Computer Graphics

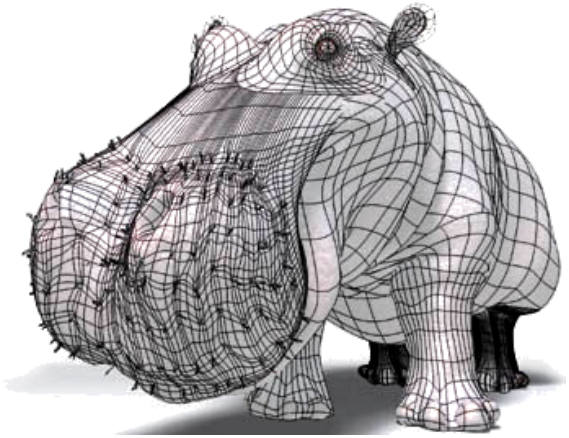
Texture Mapping I

Mark Pauly

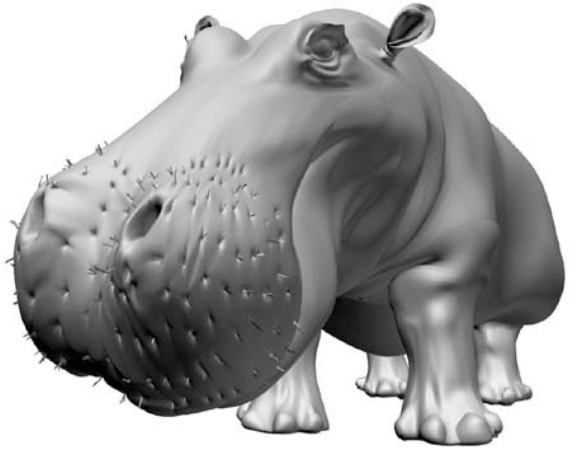
Geometric Computing Laboratory

Materials & Texture

- So far: color/material varies per model or per vertex
- Textures add visual detail without raising geometric complexity:
  Paste 2D images onto 3D geometry



Geometry



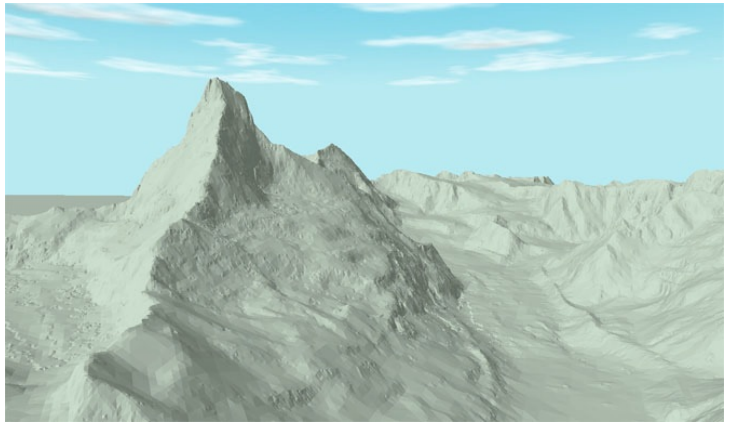
+Lighting

+Texture

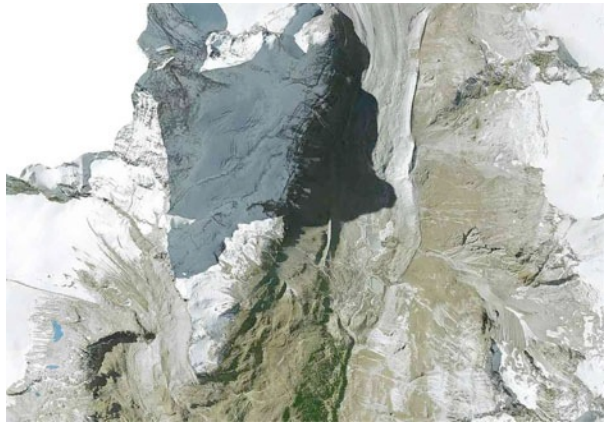
Images from http://www.3drender.com/jbirn/productions.html

Materials & Texture

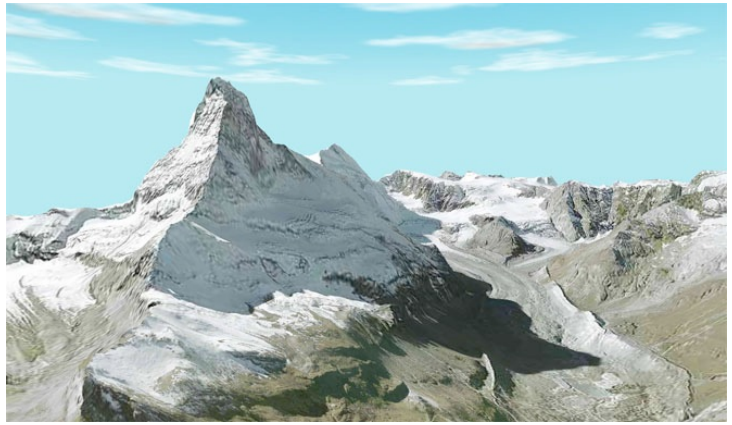
- So far: color/material varies per model or per vertex
- Textures add visual detail without raising geometric complexity:
  Paste 2D images onto 3D geometry



Geometry



Texture



Textured Mesh

Images from http://www.endoxon.ch

Materials & Texture

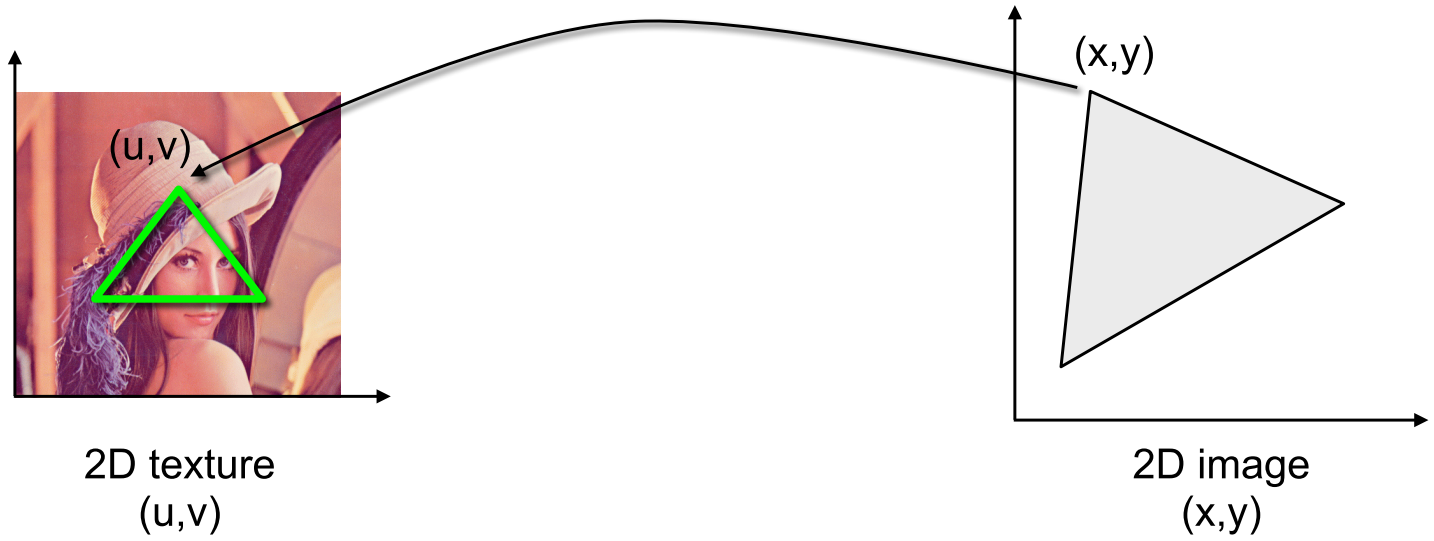
- Textures allow us to model many surface properties:
  - material (diffuse + specular colors/coefficients)
  - normal vector (normal mapping, bump mapping)
  - geometry (displacement mapping)
  - opacity (alpha mapping)
  - reflection/illumination (environment mapping)
  - …
- Textures change the parameters of the shading equation.

