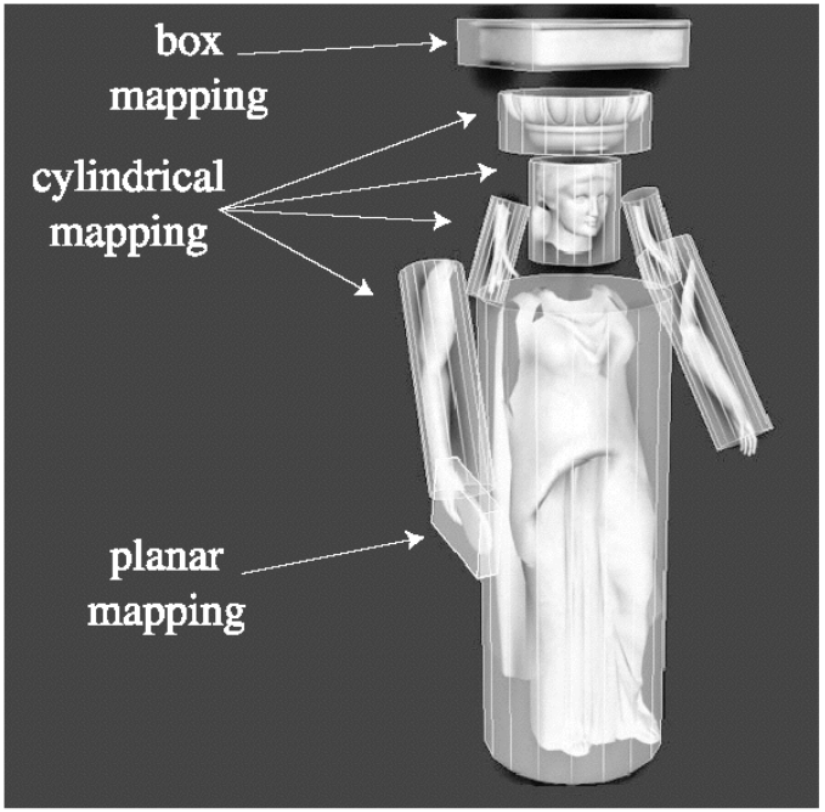
- **Cube map projection**
  - Distortion in areas which are not parallel to the multiple texture planes





# Examples of texture coordinate functions

- **Complex surfaces**
  - Try to find simple geometric objects (cylinders, boxes) to lay textures and project to different parts



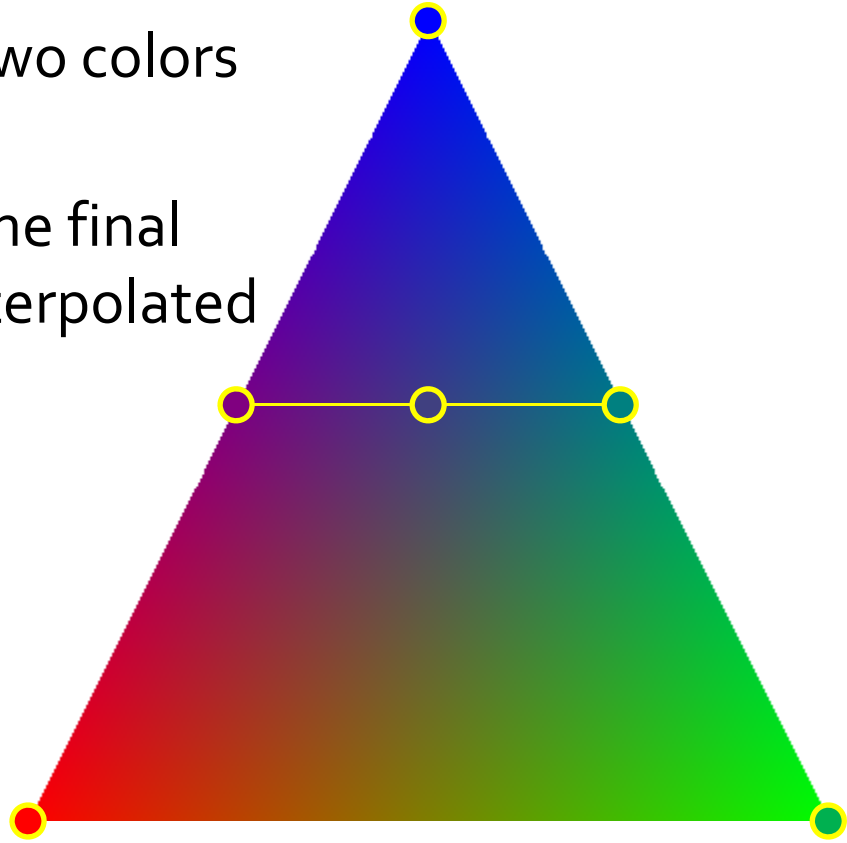
# Computing texture coordinates

- **For any vertex/projected pixel, we need to compute a texture coordinate**
  - Consider a general mesh
  - At each face (mostly triangle), a texture-space mapping is specified (computed) at vertices
  - For any pixel, we need to interpolate within a triangle to obtain texture coordinate

- **How can we do?**
  - Barycentric coordinate

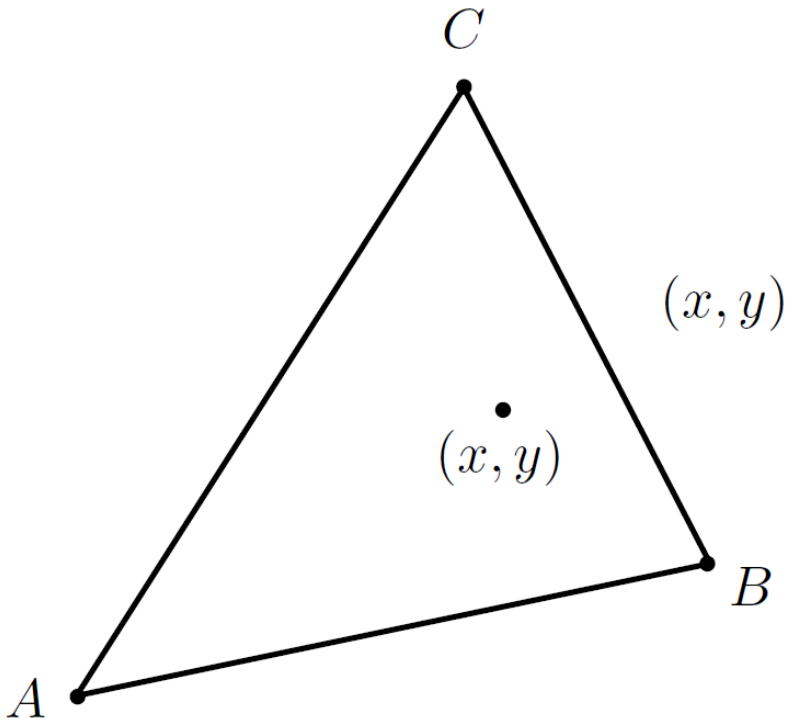
# Recall how we  interpolate color?

- **How to fill the color of the pixels inside the triangle region?**
  - Linearly interpolate two colors along two edges
  - Linearly interpolate the final color based on the interpolated two colors



# Recall on barycentric coordinates

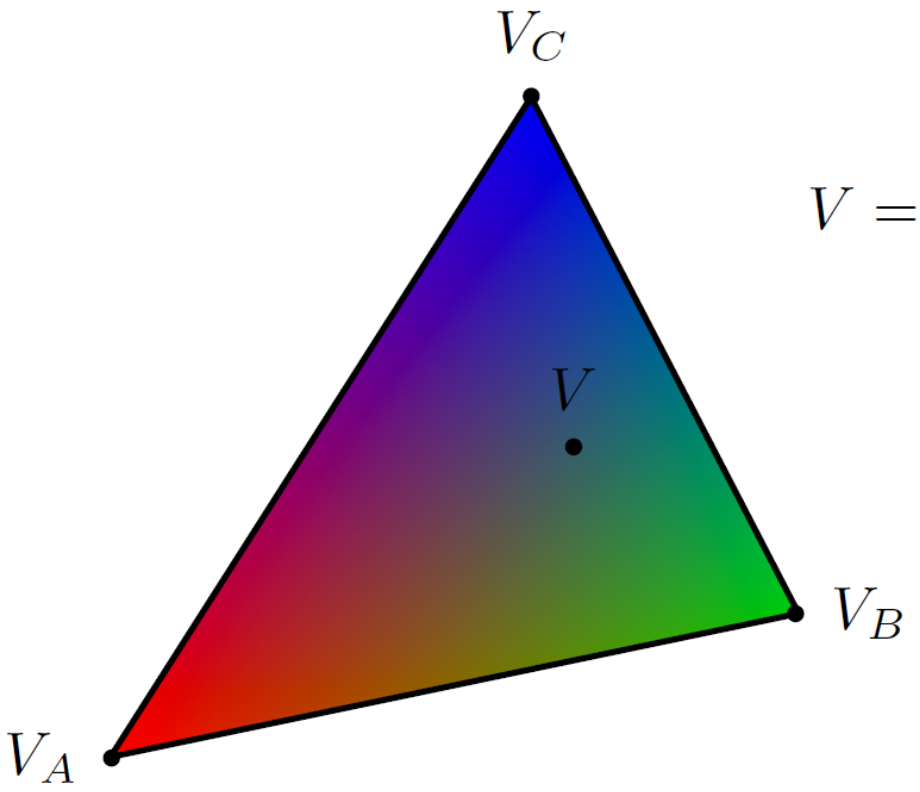
- **A coordinate system for triangles** (α,β,γ)




$$(x, y) = \alpha A + \beta B + \gamma C$$
$$\alpha + \beta + \gamma = 1$$

# Recall on barycentric coordinates

- **Barycentric coordinates linearly interpolate values at vertices**

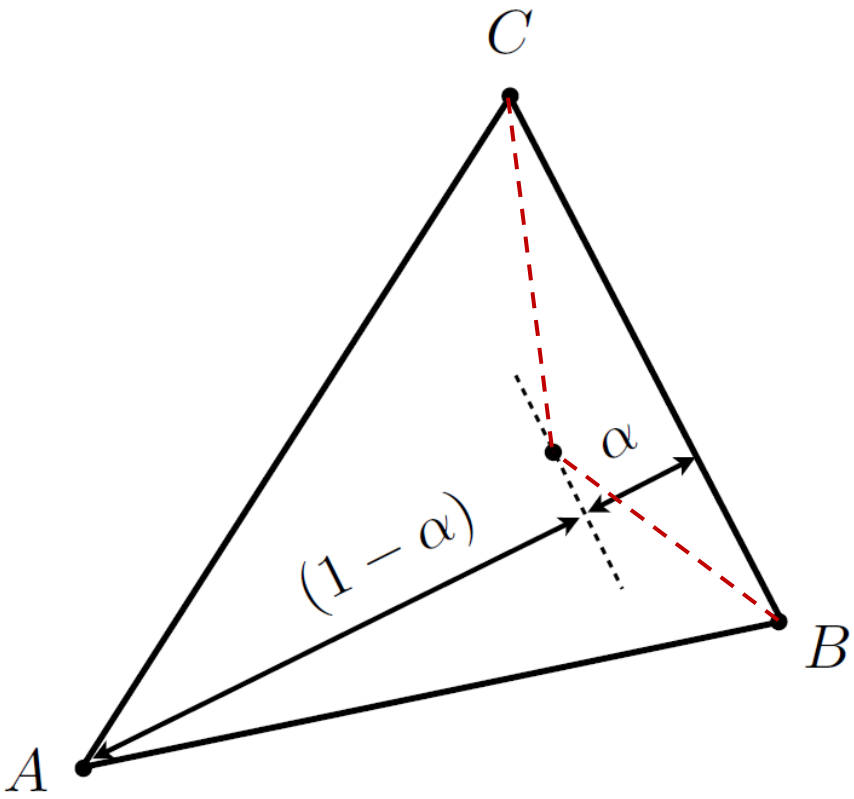



$$V = \alpha V_A + \beta V_B + \gamma V_C$$

**$V_A$, $V_B$, $V_C$ can be positions, texture coordinates, color, normal vectors, material attributes…**

# Computing barycentric coordinates

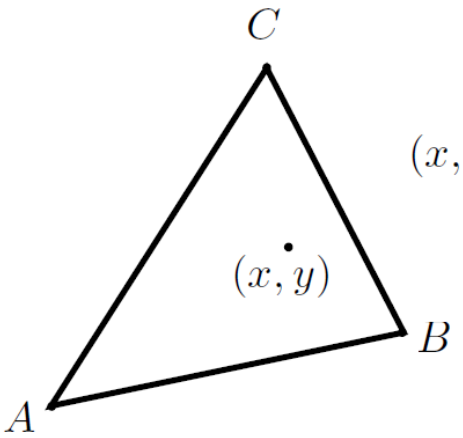
- **Geometric viewpoint — proportional distances**




**Similar construction for other coordinates**

# Computing barycentric coordinates

- **Barycentric coordinate formulas**




$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$
$$\alpha + \beta + \gamma = 1$$

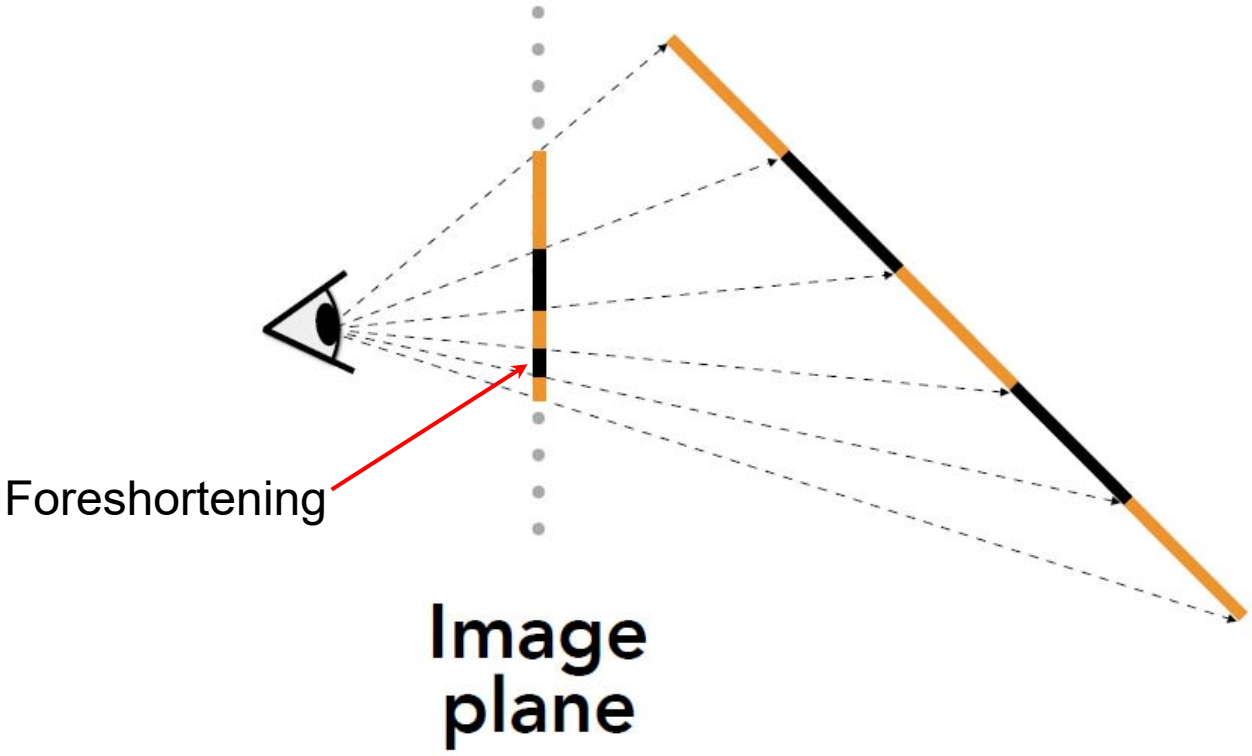
$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

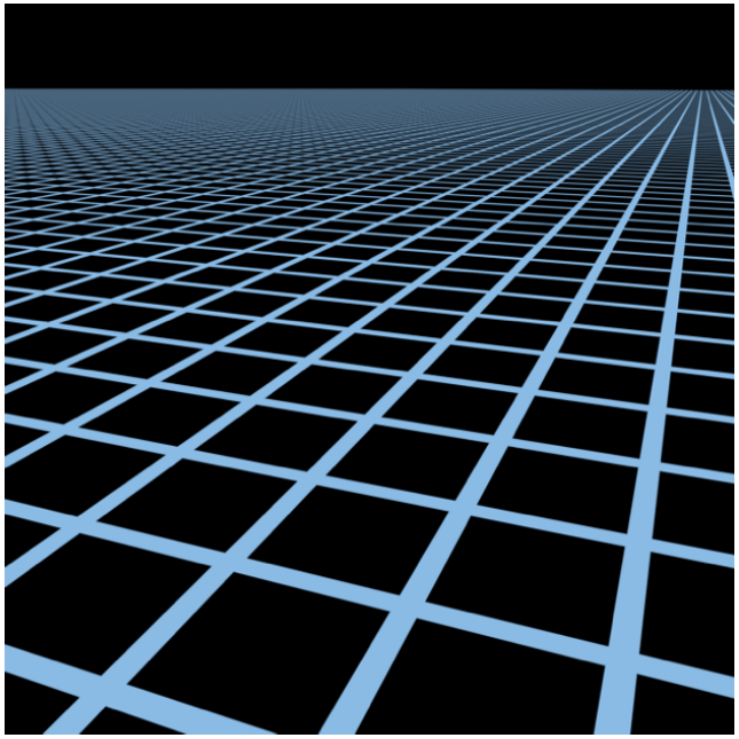
# Influence of perspective projection

- **Linear interpolation in world coordinates yields nonlinear interpolation in screen coordinates**