Outline for today

1. Texturing one triangle
2. Texturing a triangle mesh
3. Texture filtering
4. Textures in OpenGL
5. Shadow Maps

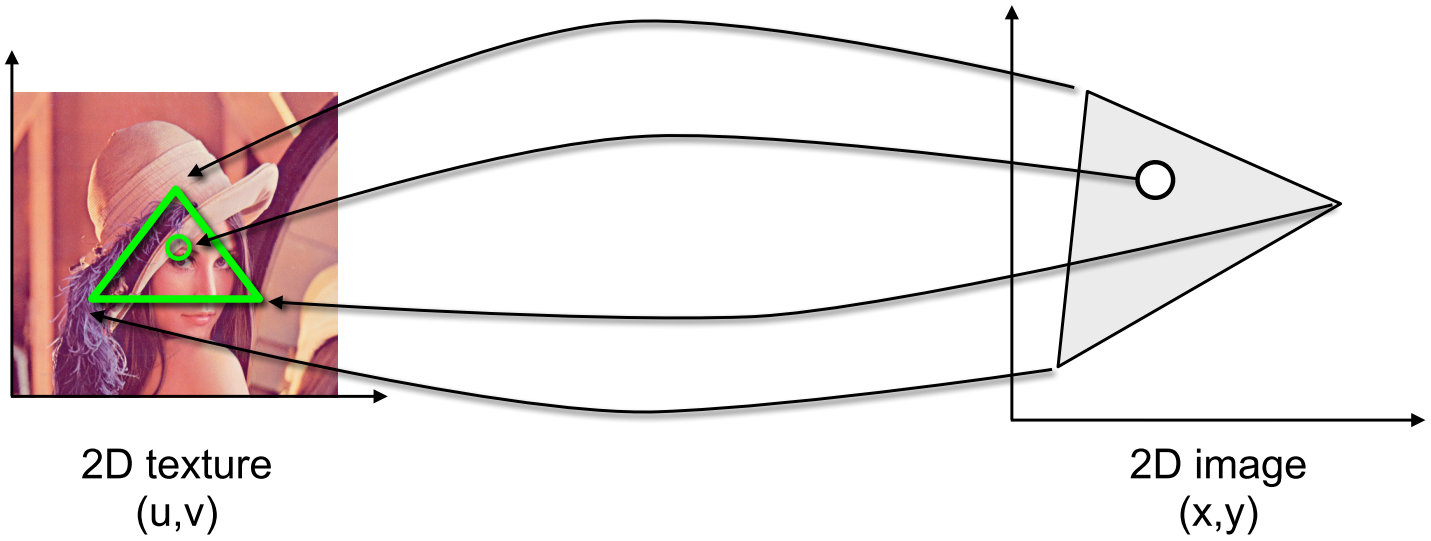
Texturing One Triangle

Texturing One Triangle



- Assign 2D texture coordinate $\mathbf{u} = (u, v)$ to each vertex

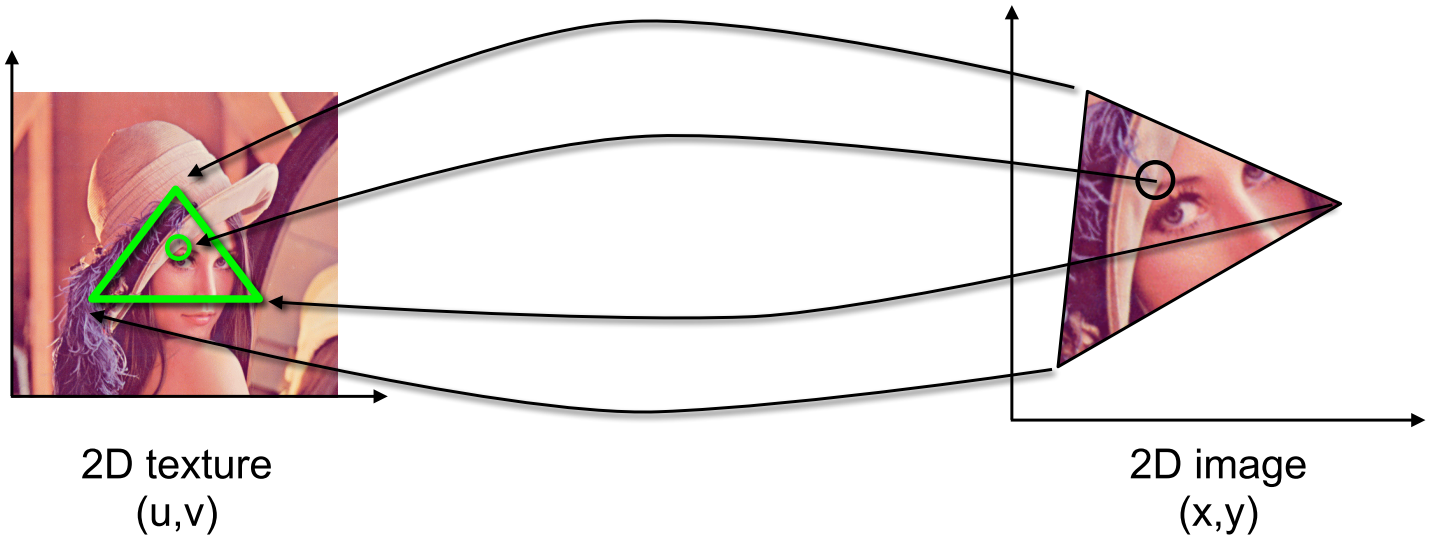
Texturing One Triangle



- Assign 2D texture coordinate $\mathbf{u} = (u, v)$ to each vertex
- Interpolate texture coordinates by barycentric coordinates

$$\mathbf{u}(\mathbf{x}) = \alpha\mathbf{u}(\mathbf{A}) + \beta\mathbf{u}(\mathbf{B}) + \gamma\mathbf{u}(\mathbf{C})$$

Texturing One Triangle

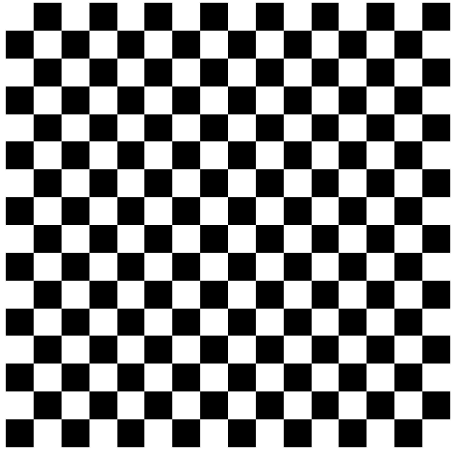


- Assign 2D texture coordinate $\mathbf{u} = (u, v)$ to each vertex
- Interpolate texture coordinates by barycentric coordinates

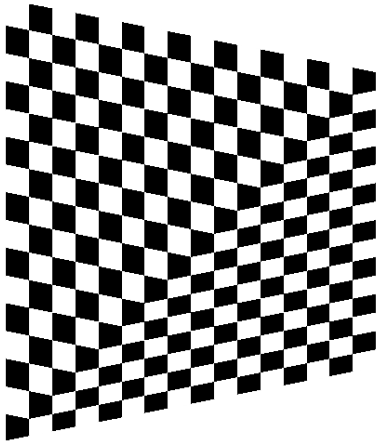
$$\mathbf{u}(\mathbf{x}) = \alpha\mathbf{u}(\mathbf{A}) + \beta\mathbf{u}(\mathbf{B}) + \gamma\mathbf{u}(\mathbf{C})$$

- Fetch color value from texture: $\mathbf{c}(\mathbf{x}) = \mathbf{texture}(\mathbf{u}(\mathbf{x}))$

Texturing One Triangle



Orthogonal view

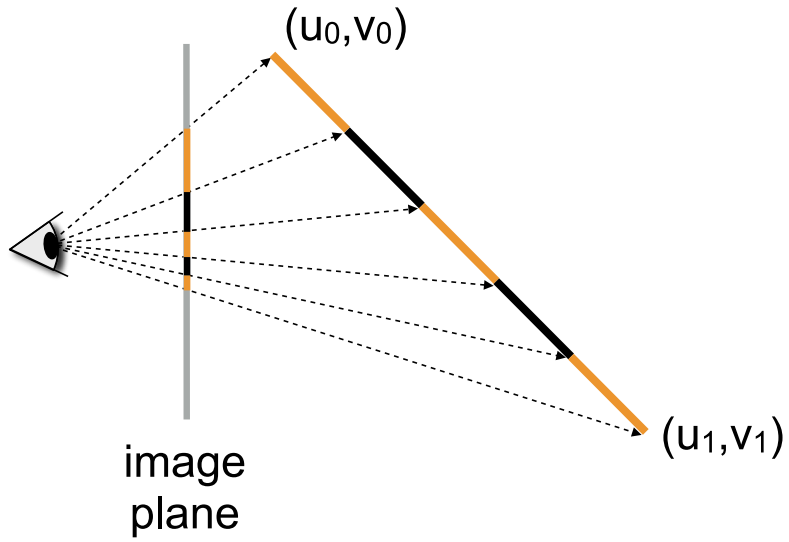


Screen-space interpolation: perspectively incorrect

Images from Akenine-Möller et al., Real-Time Rendering

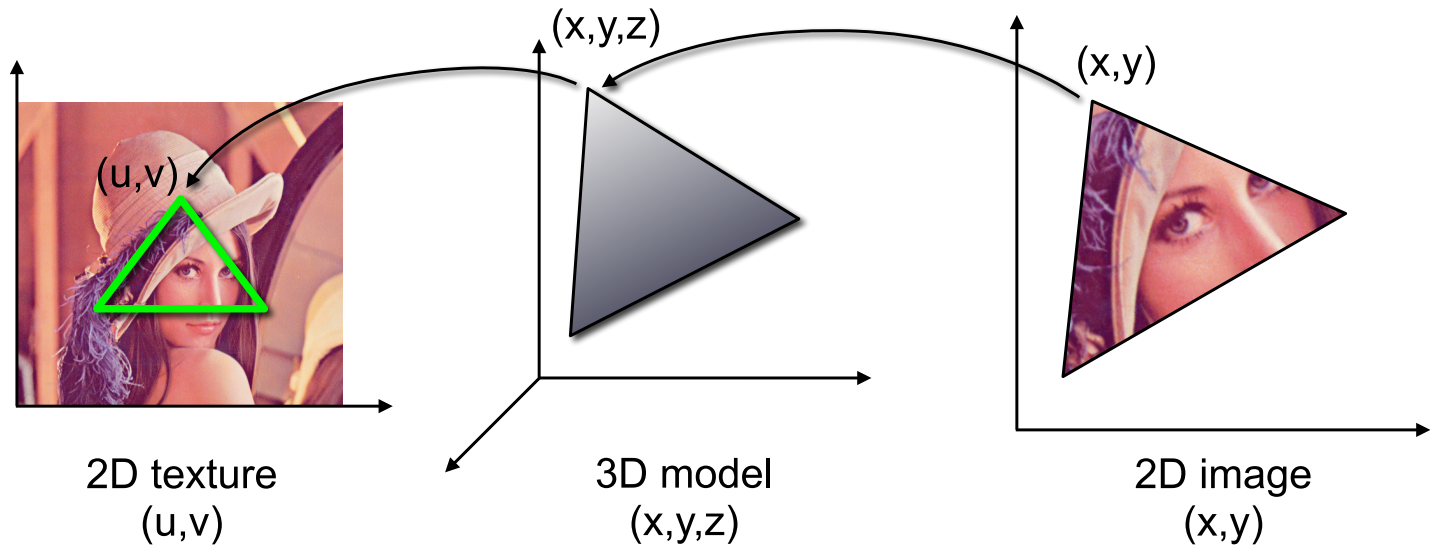
Perspective Interpolation

- Linear interpolation in world coordinates yields nonlinear interpolation in screen coordinates!
- Choose *screen-space* (`noperspective`) or *perspective* (`smooth`) interpolation for vertex shader outputs (`smooth` is default)