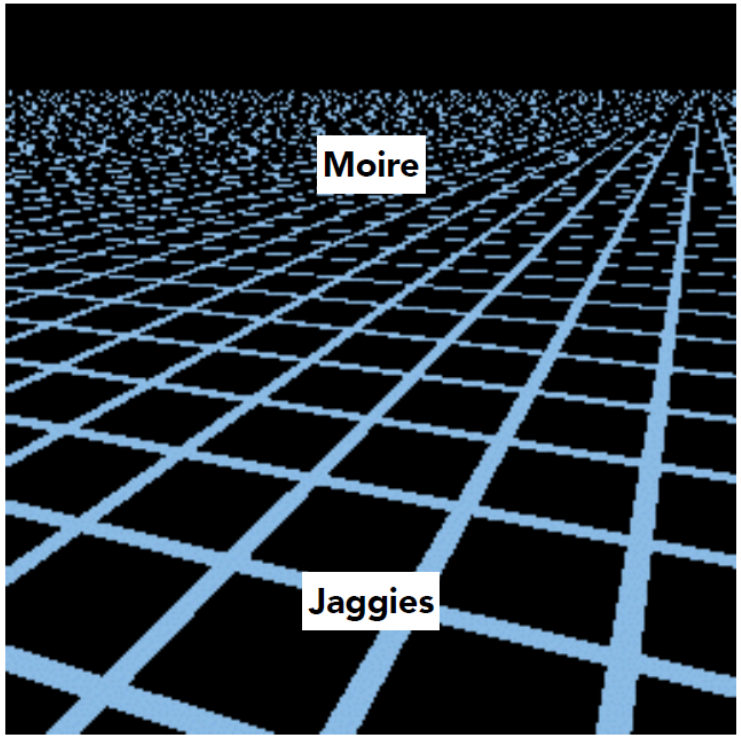


# Sampling textures

- **Point sampling textures**



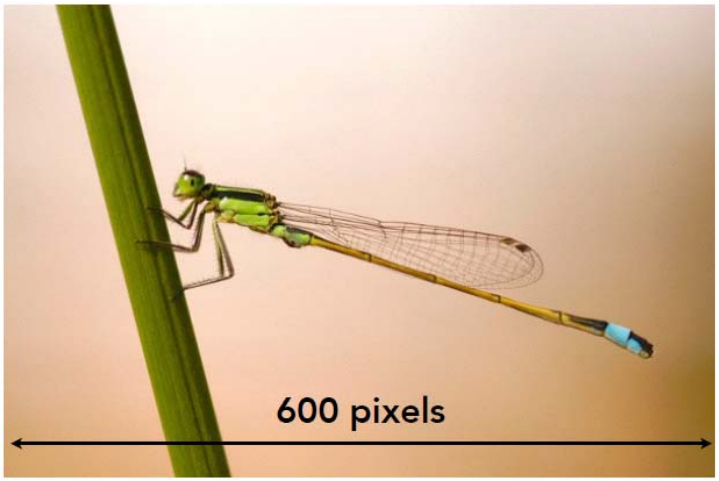
High-res reference

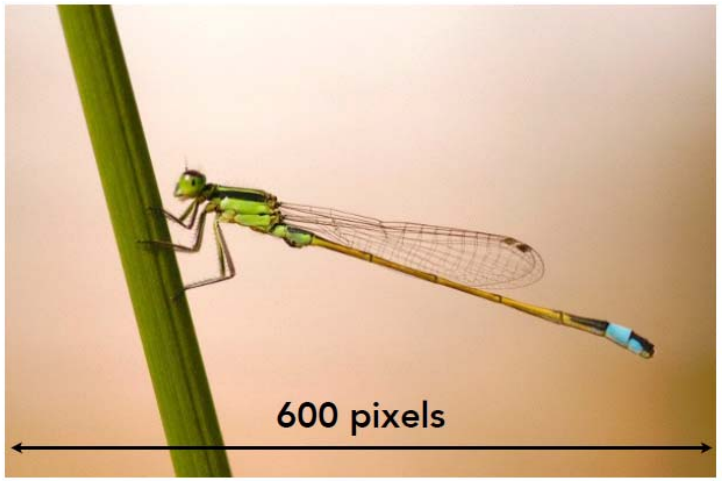


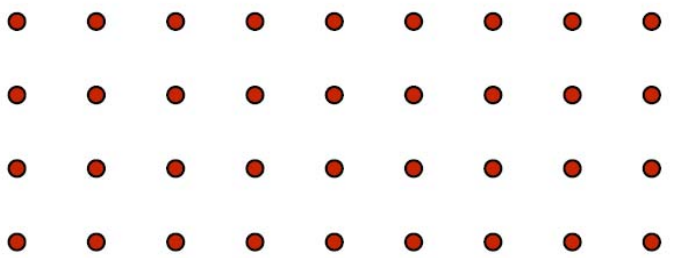

Point sampling

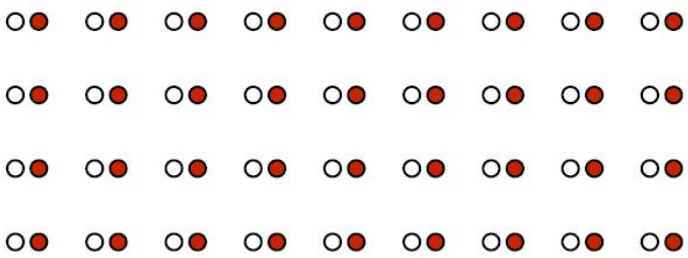
# Sampling textures

- **Sampling rate on screen v.s. texture**



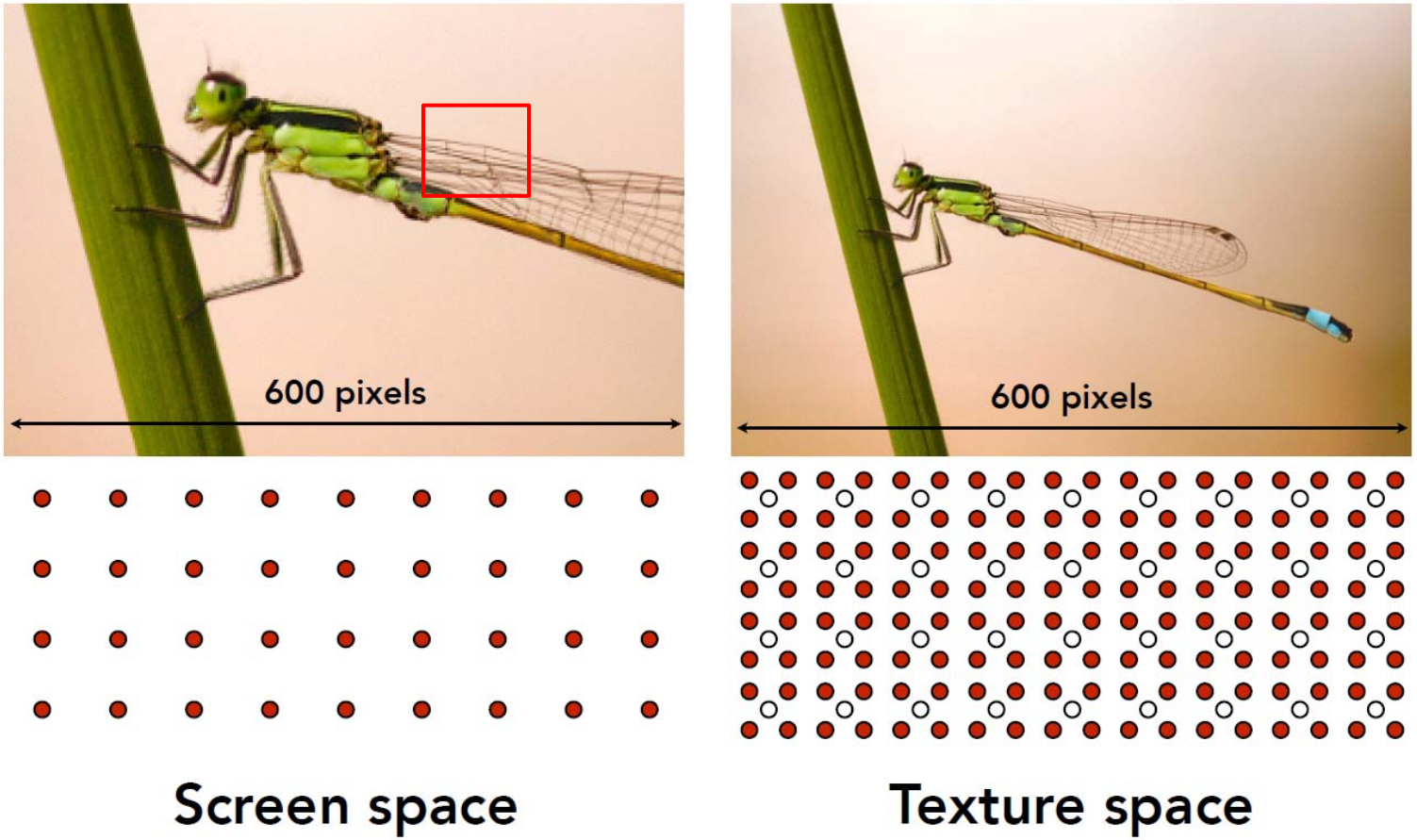


Screen space

Texture space

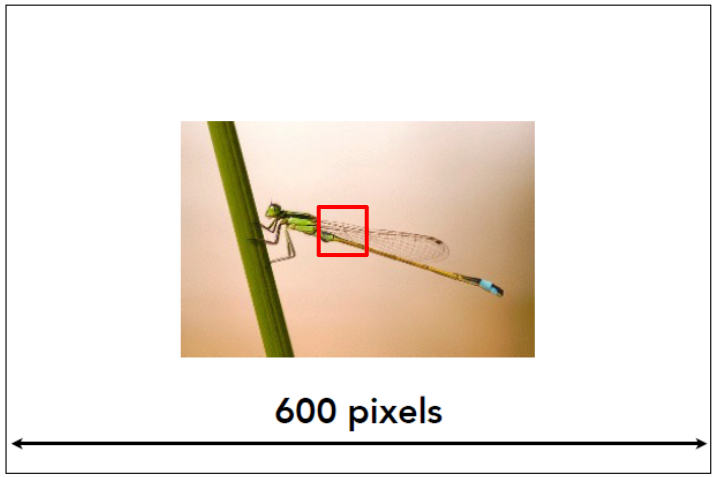
# Sampling textures

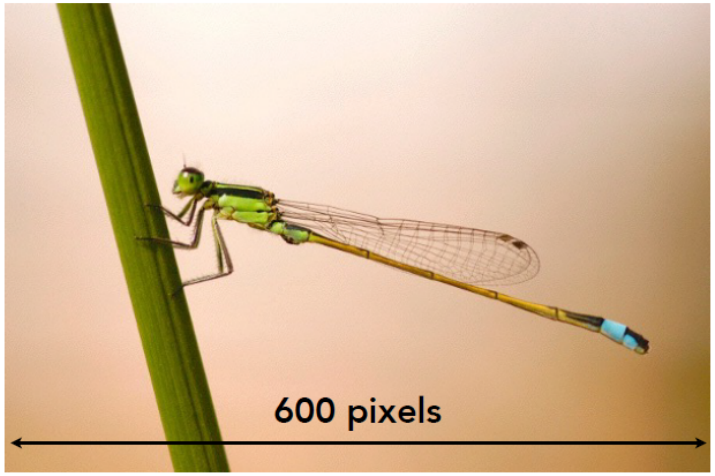
- **Sampling rate on screen v.s. texture**
  - Oversampling (magnify)

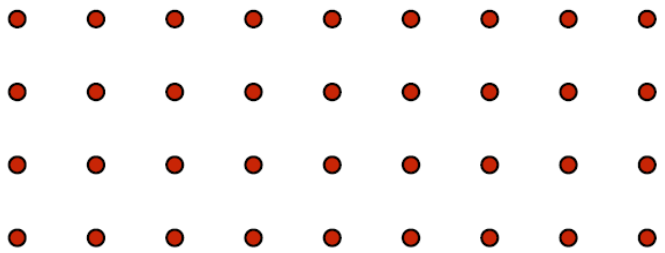


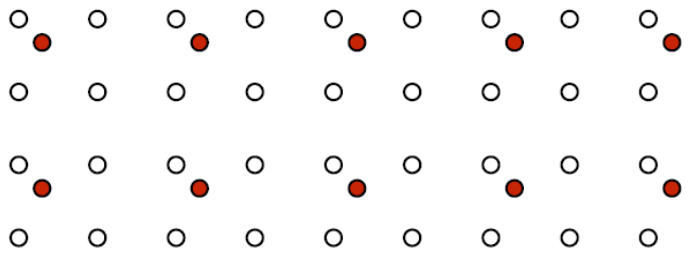
# Sampling textures

- **Sampling rate on screen v.s. texture**
  - Undersampling (minify)





Screen space

Texture space

# Texture sampling rate

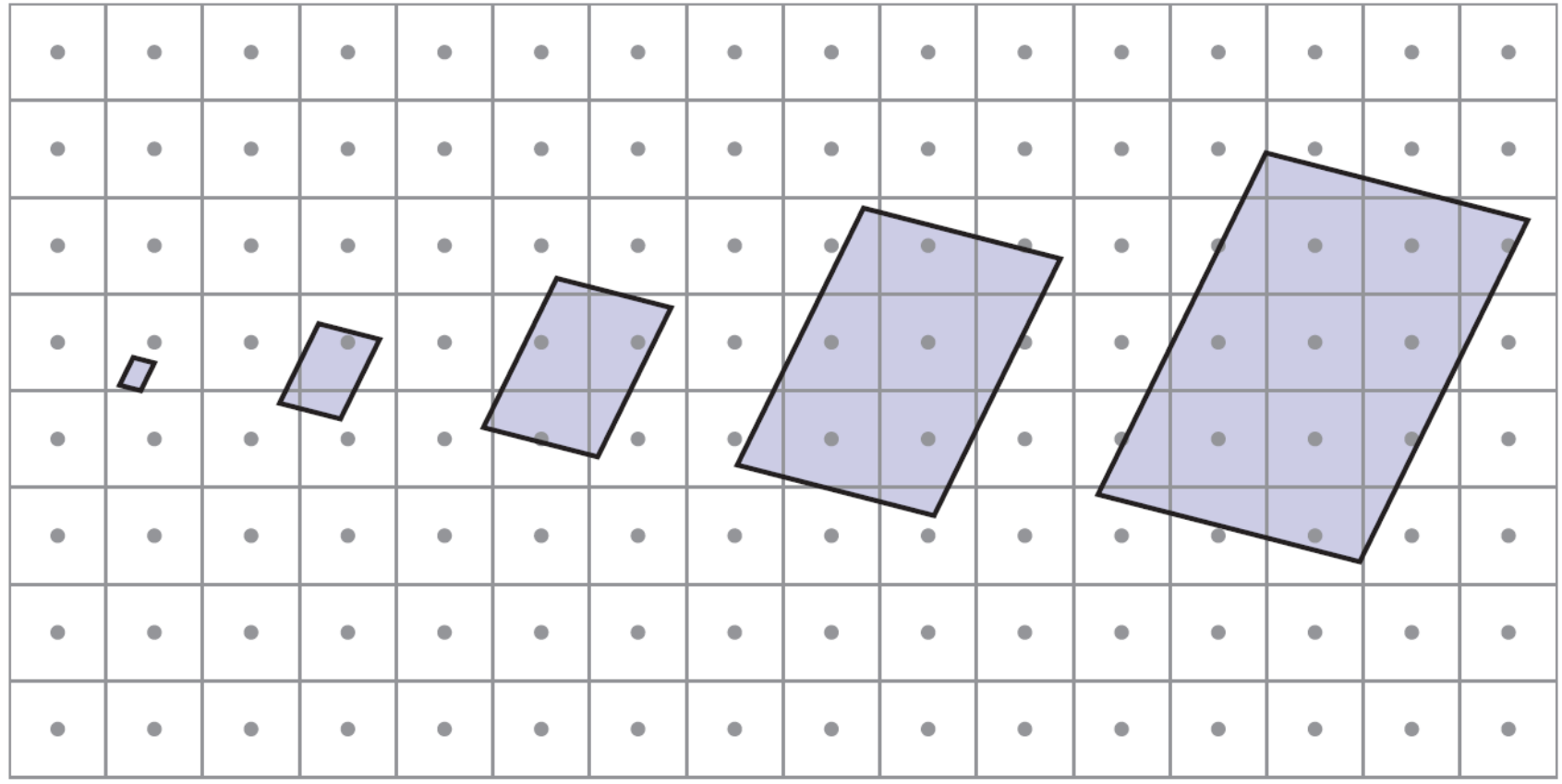
- **The sampling frequency in screen space**
  - Translate to a sampling frequency in texture space
  - Determined by how mapping is done

- **In general**
  - The frequency varies across the scene
  - Depending on
    - Geometric transforms
    - Viewing transforms
    - Texture coordinate function