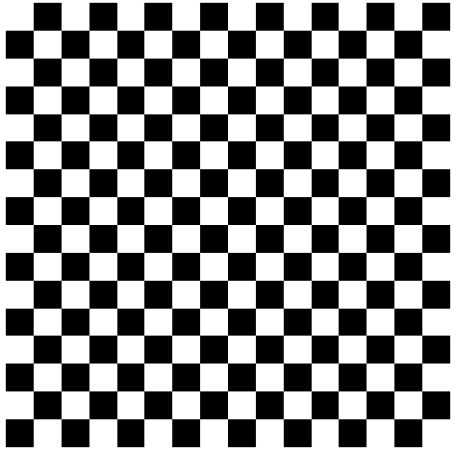


Texturing One Triangle

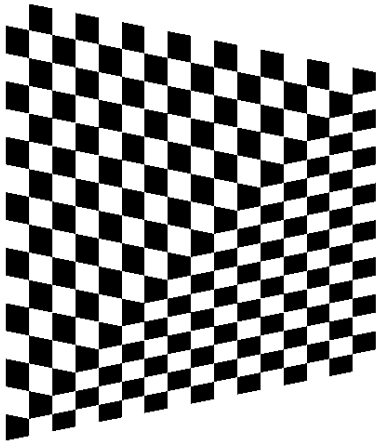
- Assign 2D texture coordinate $\mathbf{u} = (u, v)$ to each vertex
- Interpolate texture coordinates by barycentric coordinates **in 3D object space**
- Fetch color value from texture



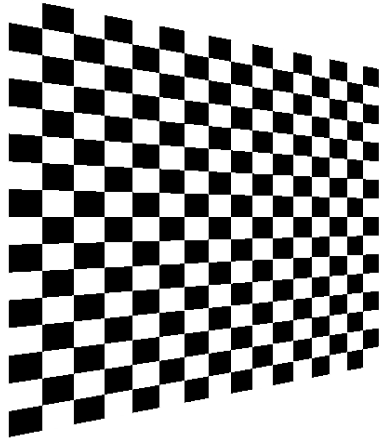
Texturing One Triangle



Orthogonal view



Screen-space interpolation: perspectively incorrect



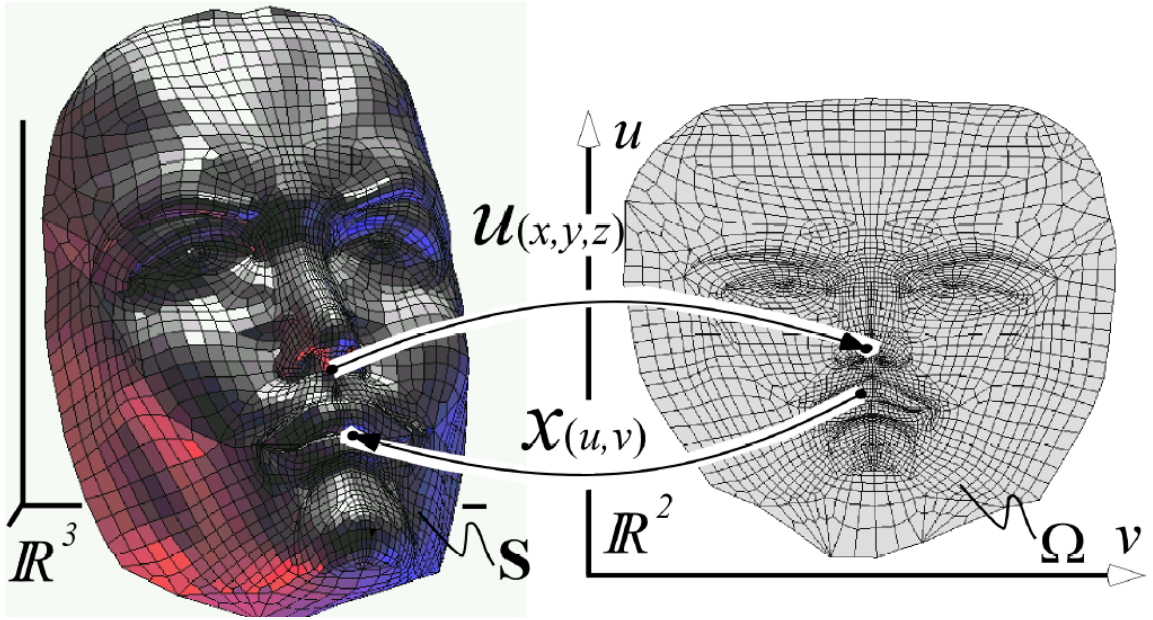
Object-space interpolation: perspectively correct

Images from Akenine-Möller et al., Real-Time Rendering

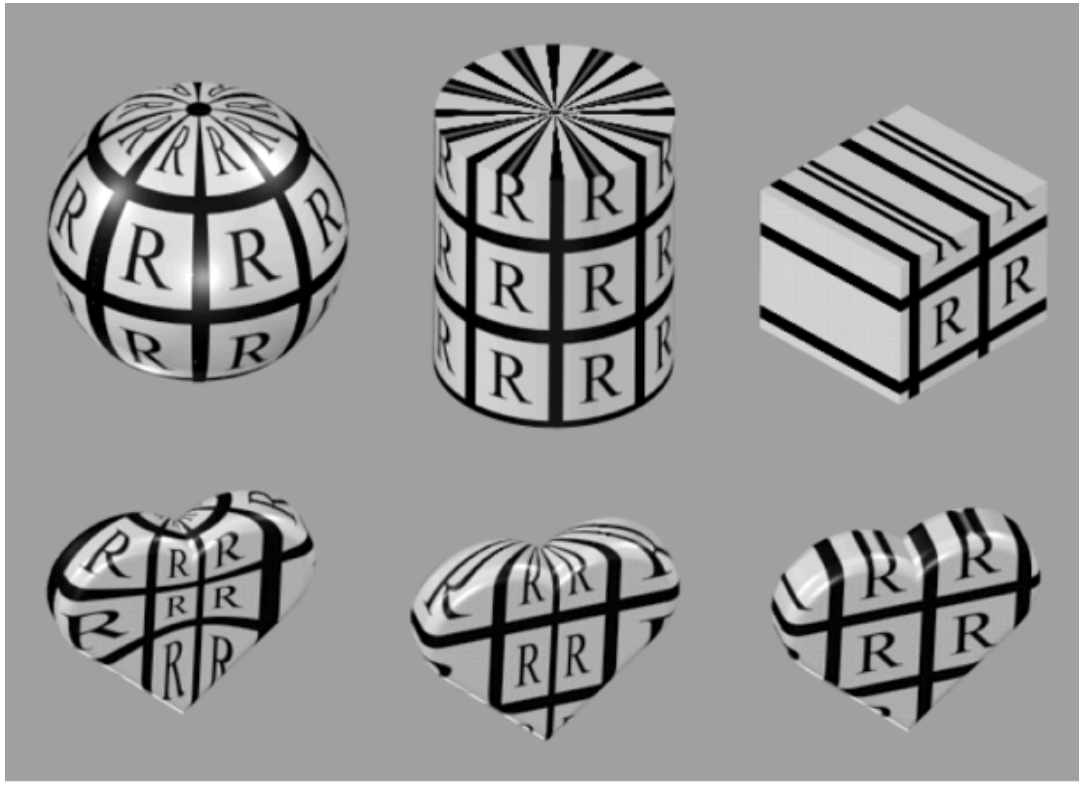
Texturing a Triangle Mesh

Texturing a Triangle Mesh

- How to find texture coordinates for each vertex?
- Find *parameterization*: Mapping between 2D texture space and 3D object space



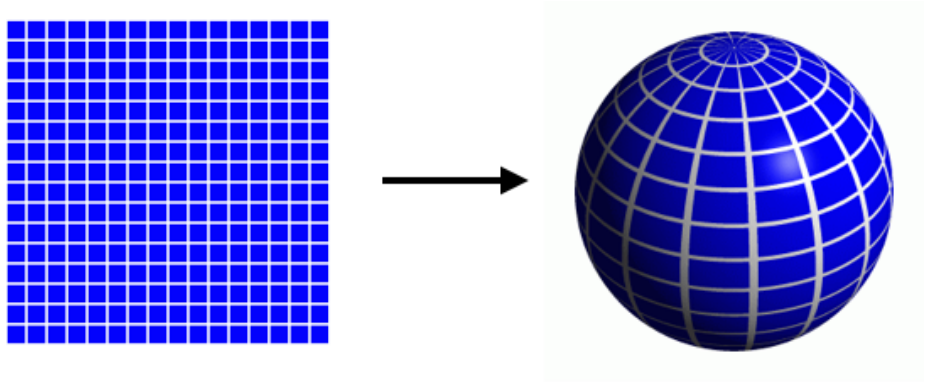
Simple Parameterizations



Images from Akenine-Möller et al., Real-Time Rendering

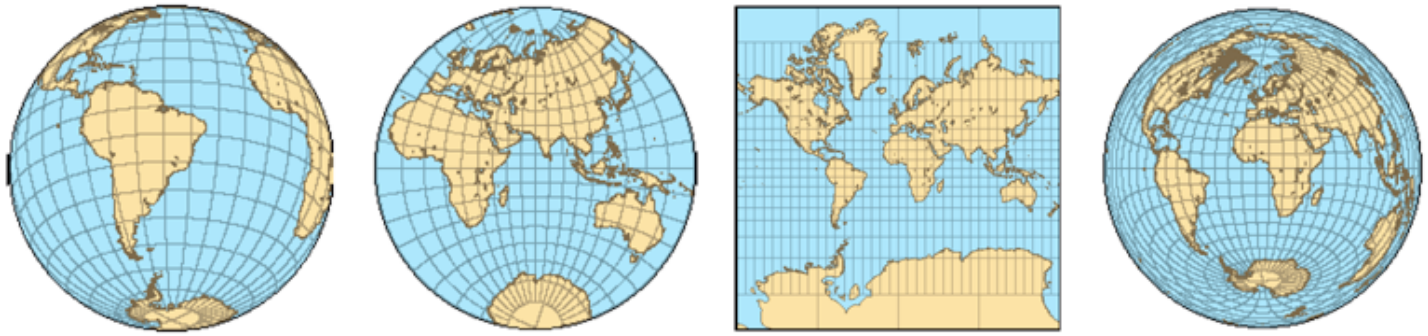
Sphere Parameterization

$$\begin{pmatrix} \phi \\ \theta \end{pmatrix} \mapsto \begin{pmatrix} \cos\phi\ \cos\theta \\ \sin\phi\ \cos\theta \\ \sin\theta \end{pmatrix}$$



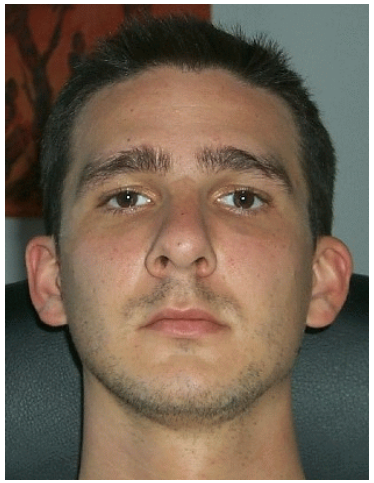
Cartography

- Many ways to parameterize a sphere!
- Some parameterizations have special properties:
  - preserve angles (*conformal*, 2nd image)
  - preserve areas (*equi-areal*, 4th image)