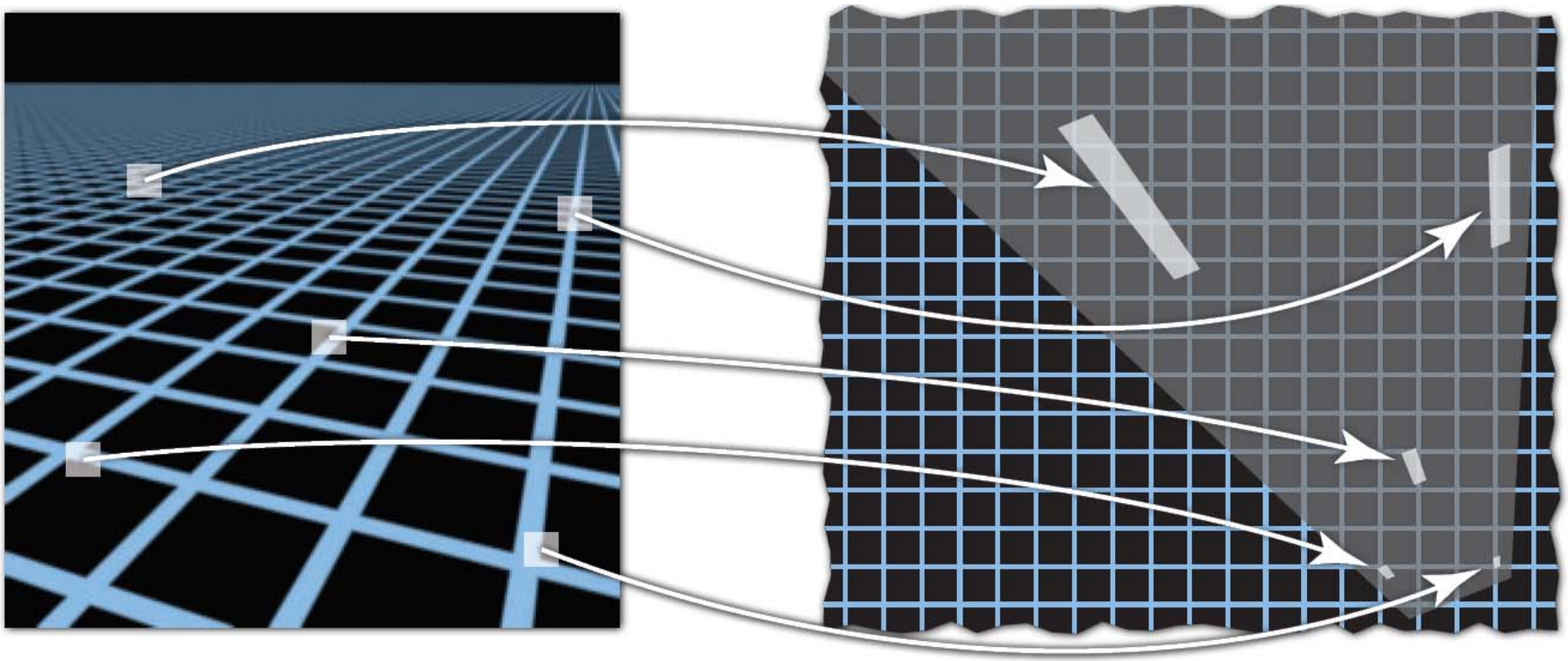
# Screen pixel area v.s. texel area

- **At optimal viewing size**
  - 1:1 mapping between pixel area and texel area
  - Depending on texture resolution
- **When larger (magnification)**
  - Each pixel is a small part of the texel
- **When smaller (minification)**
  - Each pixel could include many texels

# Screen pixel footprint in texture



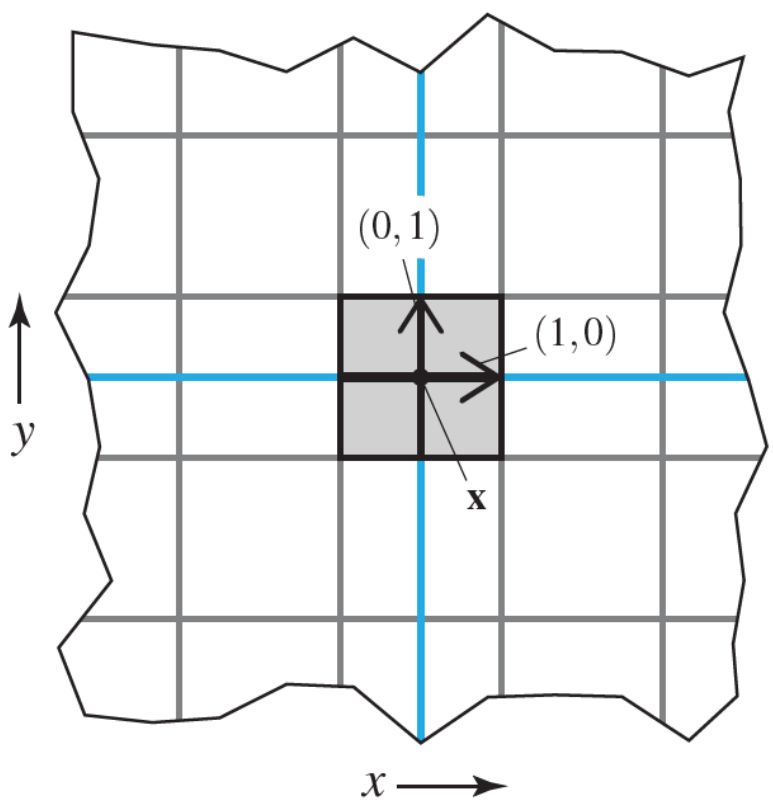
# Screen pixel footprint in texture



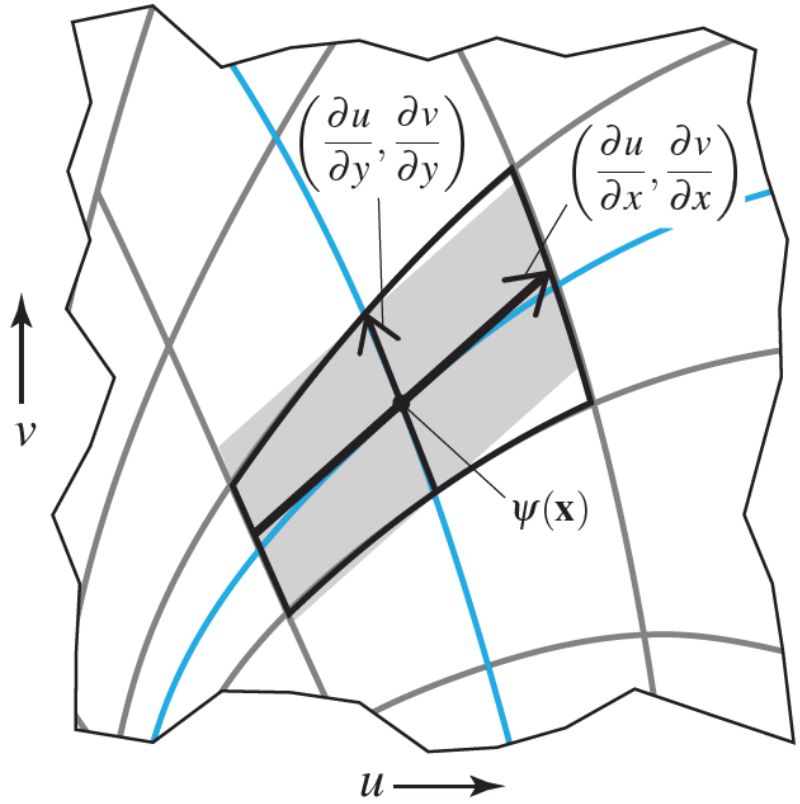
Screen space

Texture space

# Estimating footprint area with Jacobian



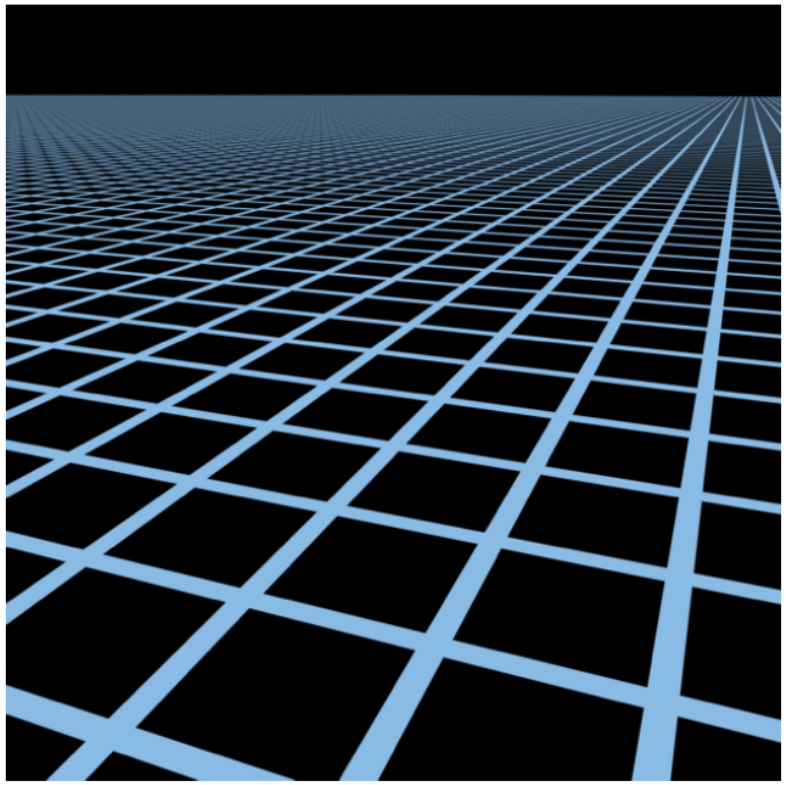

Screen space



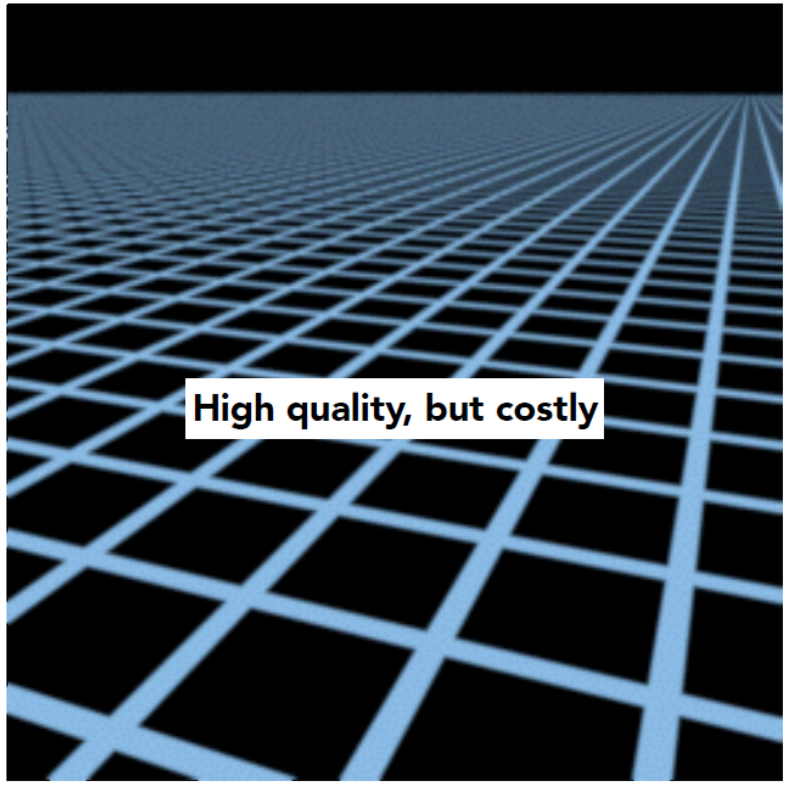

Texture space

# Texture antialiasing

- **Will super-sampling anti-alias?**



High-res reference



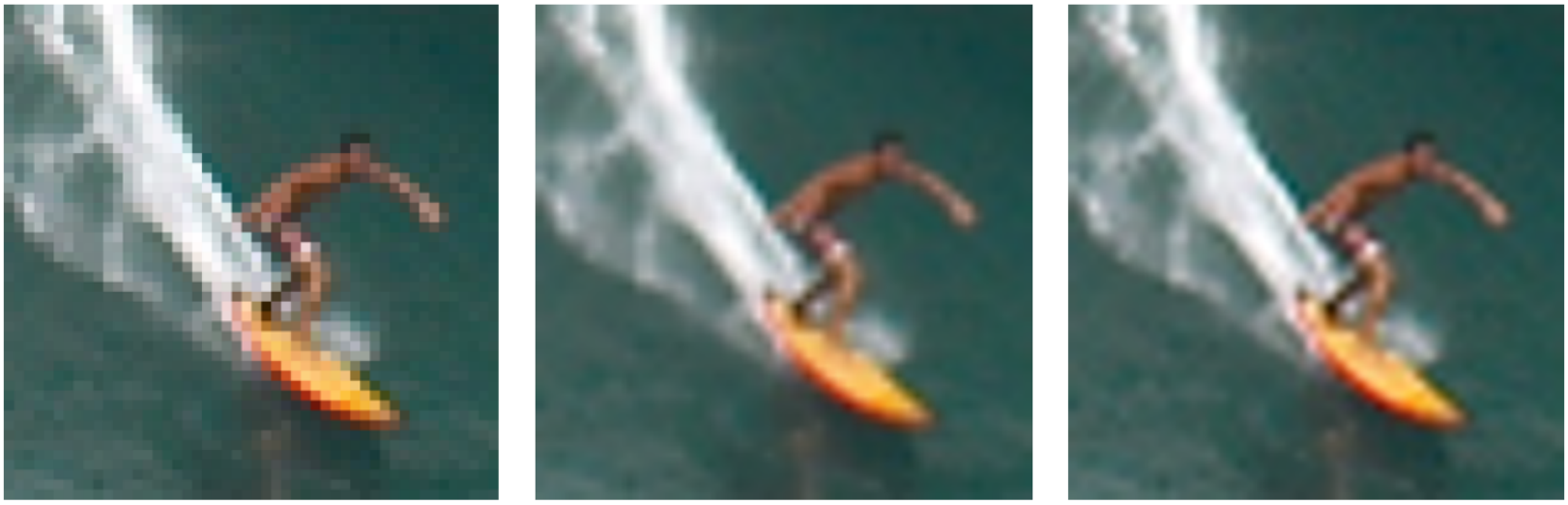

512x supersampling

# Texture antialiasing

- **Will super-sampling anti-alias?**
  - Yes, high quality, but costly
  - When highly minified, many texels in pixel footprint
- **Efficient texture anti-aliasing**
  - Want antialiasing with one/few texels per pixel
  - How? Antialiasing = filtering before sampling!

# Texture filtering

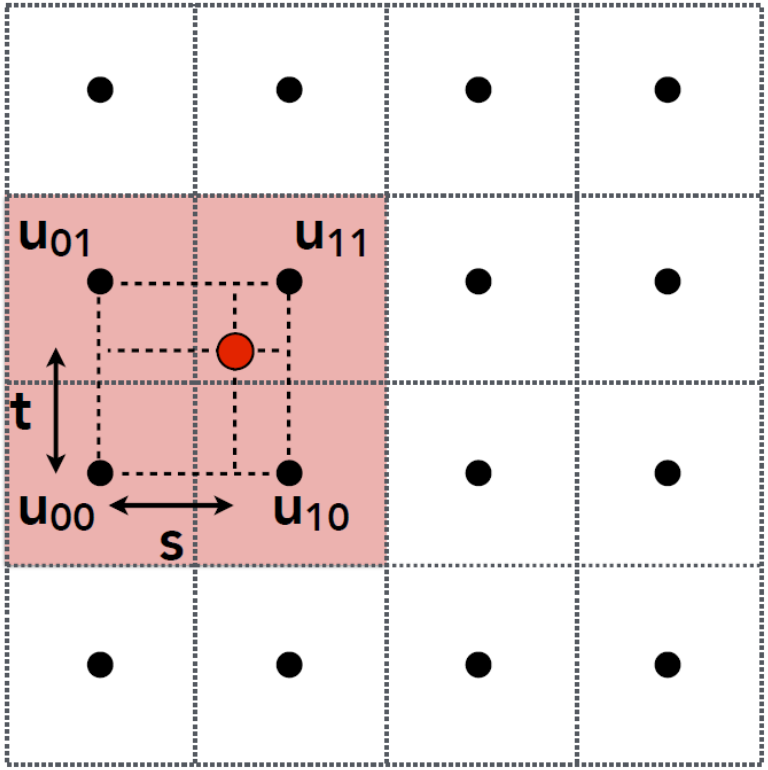
- **Texture magnification - easy case**
  - This is image interpolation
  - Generally don't want this — insufficient resolution



Box (nearest) Bilinear Bicubic

# Texture filtering

- **Bilinear filtering**




$$\mathrm{lerp}(x, v_0, v_1) = v_0 + t(v_1 - v_0)$$

$$u_0 = \mathrm{lerp}(s, u_{00}, u_{10})$$
$$u_1 = \mathrm{lerp}(s, u_{01}, u_{11})$$
$$u = \mathrm{lerp}(t, u_0, u_1)$$

# Texture filtering

- **Texture minification – hard case**
  - Challenging
    - Many texels can contribute to pixel footprint
    - Shape of pixel footprint can be complex
  - Idea
    - Low-pass filter and down-sample texture
    - Use resolution that matches screen sampling rate

# Texture filtering

- **Level 0 - full resolution texture**

# Texture Filtering

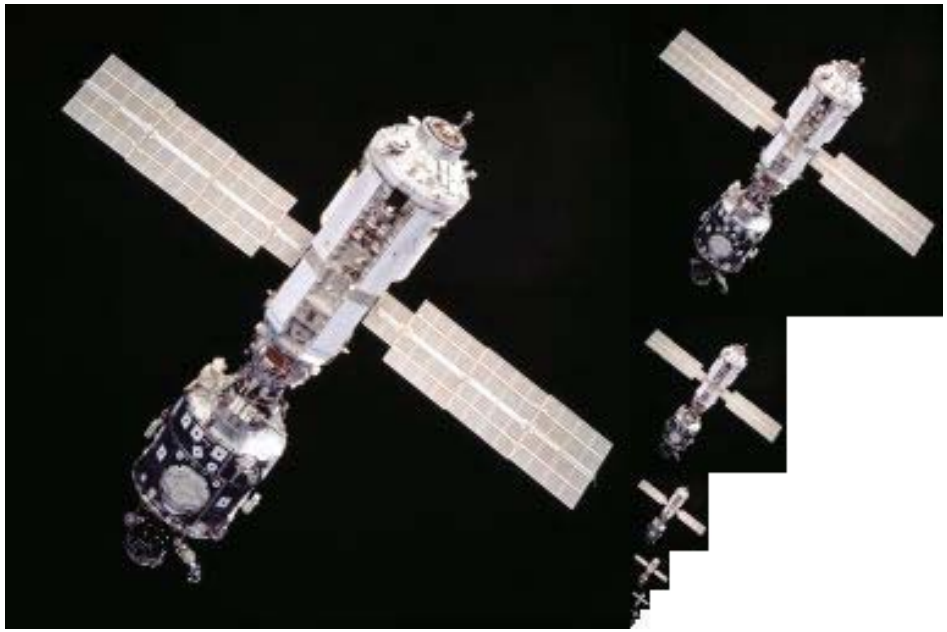
- **Level 2 – down-sample 4x4**

# Texture Filtering

- **Level 4 – down-sample 16x16**

# Mipmap

- **Mipmap (L. Williams 1983)**

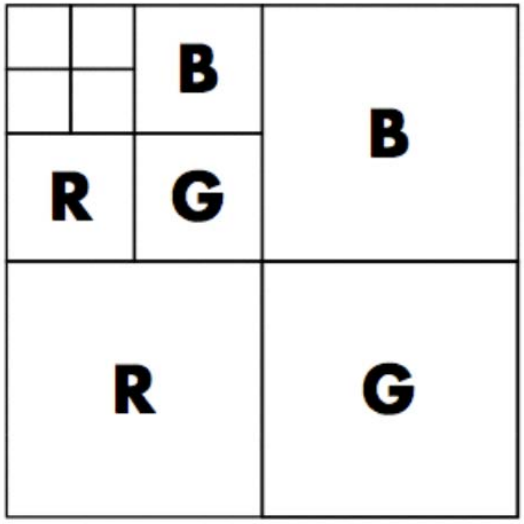


“Mip” comes from the Latin “multum in parvo", meaning a multitude in a small space