Floater, Hormann: Surface Parameterization: A Tutorial and Survey, 2005

Low-Distortion Parameterization

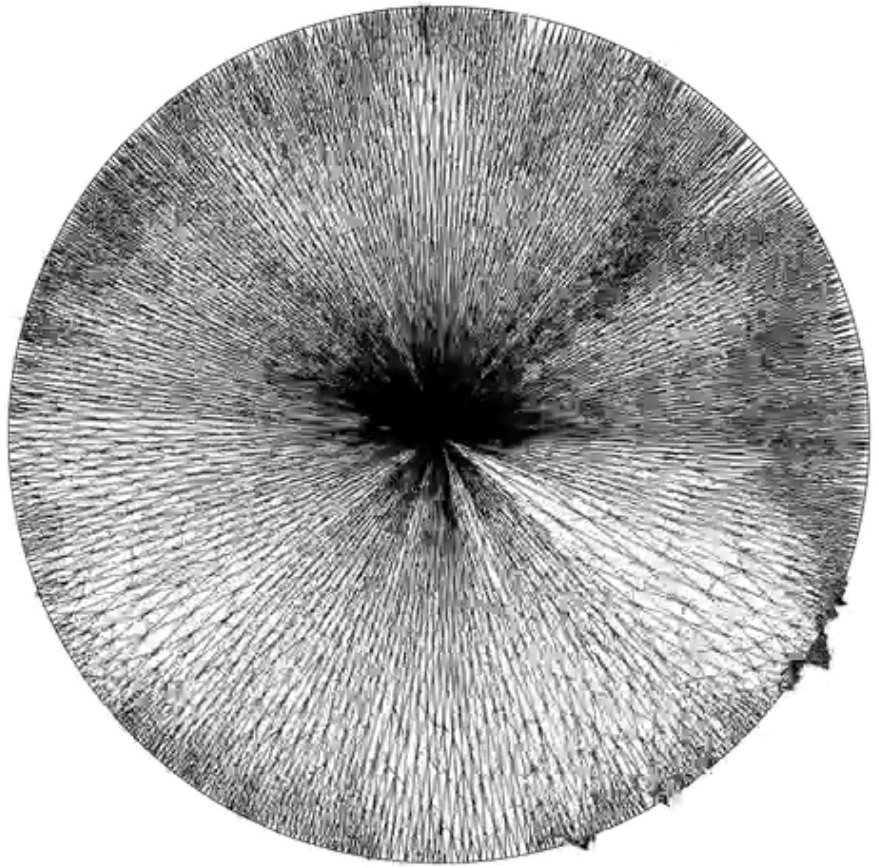


Orthogonal projection

Low-distortion parameterization

Low-Distortion Parameterization

Low-Distortion Parameterization



Texture Filtering

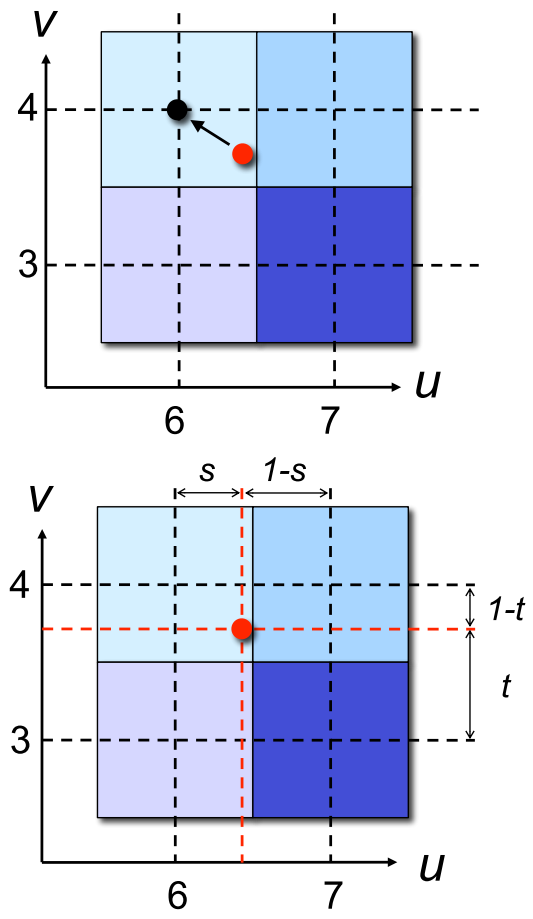
Texture Interpolation

- How to get color value from real-valued texture coordinates (u, v), such as $(u, v) = (6.4, 3.7)$?
- Round to *nearest* integer coordinate

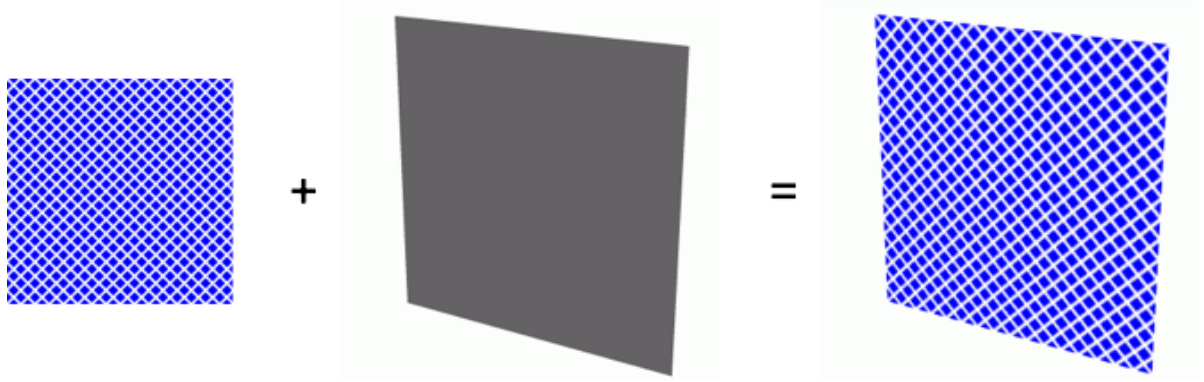
```
color = tex[6,4];
```

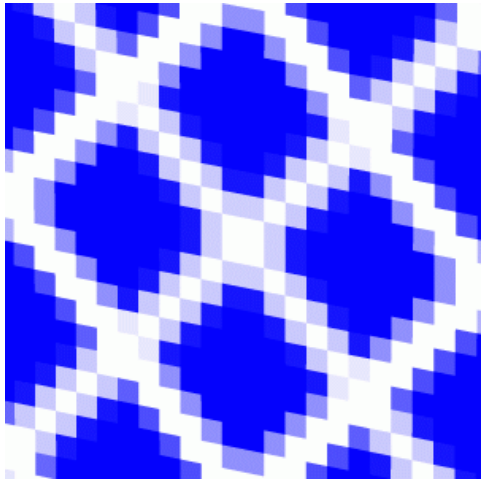
- *Bilinear interpolation* of neighboring texture pixels

```
color = (1-s)*(1-t)*tex[6,3] + (1-s)*t*tex[6,4]
            s*(1-t)*tex[7,3] +     s*t*tex[7,4];
```