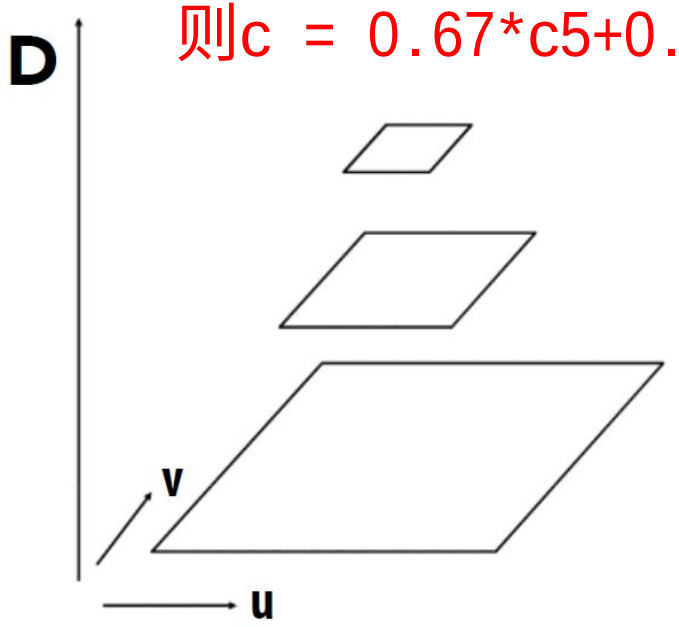
# Mipmap

- **What is the storage overhead of a mipmap?**

颜色为层数的插值
e.g. c1,c2,c3,...
D=5.333
则c = 0.67*c5+0.33*c6



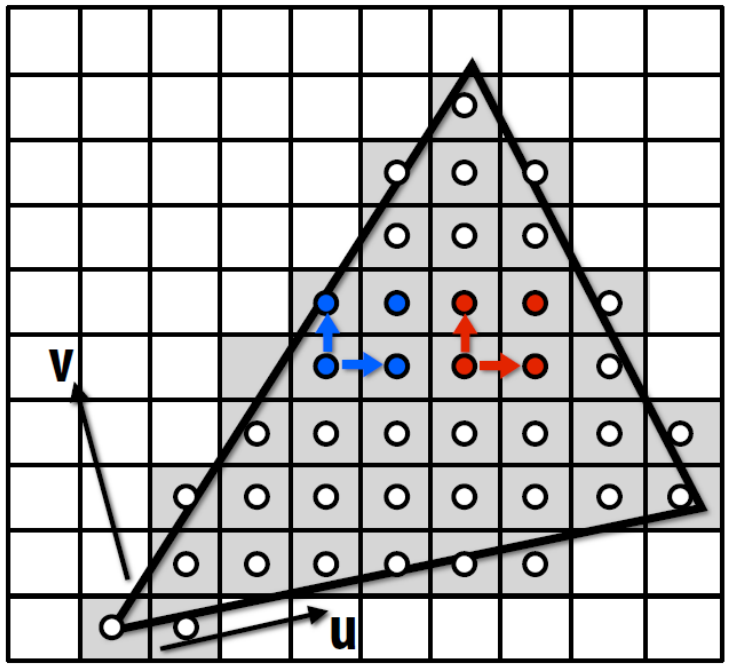

Williams' original
proposed mipmap layout



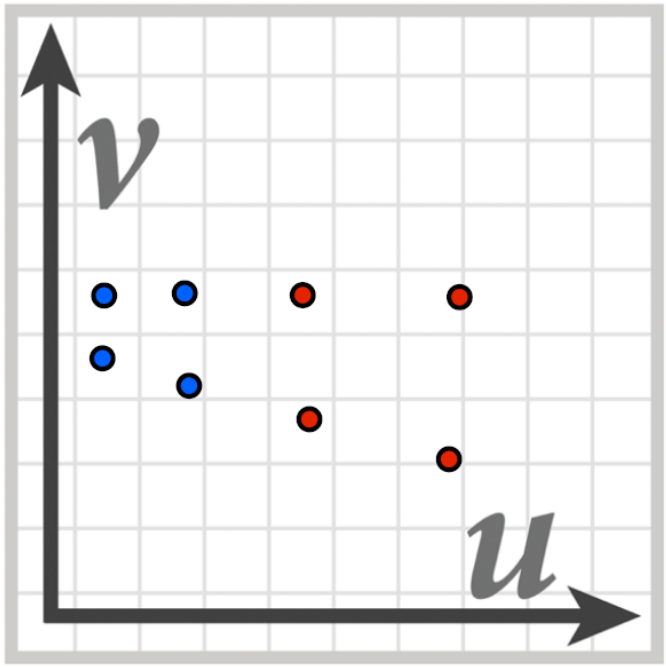

"Mip hierarchy"
level = D

# Mipmap

- **Computing mipmap level D**
  - Estimate texture footprint using texture coordinates of neighboring screen samples



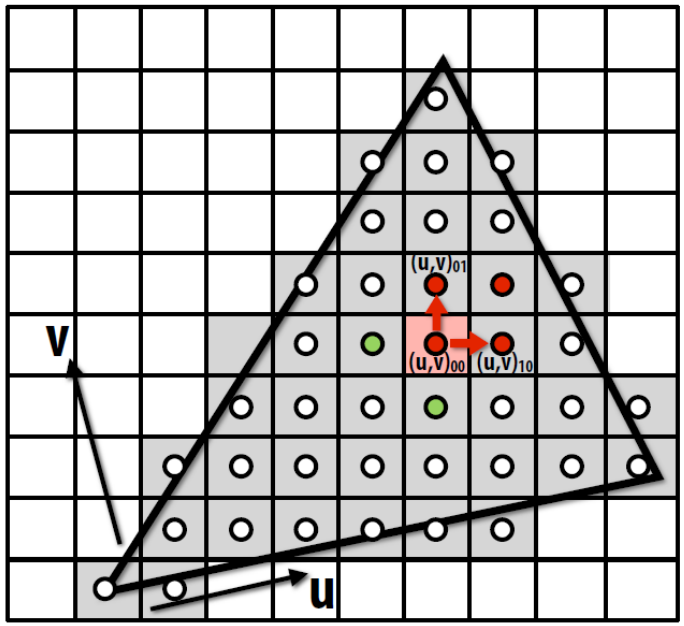

Screen space

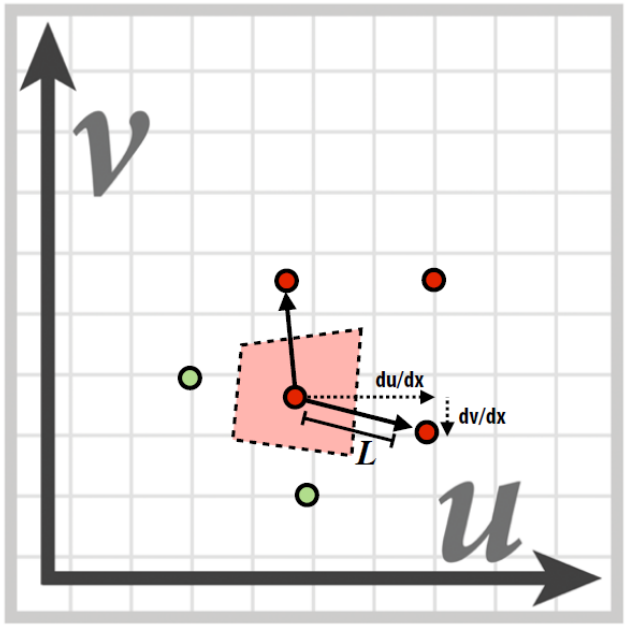



Texture space

# Mipmap

- **Computing mipmap level D**



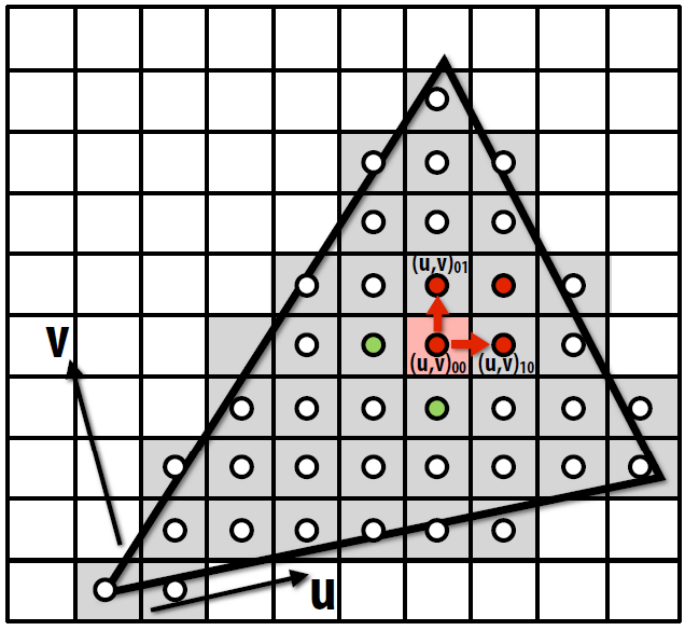



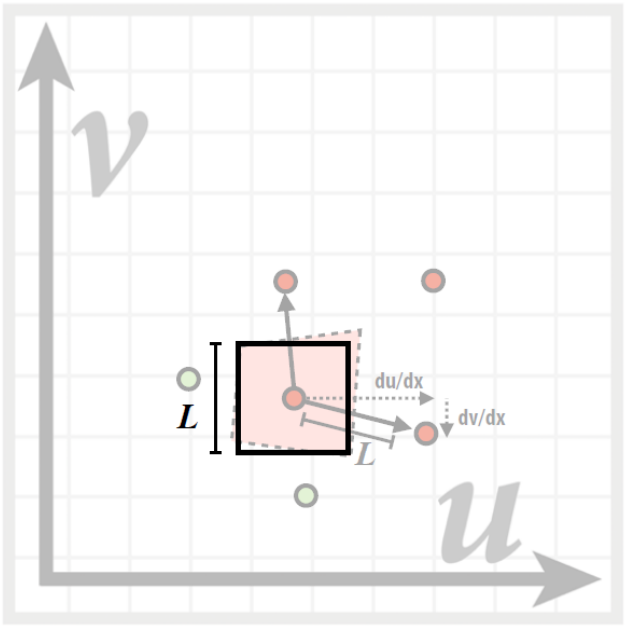

$$D = \log_2 L$$

$$L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$

# Mipmap

- **Computing mipmap level D**



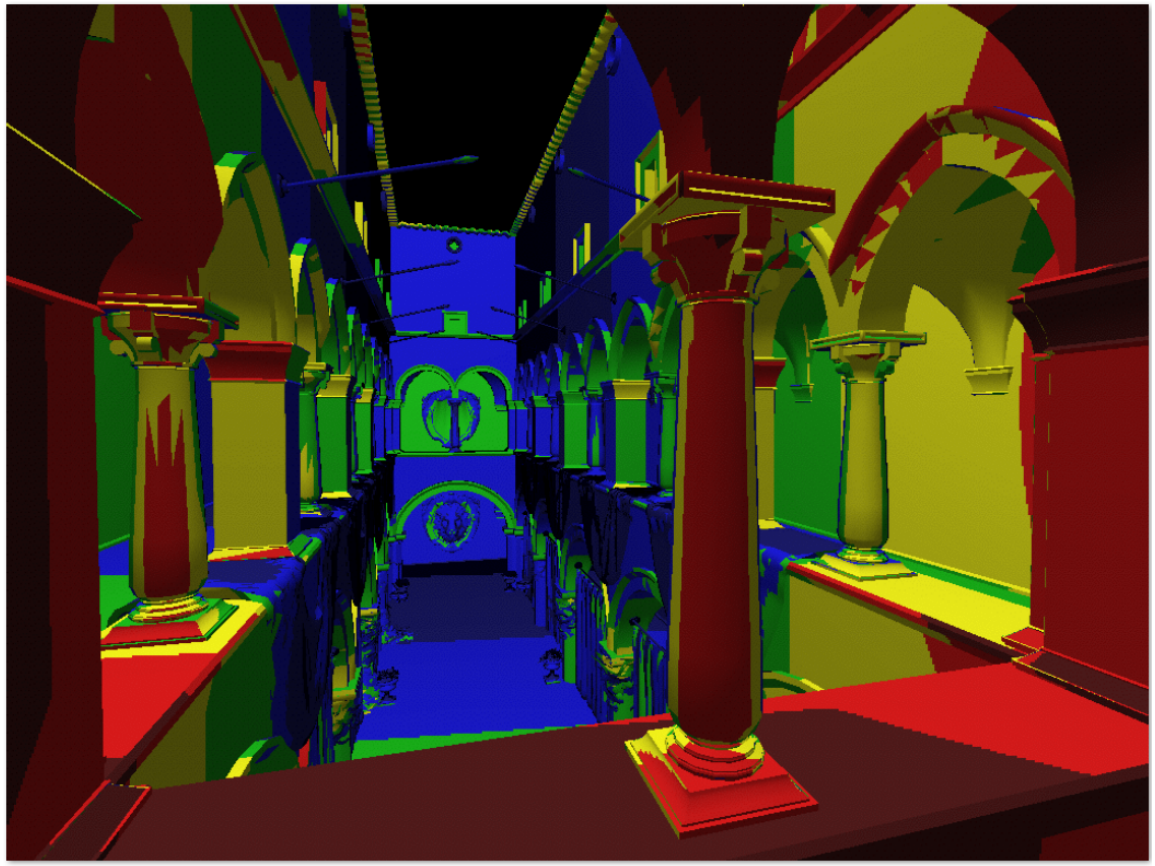




$$D = \log_2 L$$

$$L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$

# Mipmap

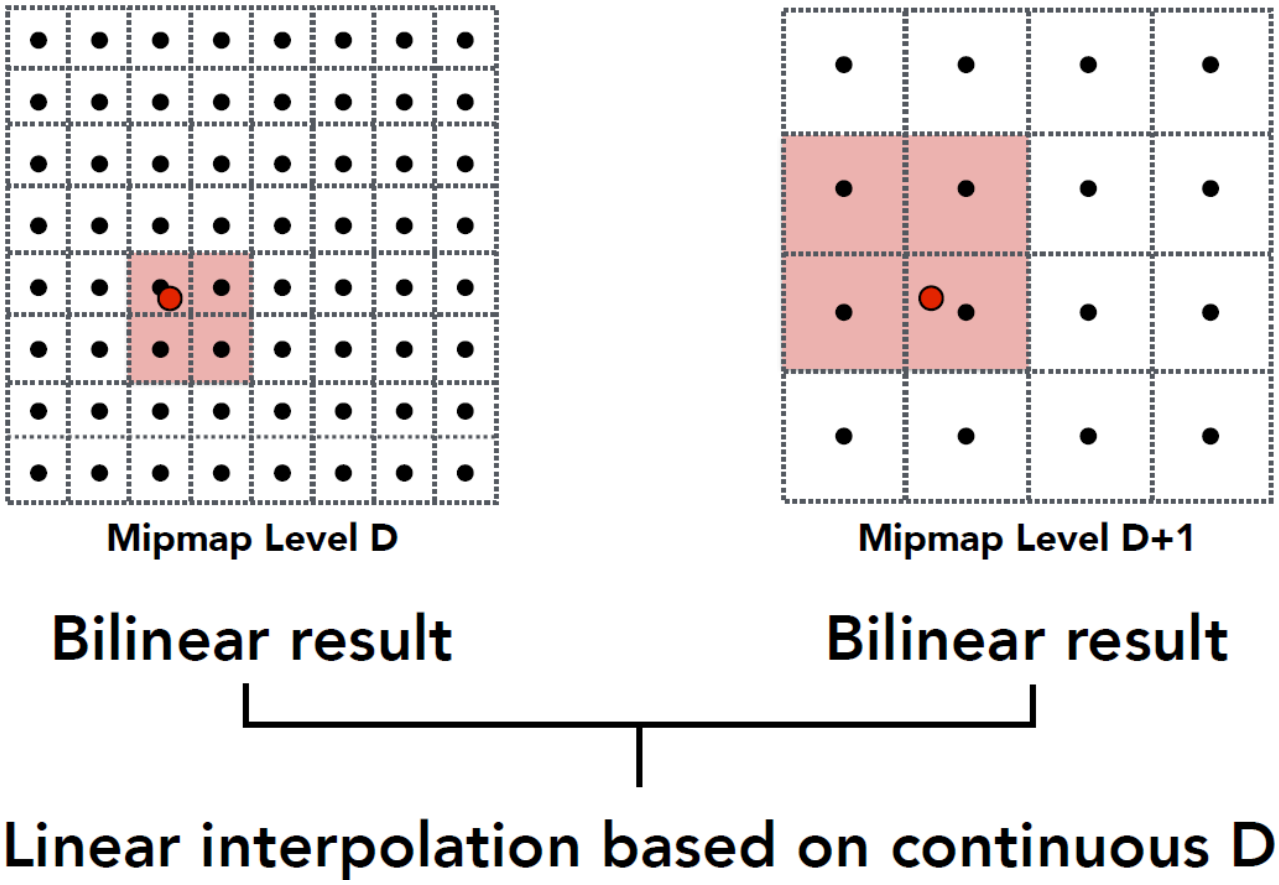
- **Visualization of mipmap level**



D clamped to nearest level

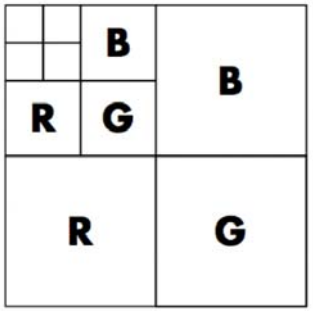
# Mipmap

- **Trilinear filtering**

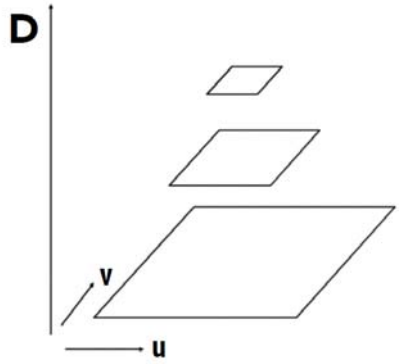


# Mipmap - texture color interpolation

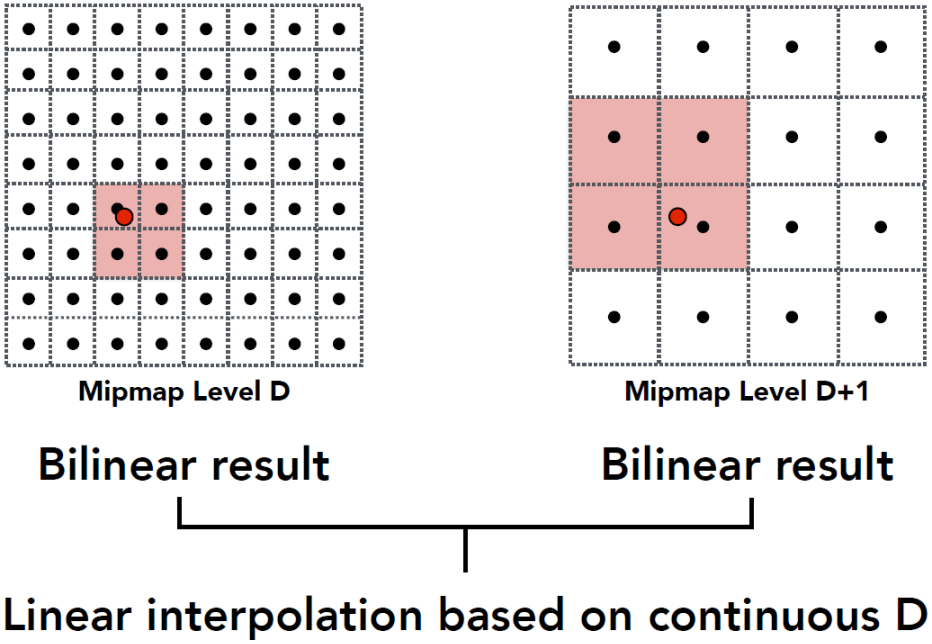
- **Interpolating texture colors**
  - With mipmapping, given a (u,v, D) (could be arbitrary values), how can we determine the color?
  - Trilinear interpolation




Williams' original proposed mipmap layout

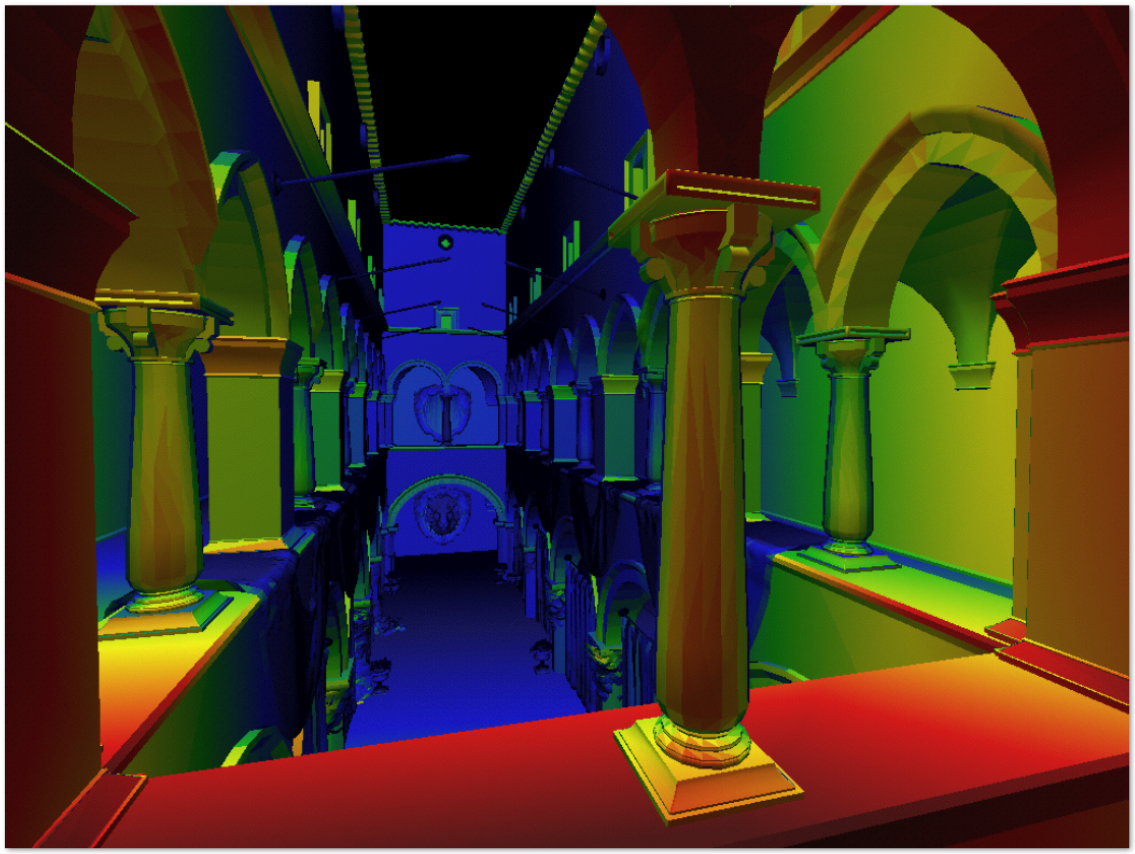



"Mip hierarchy" level = D



# Mipmap

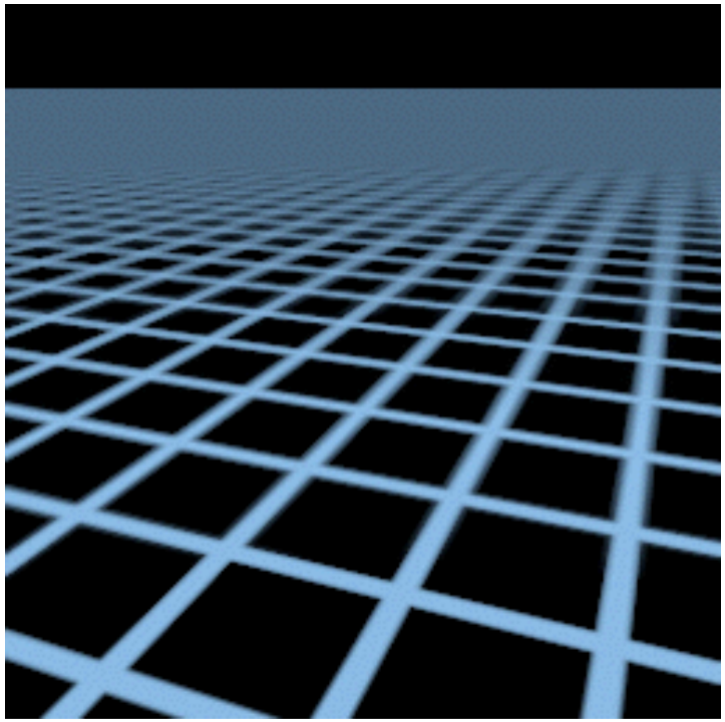
- **Visualization of mipmap level**



Trilinear filtering: visualization of continuous D

# Mipmap limitations

- **Observation**
  - Over-blur
  - Blur along some specific directions



Mipmap trilinear sampling

# Anisotropic filtering

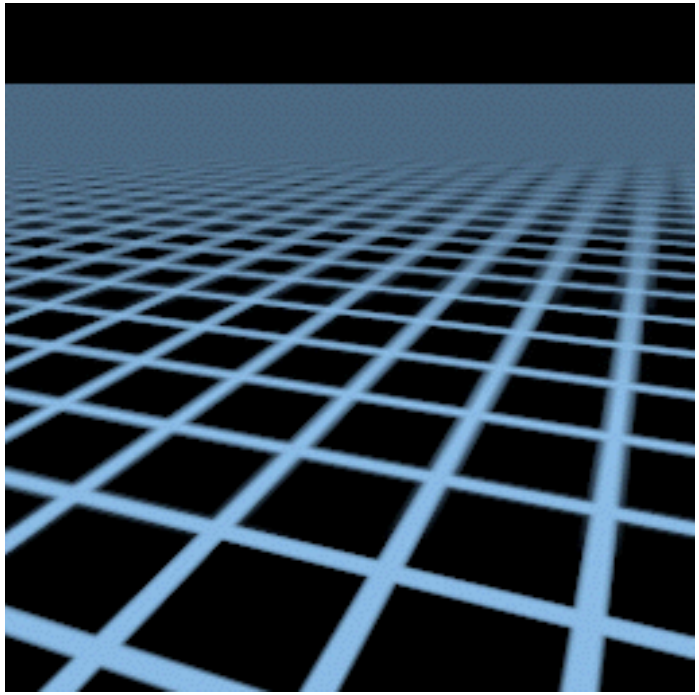
- **A method of enhancing the image quality of textures on surface**
  - At oblique viewing angles with respect to the camera
  - The projection of the texture appears to be non-orthogonal
- **Ripmap**
  - An extension from mipmap
  - Anisotropically downsample texture images

# Anisotropic filtering

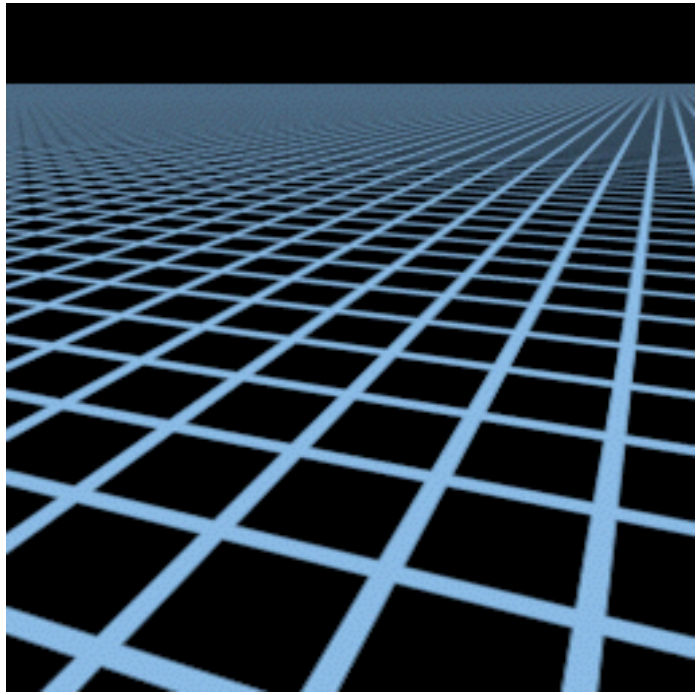
- **Ripmap**

# Anisotropic filtering

- **Elliptical weighted average filtering**



Mipmap



Ripmap

# Texture mapping in OpenGL

- Prepare texture map

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
// set the texture wrapping/filtering options (on the currently bound texture object)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// load and generate the texture
int width, height, nrChannels;
unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0);
if (data)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
```

# Texture mapping in OpenGL

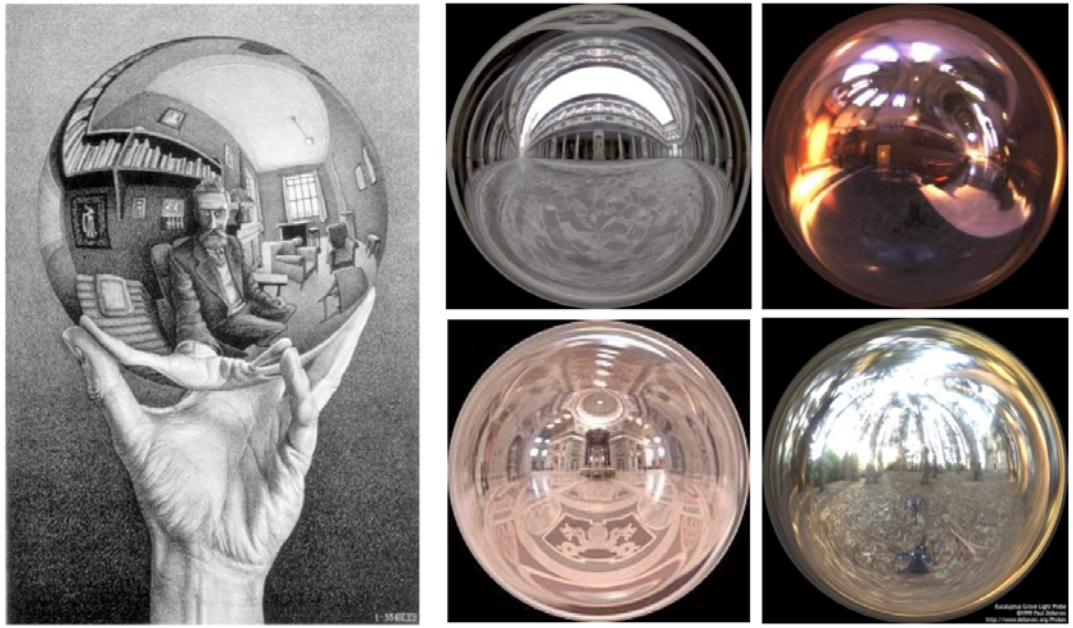
- Rendering

```
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,   // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,   // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f    // top left
};

//…prepare vertex array data…

glBindTexture(GL_TEXTURE_2D, texture);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

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

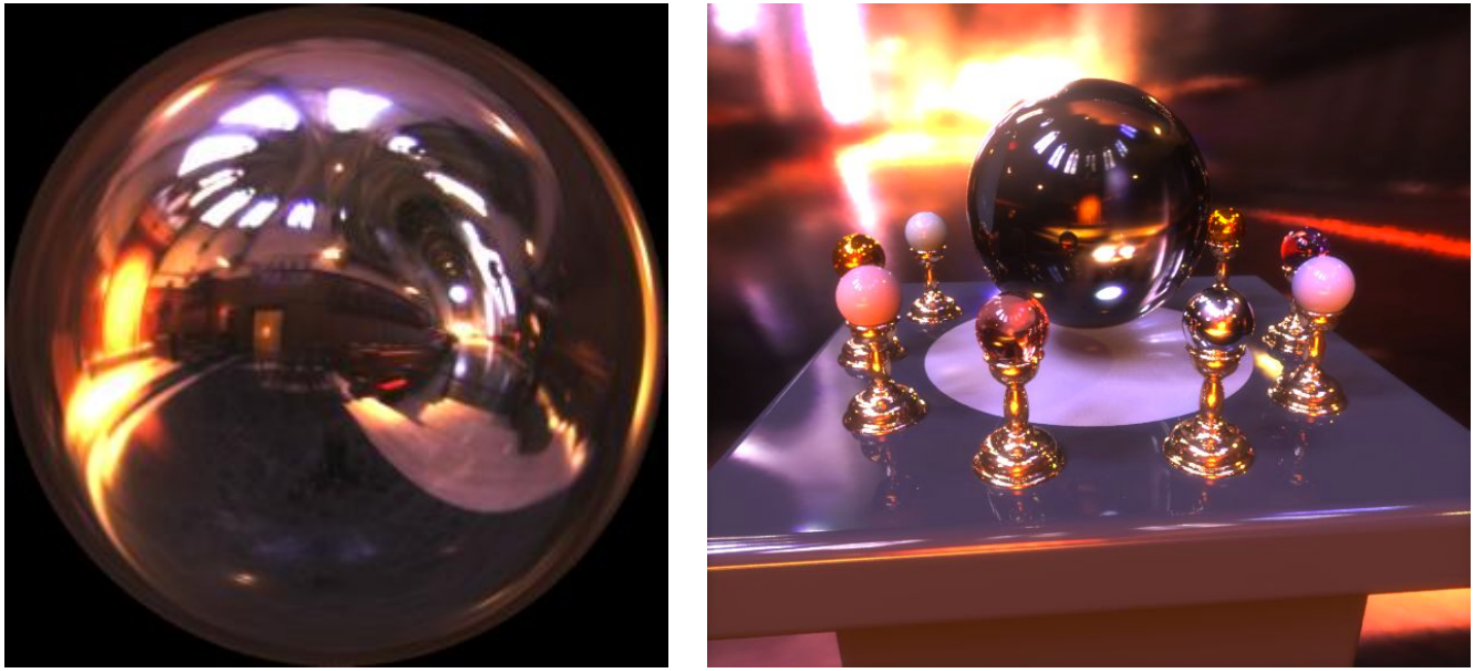
# Environment mapping

- **An efficient image-based lighting**
  - Approximate the appearance of a reflective surface
  - By means of a pre-computed texture image
  - The texture is used to store the image of the distant environment surrounding the rendered object