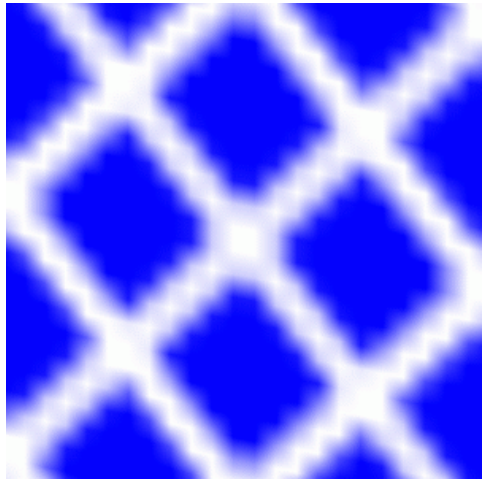


Magnification





nearest



linear

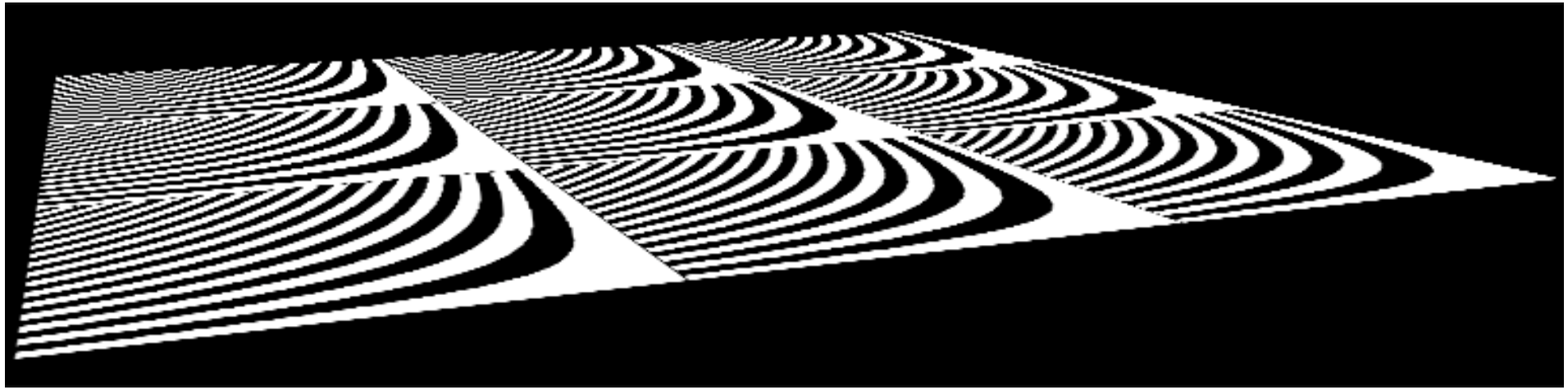
Magnification

nearest

linear

Images from Akenine-Möller et al., Real-Time Rendering

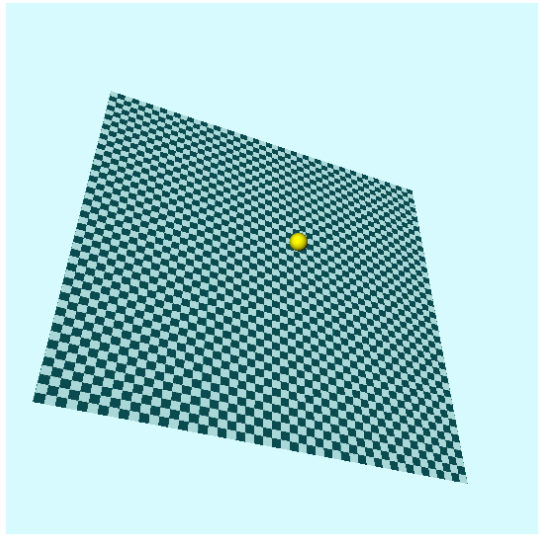
Minification

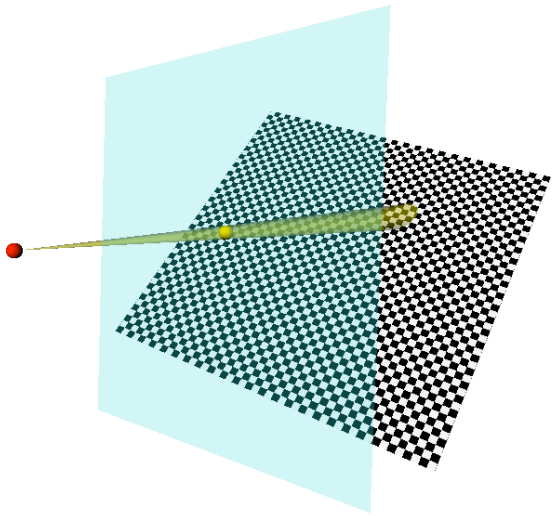


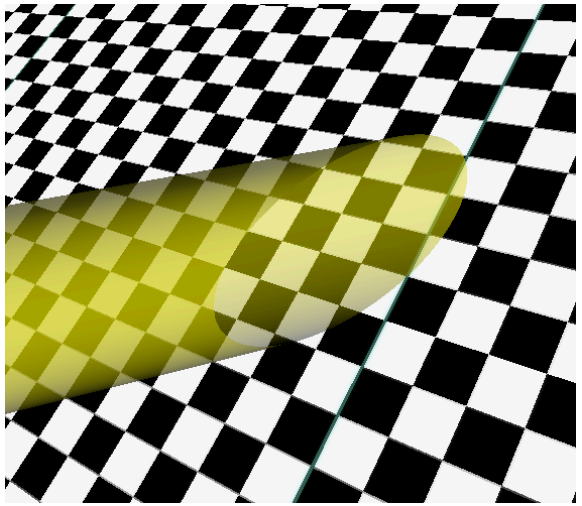
Bilinear filtering: Aliasing for minification

Minification

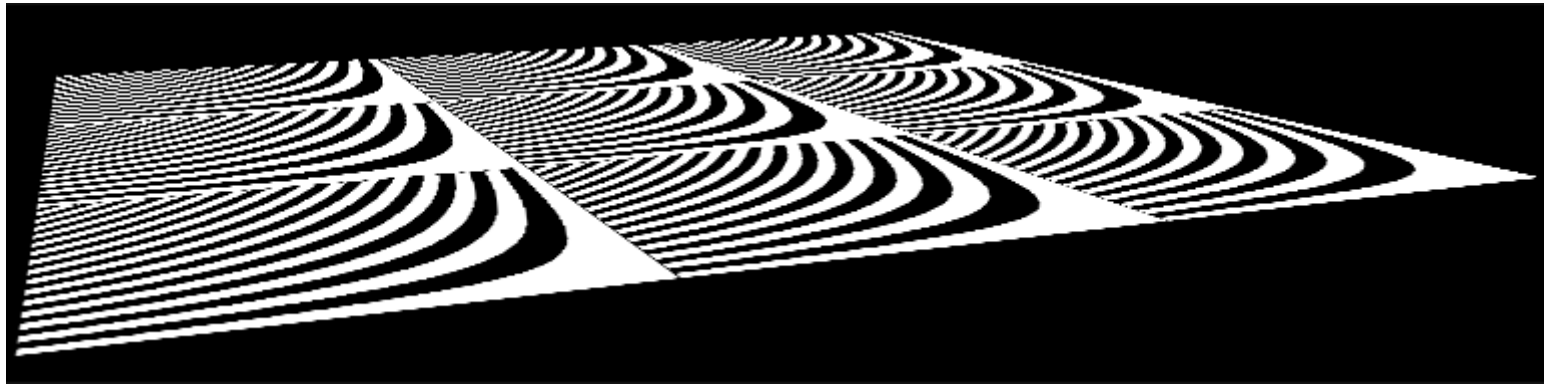
- Point sampling is the wrong model
  - Texture minification leads to aliasing
- Integrate over image pixel's area in texture space
  - Approximated by an ellipse
  - Elliptically weighted averaging (*EWA filtering*)