# Environment mapping

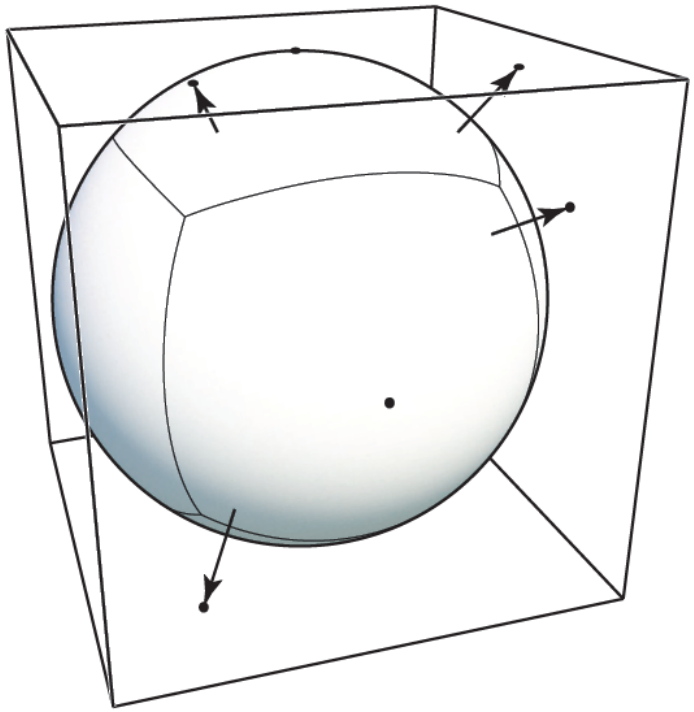
- **Environmental Lighting**
  - Texture storing illumination

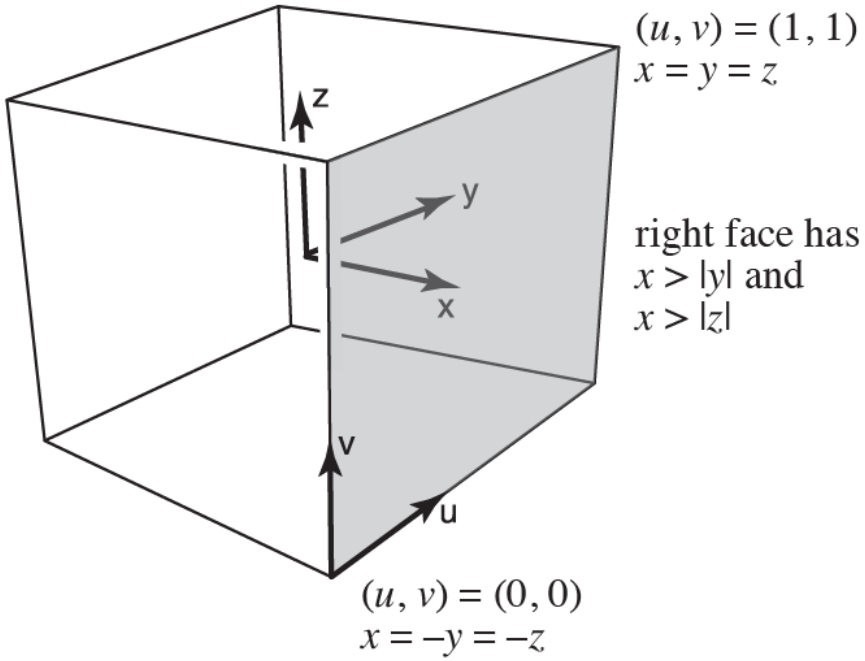


Environment map (left) used to render realistic lighting (right)

# Environment mapping

- **Cube Map**

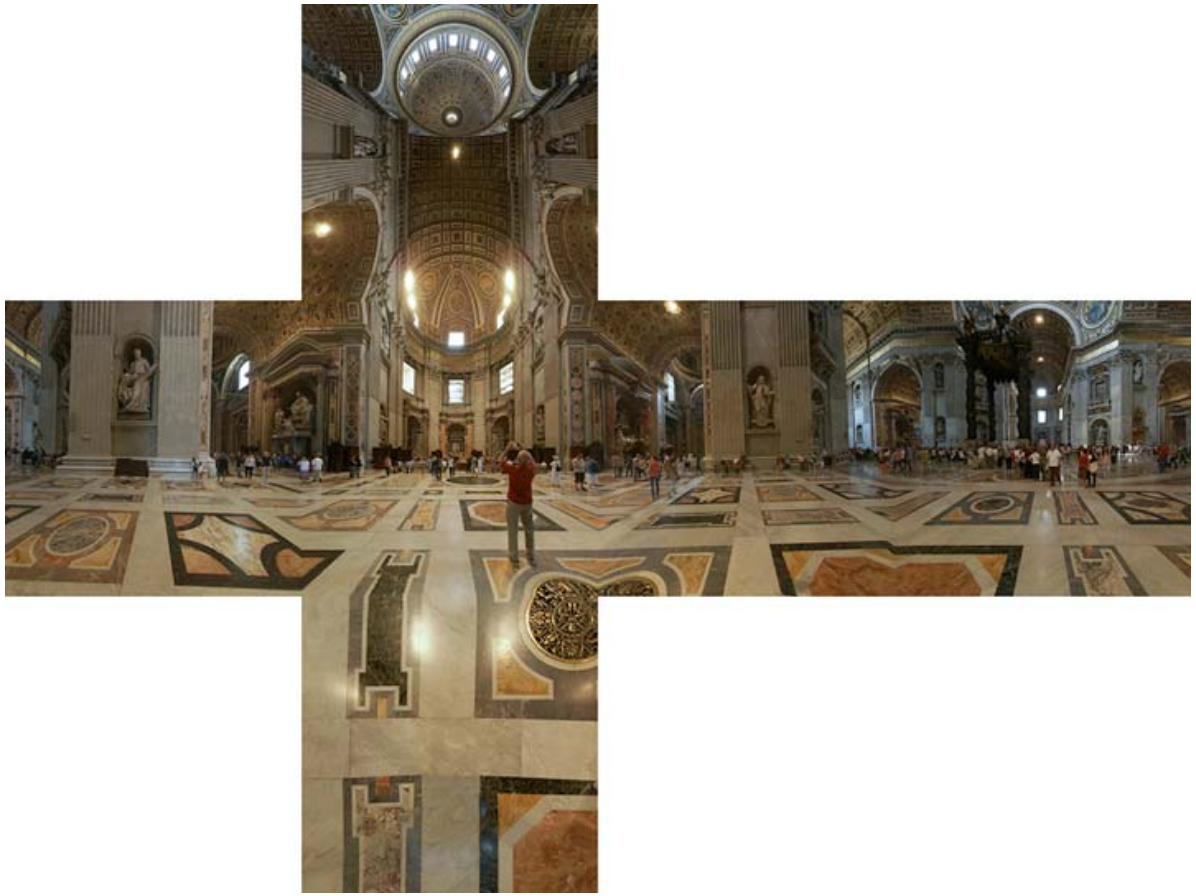





A vector maps to cube point along that direction.
The cube is textured with 6 square texture maps.

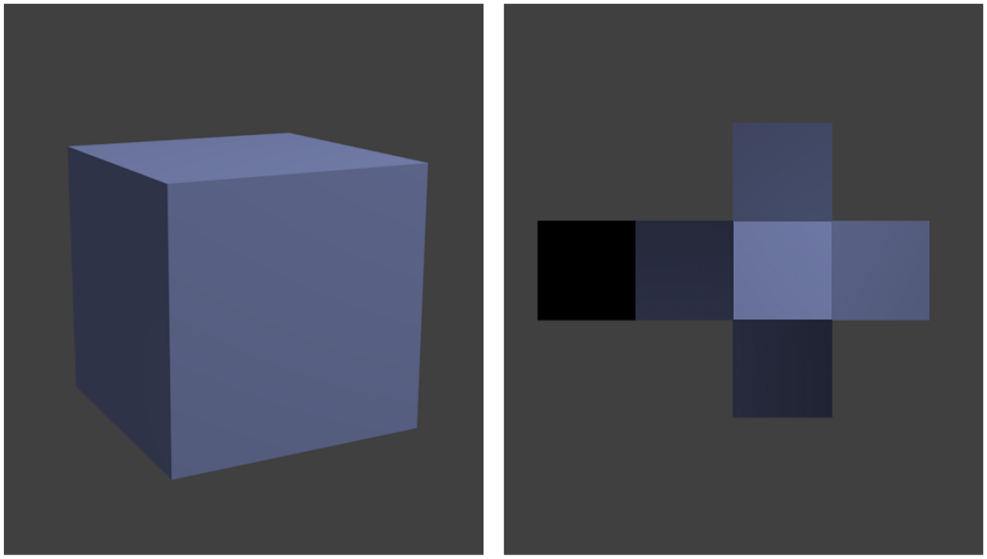
# Environment mapping

- **Cube Map**



# Lightmap

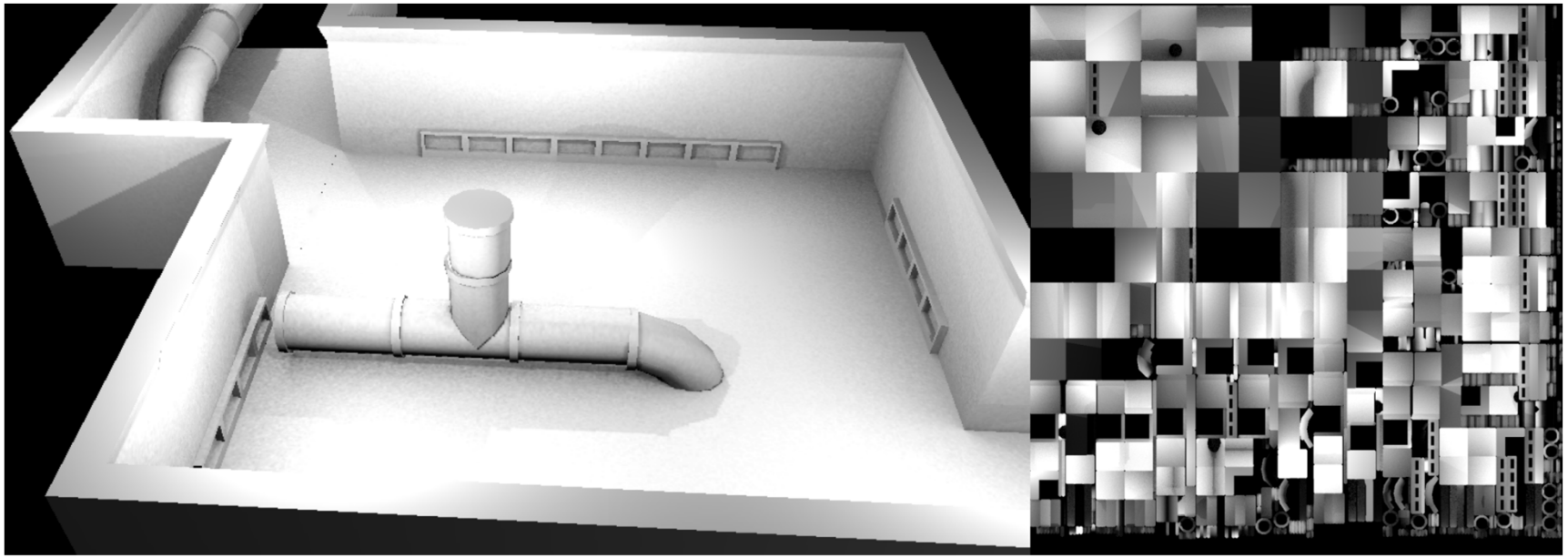
- **A data structure used in lightmap**
  - The brightness of surfaces is pre-calculated and stored in texture maps for later use
  - Lightmaps are most commonly applied to static objects in real-time 3D graphics (e.g. video game)



Simple lightmap

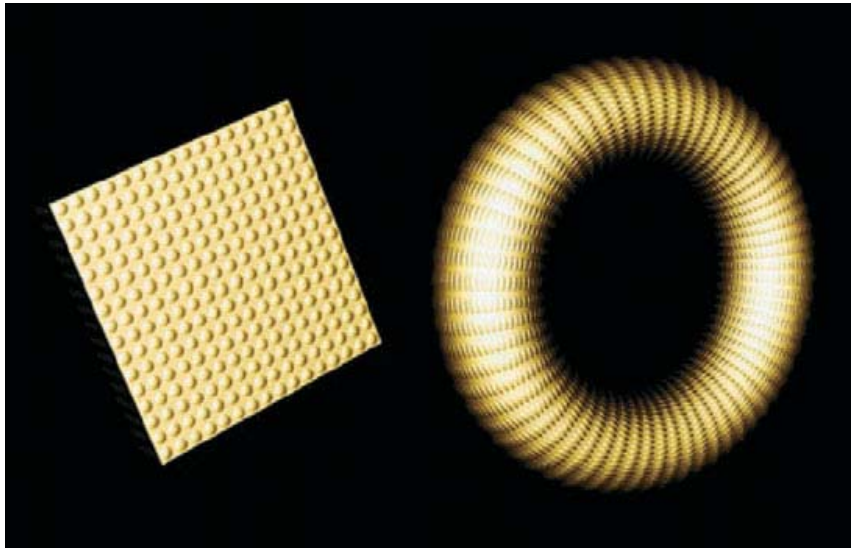
# Lightmap

- **More complex lightmap example**



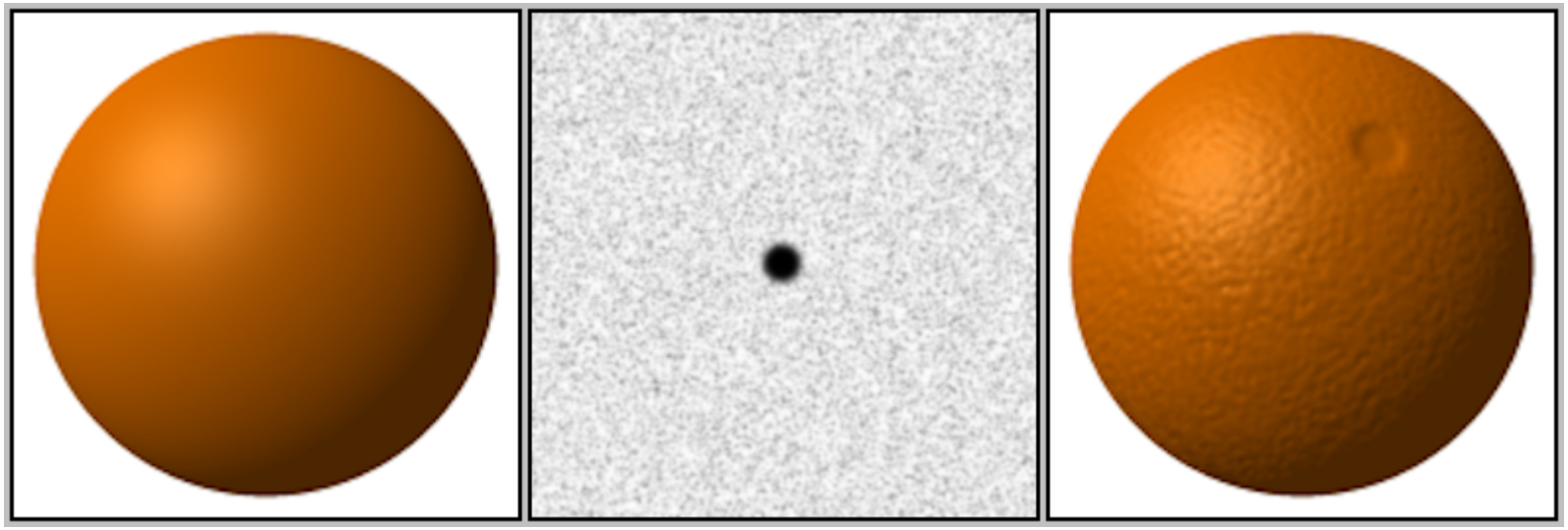
# Normal mapping

- **Add apparent geometric complexity during fragment processing**
  - Using fragment shader
  - A valid surface normal at each fragment location
  - Procedurally perturb the normal prior to the lighting calculation



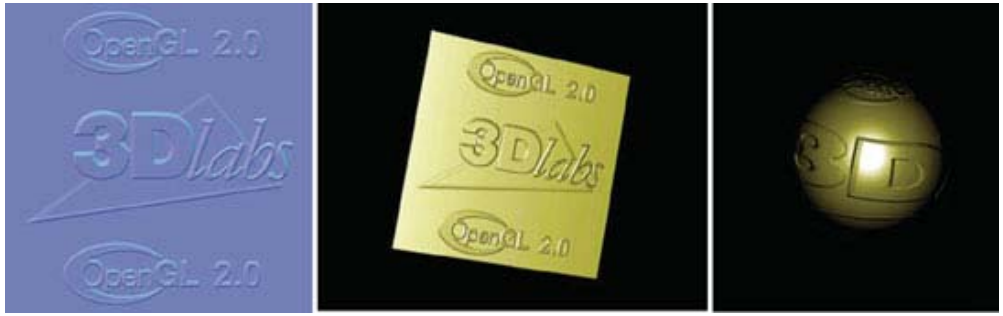
# Normal mapping

- **Simulating bumps and wrinkles on the surface of an object**
  - Perturb the surface normals of the object
  - Perturbation is defined by a texture



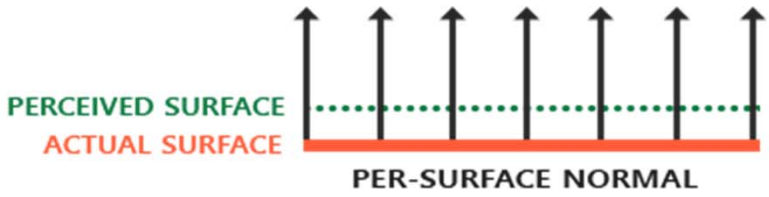
# Normal mapping

- **More examples of normal mapping**
  - Perturb the surface normals of the object
  - Perturbation is defined by a texture

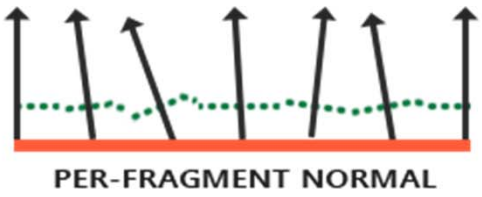



# Normal mapping

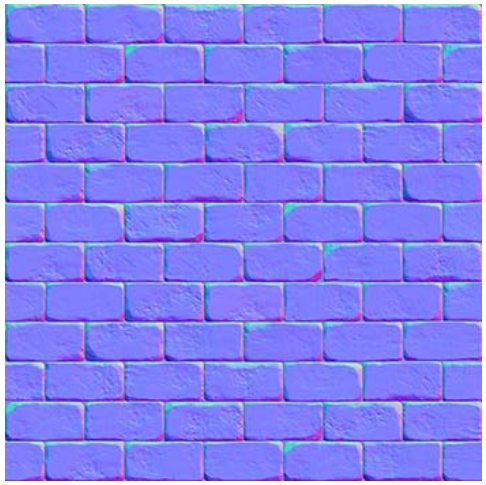
- **How normal mapping works**
  - Store normal in a texture
  - Look up a texture for the normal of a particular point



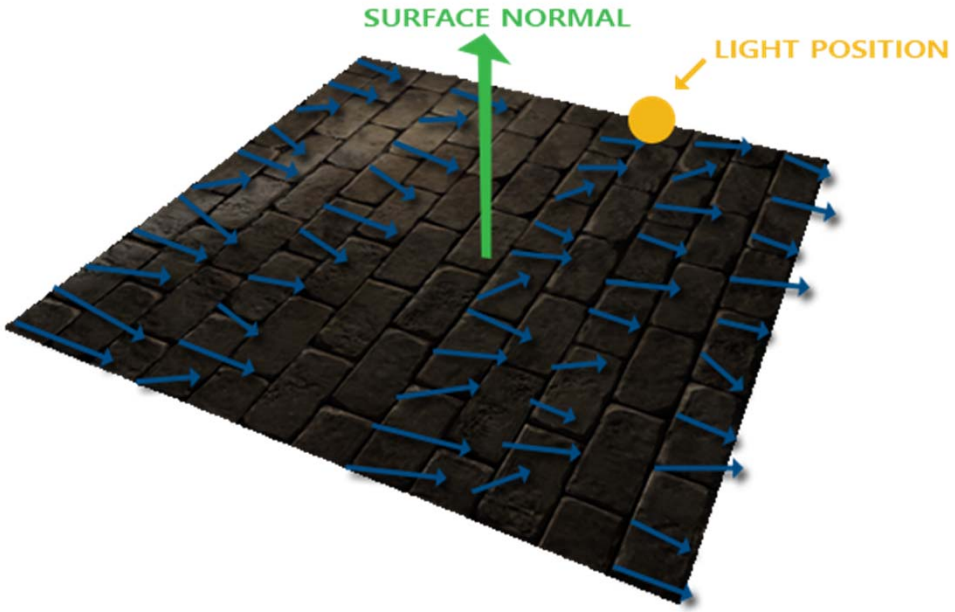

PER-SURFACE NORMAL



PER-FRAGMENT NORMAL



Normal stored in RGB



# Normal mapping

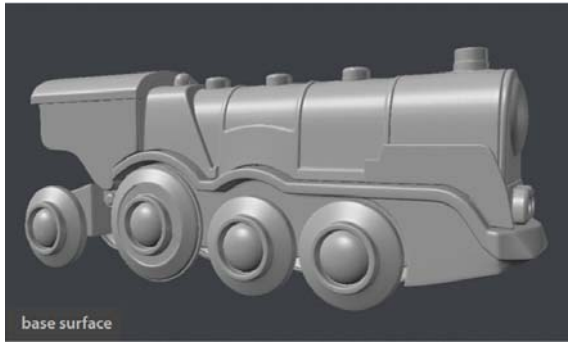
- **Rendering without vs. with normal mapping**
  - Depth effect is more obvious for normal mapping
  - Vary with lighting changes


# Displacement mapping

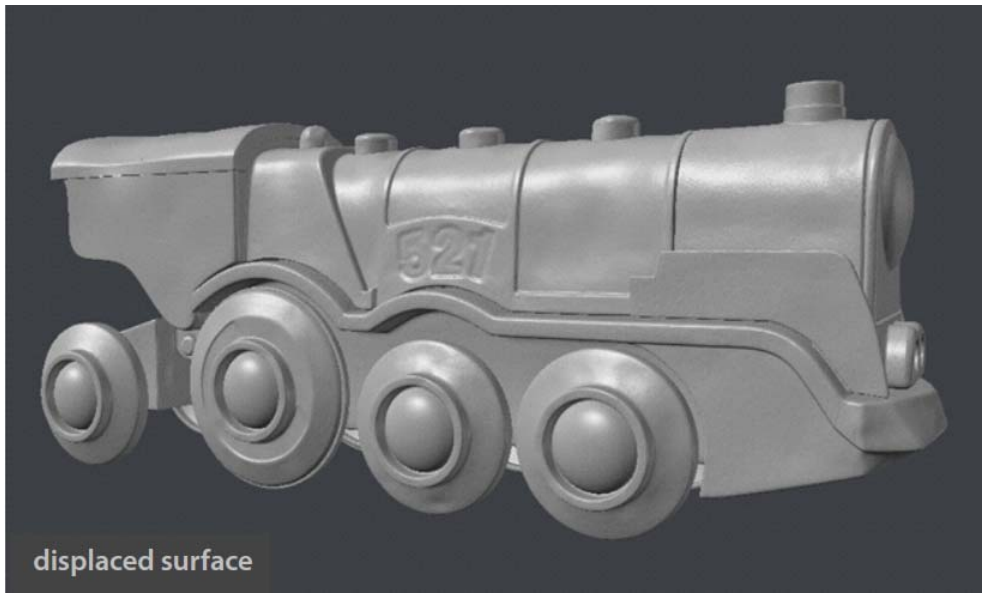
- **Texture stores perturbation to surface position**

# Displacement mapping

- **Create surface detail by displacement mapping**

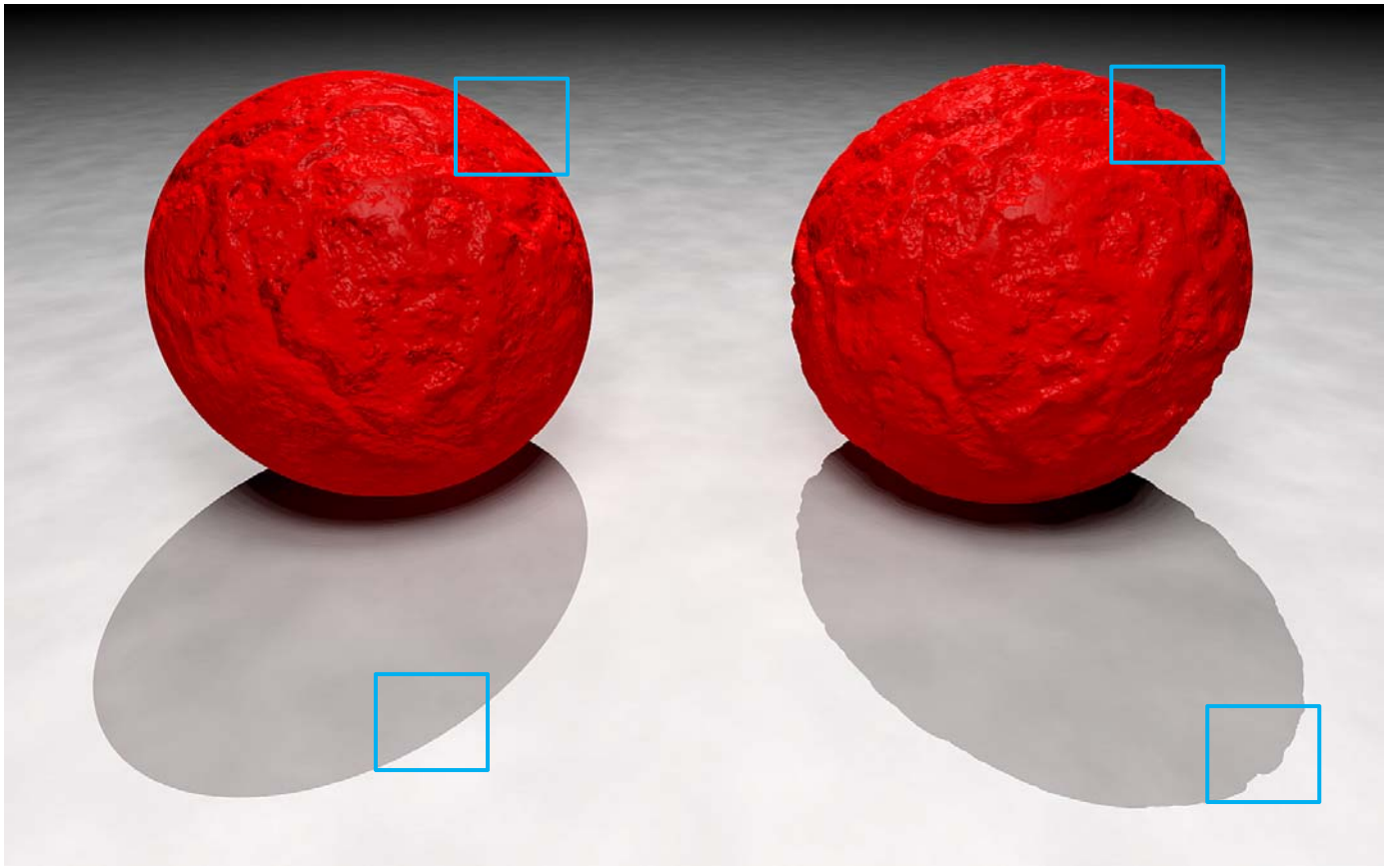






# Normal mapping v.s. displacement mapping

- **What is missing for normal mapping?**



# Procedural texture

- **A texture created using an algorithm**
  - Create a realistic surface or volumetric representation of natural elements
  - Wood, marble, granite, metal, stone, etc.

- **Solid texturing**
  - A process where the texture generating function is evaluated over $R^3$
  - At each visible surface point of the model
  - Texture color is evaluated directly from texture generating function

# Procedural texture

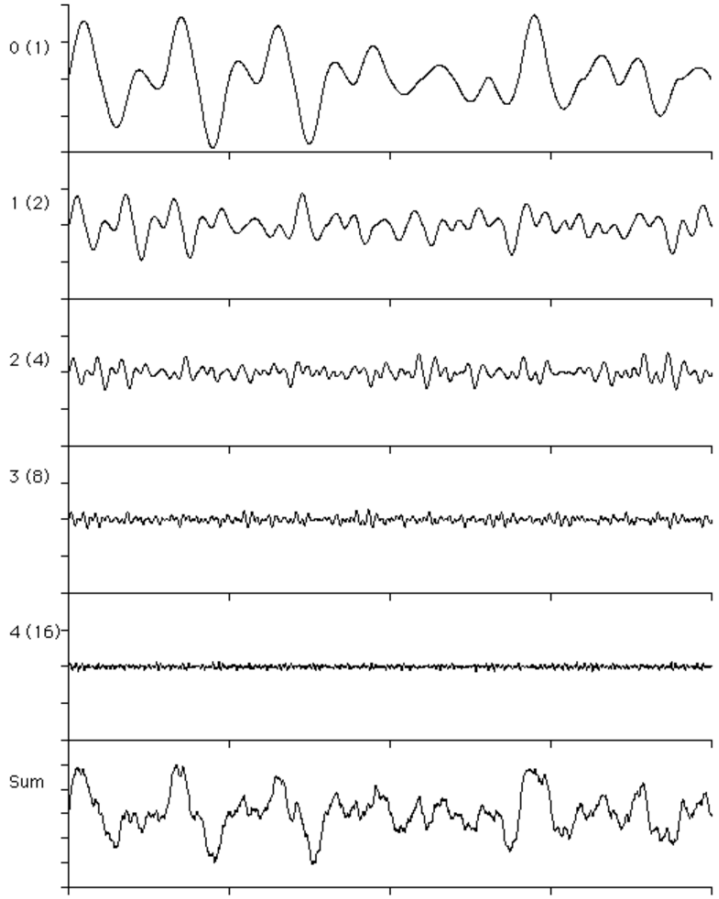
- **Perlin noise**
  - Developed by Ken Perlin in 1983
  - Its foundation is related to turbulence

- **Basic idea**
  - Generate noise function at different scales
  - Composite all of them in a wavelet-like form