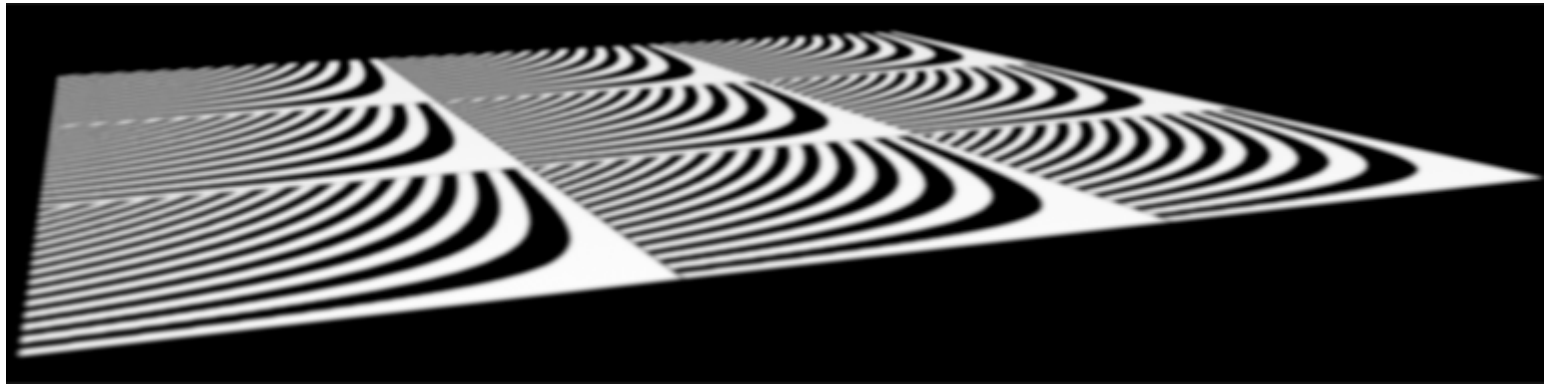




Minification

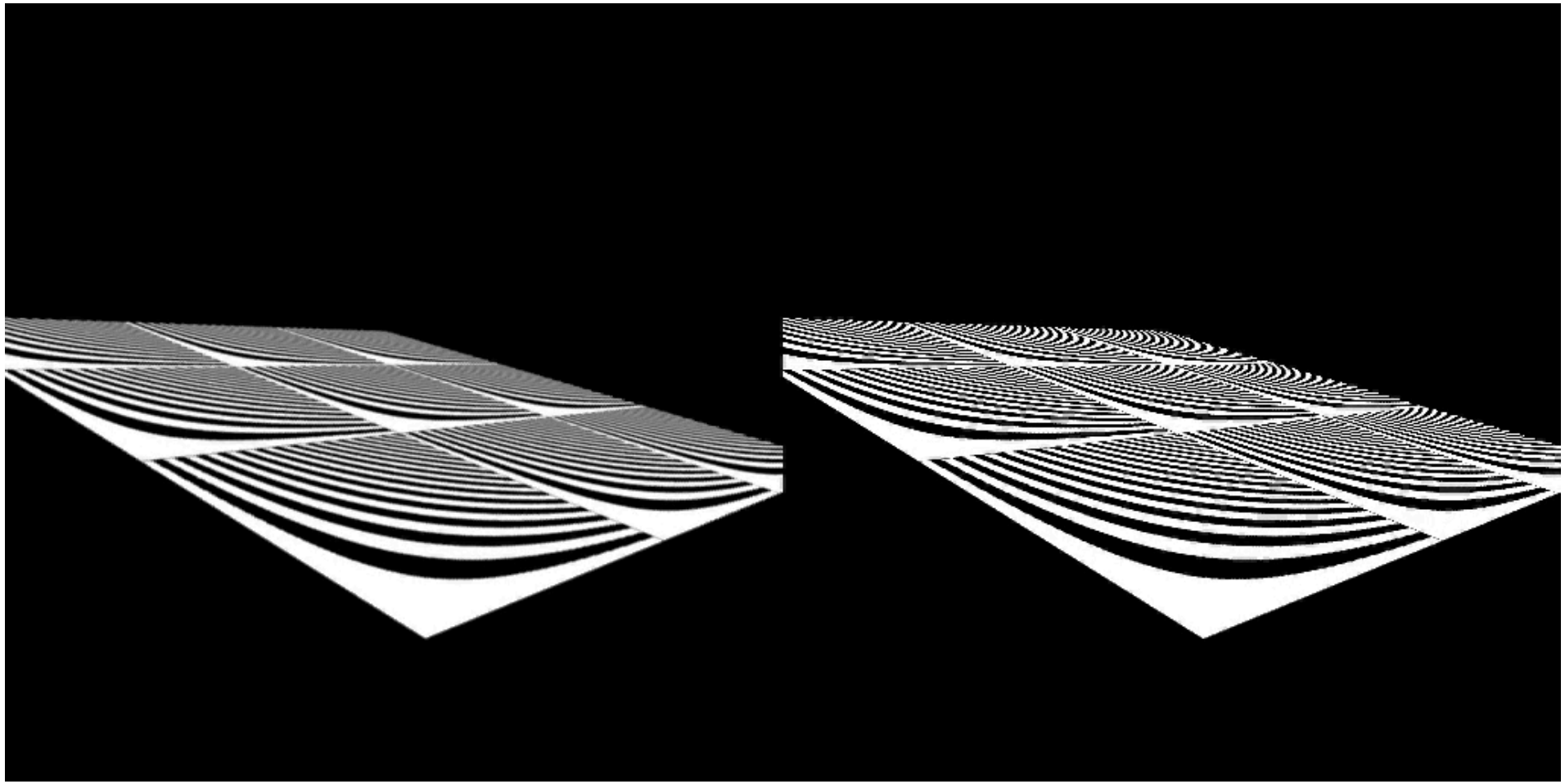


Bilinear filtering: Aliasing for minification



EWA filtering

Minification



EWA filtering

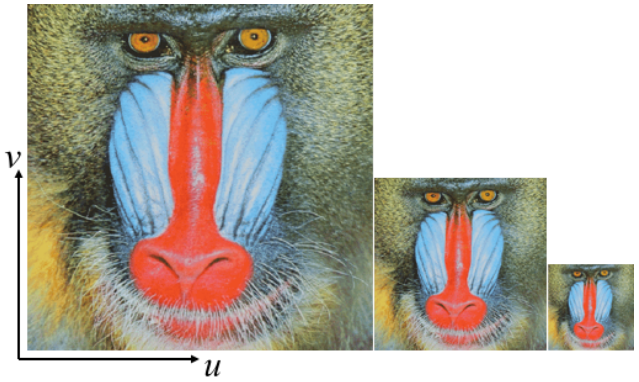
Bilinear filtering

Minification

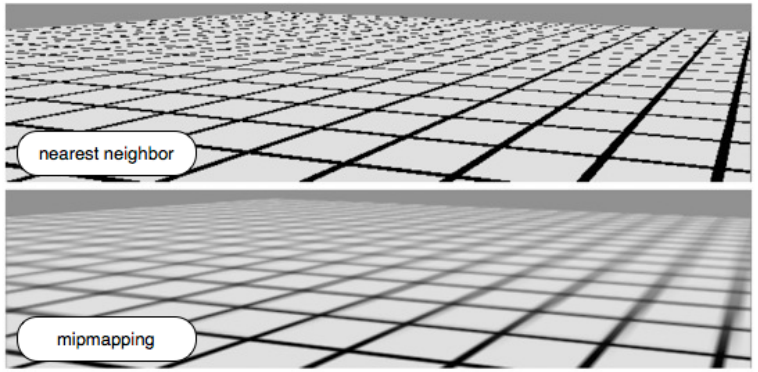
- Point sampling is the wrong model
  - Texture minification leads to aliasing
- Integrate over image pixel's area in texture space
  - Approximated by an ellipse
  - Elliptically weighted averaging (*EWA filtering*)
  - Computationally too expensive
- Approximate EWA filtering by
  - mip-mapping
  - anisotropic texture filtering

Mip-Mapping

- Store texture at multiple levels-of-detail
  - *MIP* means “multum in parvo”: many in a small space.
  - Precompute down-scaled versions of texture image
- Use lower-resolution versions when far from camera
  - OpenGL picks the most suitable image resolution for each per-pixel texture lookup based on pixel’s depth value



MIP map



Comparison

Images from Akenine-Möller et al., Real-Time Rendering

Quiz

What is the storage overhead of mip-mapping?

A: 25%

B: 33%

C: 50%

D: 100%

Demo: Texture Filtering

threejs texture filtering example

Demo: Texture Filtering

threejs anisotropic texture filtering

Demo: Texture + Diffuse Lighting

Press `Shift–Enter` to compile shaders