$$NOISE\ (x) = \sum_{i=0}^{N-1} \frac{1}{a^i} Noise(b^i x)$$

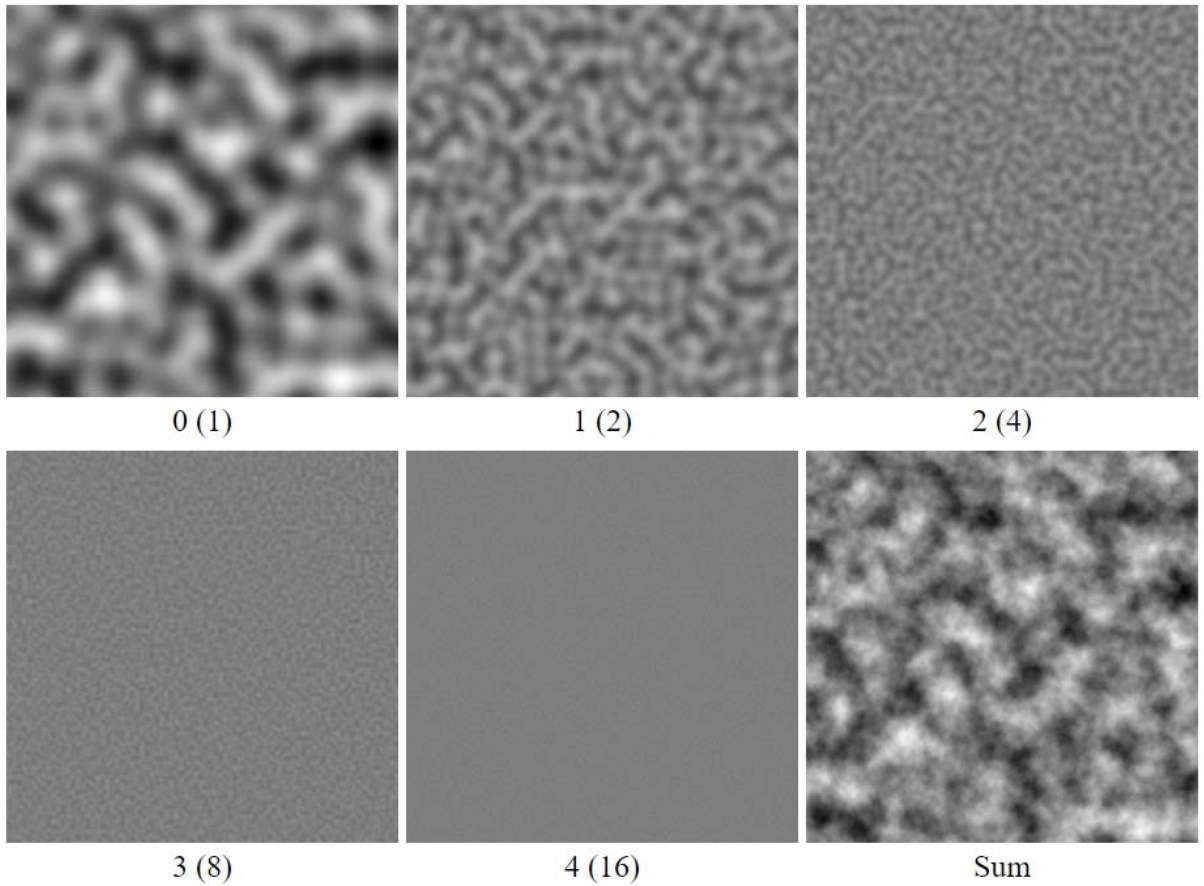
# Procedural texture

- **Perlin noise**
  - Example: 1D Perlin noise, a=b=2



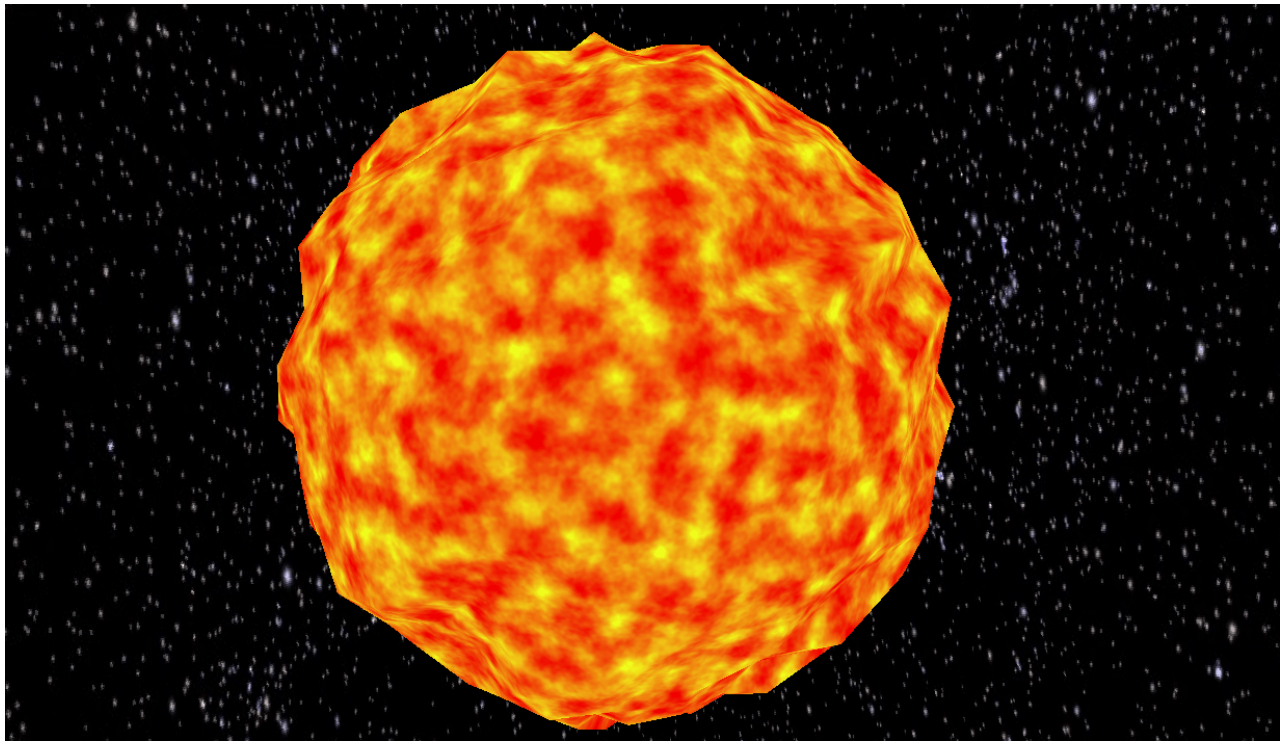
# Procedural texture

- **Perlin noise**
  - Example: 1D Perlin noise, a=b=2



# Procedural texture

- **Perlin noise**
  - Rendered sun object



# Buffers in OpenGL

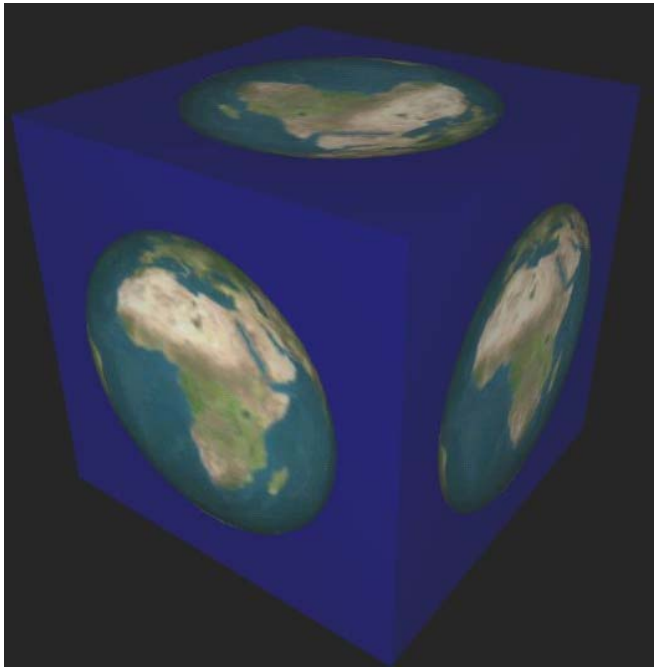
- **Vertex buffer**
  - Store all vertices including its attributes
- **Texture buffer**
  - Store texture data
- **Framebuffer**
  - Usually associated with the screen
  - Different types of framebuffers
    - Color buffer
    - Depth buffer etc.
- **Renderbuffer**
  - Allow rendering to off-screen buffer

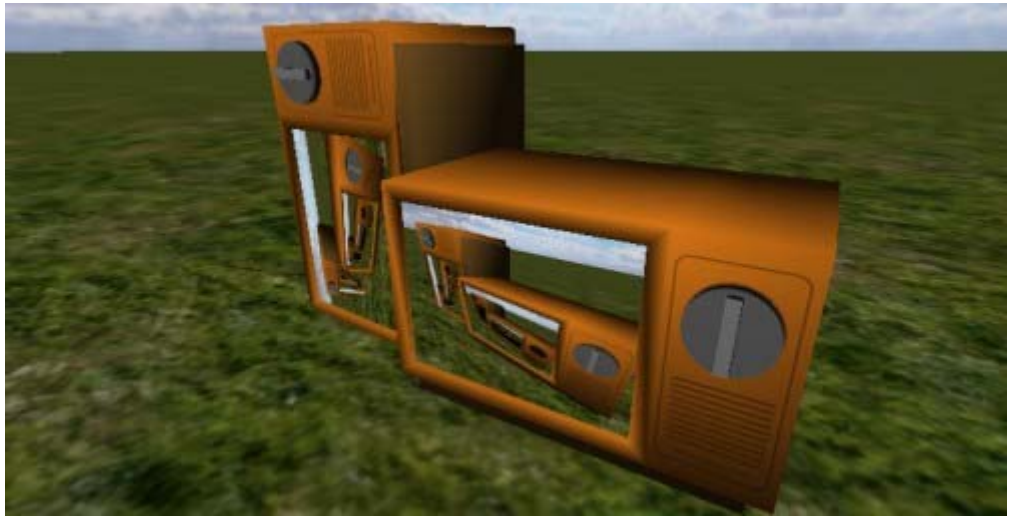
# Framebuffer objects

- **Renderbuffer**
  - Memory managed by OpenGL that contains formatted image data
  - Meaningful once it is attached to a framebuffer object
- **Framebuffer attachments**
  - When you render, you can send the results of that rendering to a number buffers
  - Attach a renderbuffer or a texture buffer

# Render to texture

- **Rendering directly to a texture buffer**
  - Render a scene into a texture like you render onto a screen
  - Attach a texture buffer to the framebuffer object

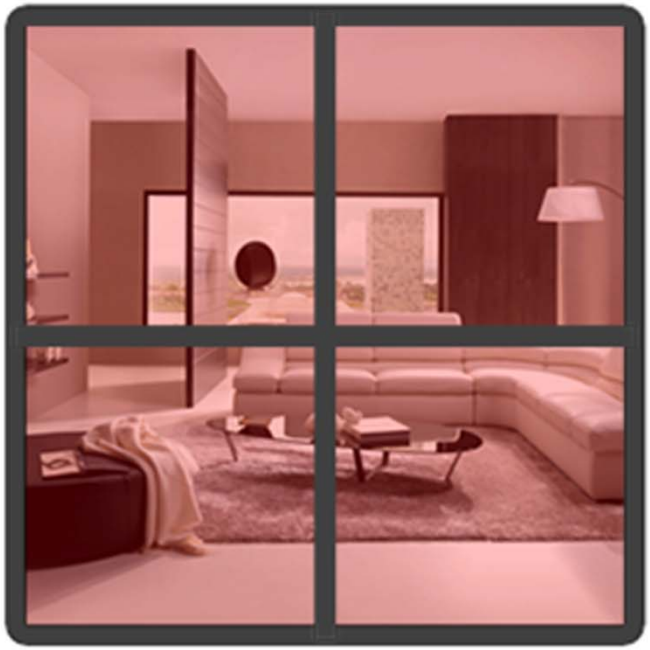




# Blending

- **How to model transparency?**
  - Light can go through materials

Full transparent window

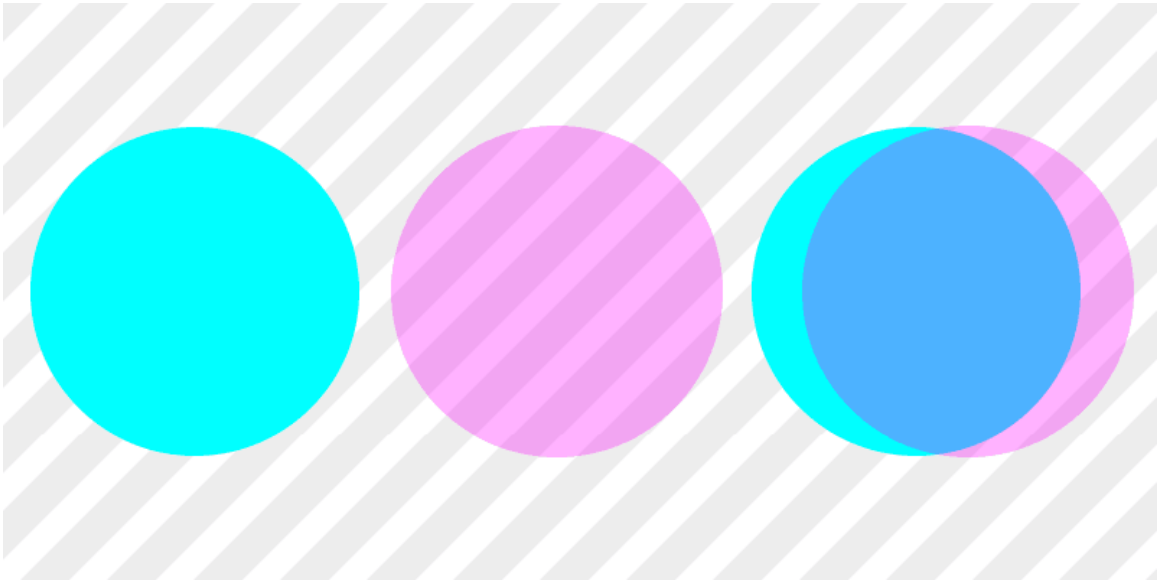


Partially transparent window

# Blending

- **Alpha blending**
  - Color weighting scheme

$$C_o = \alpha_a C_a + (1 - \alpha_a) C_b$$

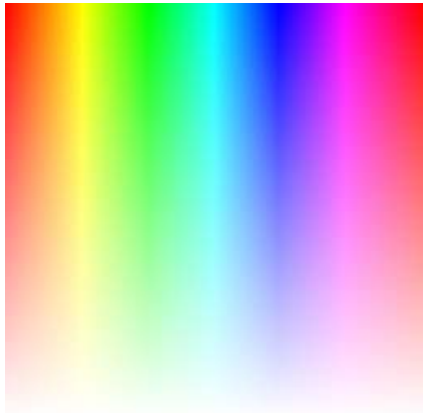


# Blending

- **RGBA color**
  - Alpha channel specifies transparency

| Sample Length: | 8 | | | | | | | | 8 | | | | | | | | 8 | | | | | | | | 8 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Channel Membership: | Red | | | | | | | | Green | | | | | | | | Blue | | | | | | | | Alpha | | | | | | | |
| Bit Number: | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

  - Define colors with each a transparency component
    - Spatially varying transparency



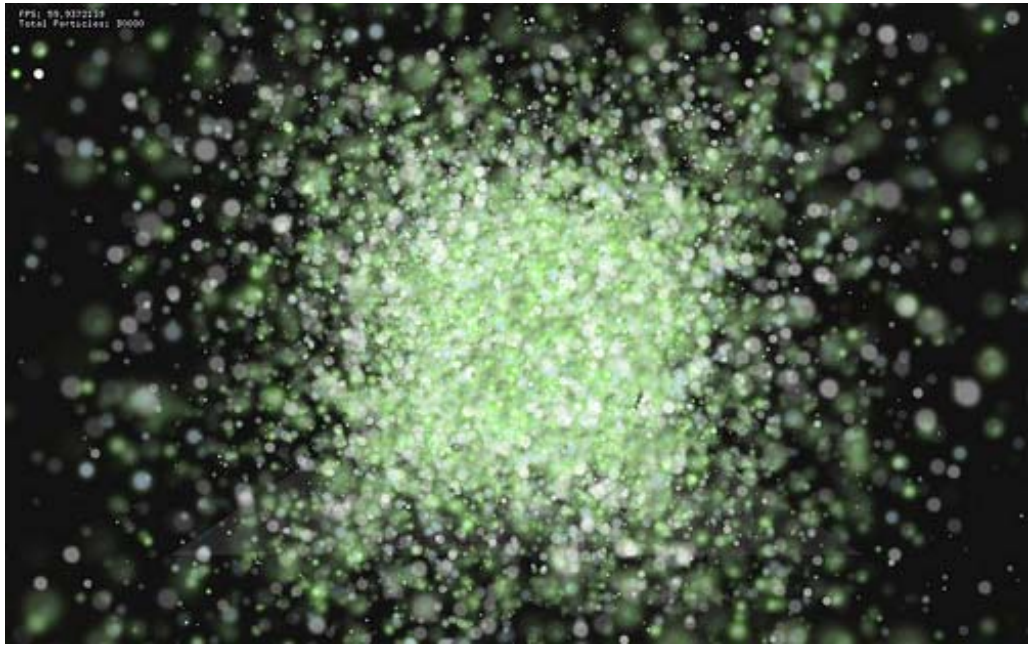
RGBA image with translucent and transparent portions, displayed on a white background

# Multi-texturing

- **Mapping multiple textures onto the same surface**
  - Transparency enabled
    - Combine different texture contents
    - Simple model for subsurface effects
    - Blending different layers of textures

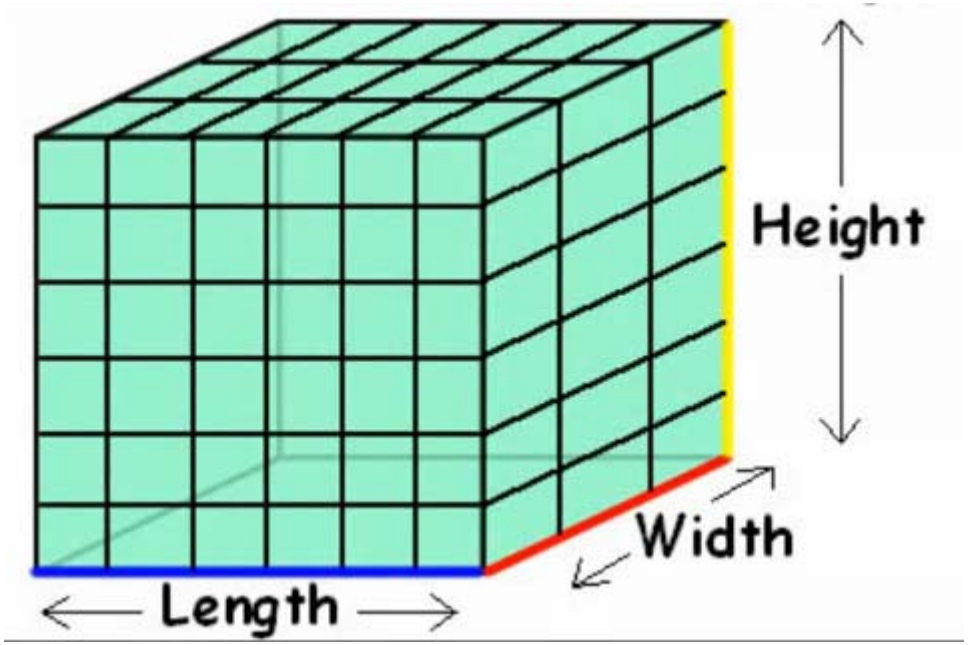
# Billboard textures

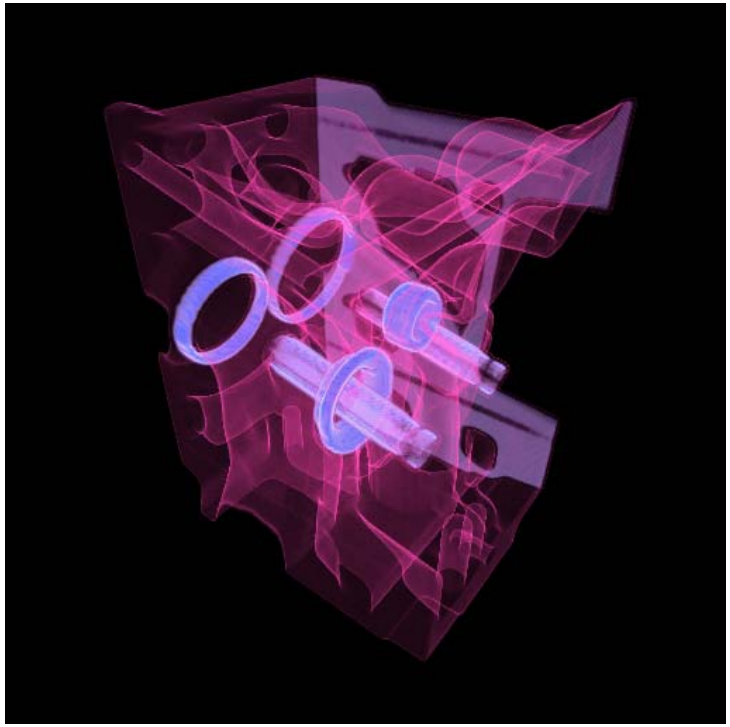
- **Billboard**
  - A plane with texture mapping
  - The plane always orients towards camera
  - Tree/smoke/fire/particle rendering



# 3D texture

- A slice of 2D textures



# 3D texture

- **3D texture mapping**
  - Mapping to a volume or a surface

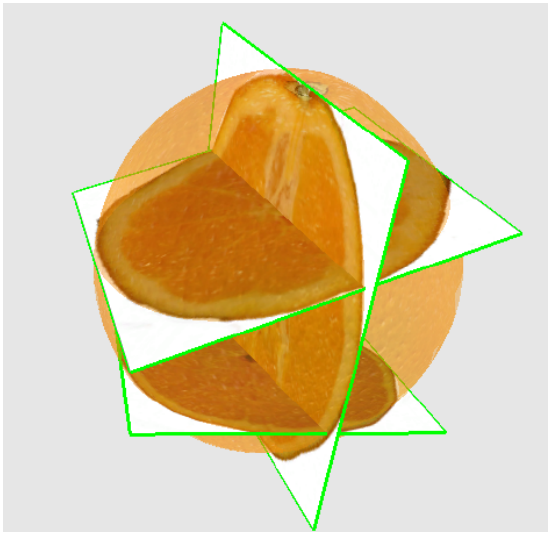
# Solid texture

- **A volumetric texture with 3D texture coordinates**
  - From 3D procedure or a 3D volumetric data

Results from “Solid Texture Synthesis from 2D Exemplars”, Johannes Kopf et al. SIGGRAPH 2007

# Solid texture

- **How to perform 3D texturing**
  - Each vertex is associated with a 3D texture coordinate
  - Interpolate color from a 3D dataset or procedure
  - Allow cutting through a solid volume



# Next lecture: Ray tracing basics