Textures (Part I)

Computer Graphics

Yu-Ting Wu

Outline

- Overview
- Texture data (Part I)
- Texture filtering
- Applications

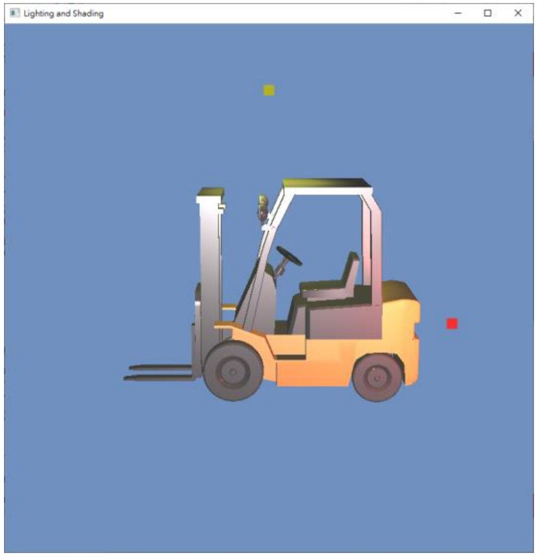
- OpenGL implementation (Part II)

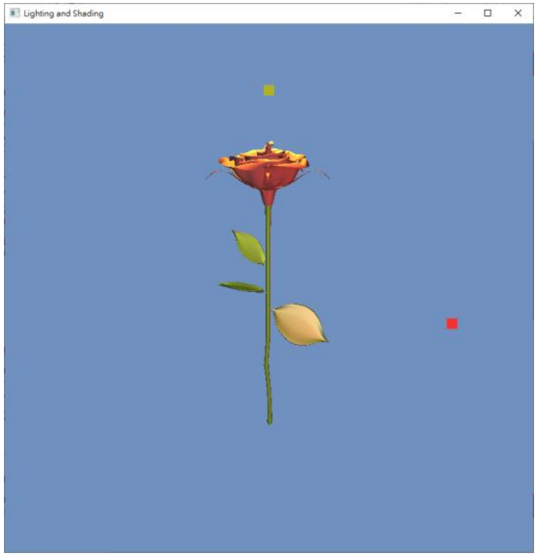
Outline

- **Overview**
- Texture data
- Texture filtering
- Applications
- OpenGL implementation

Why Do We Need Textures

- So far, we have described object colors using their reflectance functions
  - Subdivide an object into several parts, each has its reflectance properties (e.g., different diffuse and specular colors)



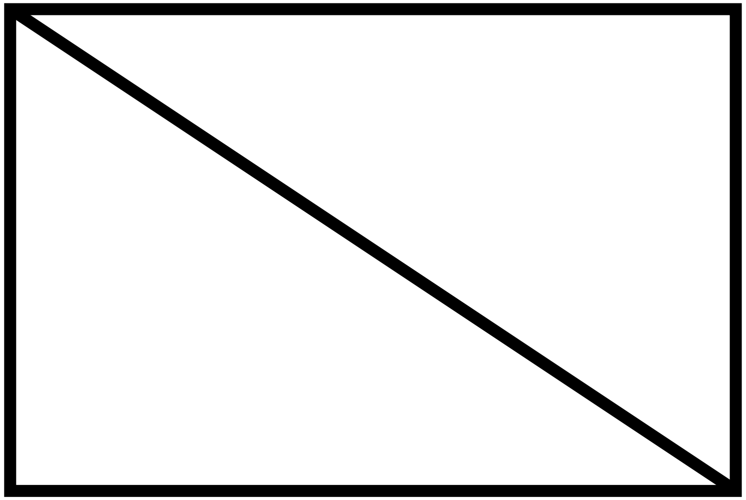


Why Do We Need Textures (cont.)

- Consider the following cases
  - Do we need (or can we) to finely subdivide the object?

Textures

- Can be used to represent **spatially-varying** data
- Can **decouple** materials from the geometry



Geometry: two triangles
Material: Kd(1, 1, 1)

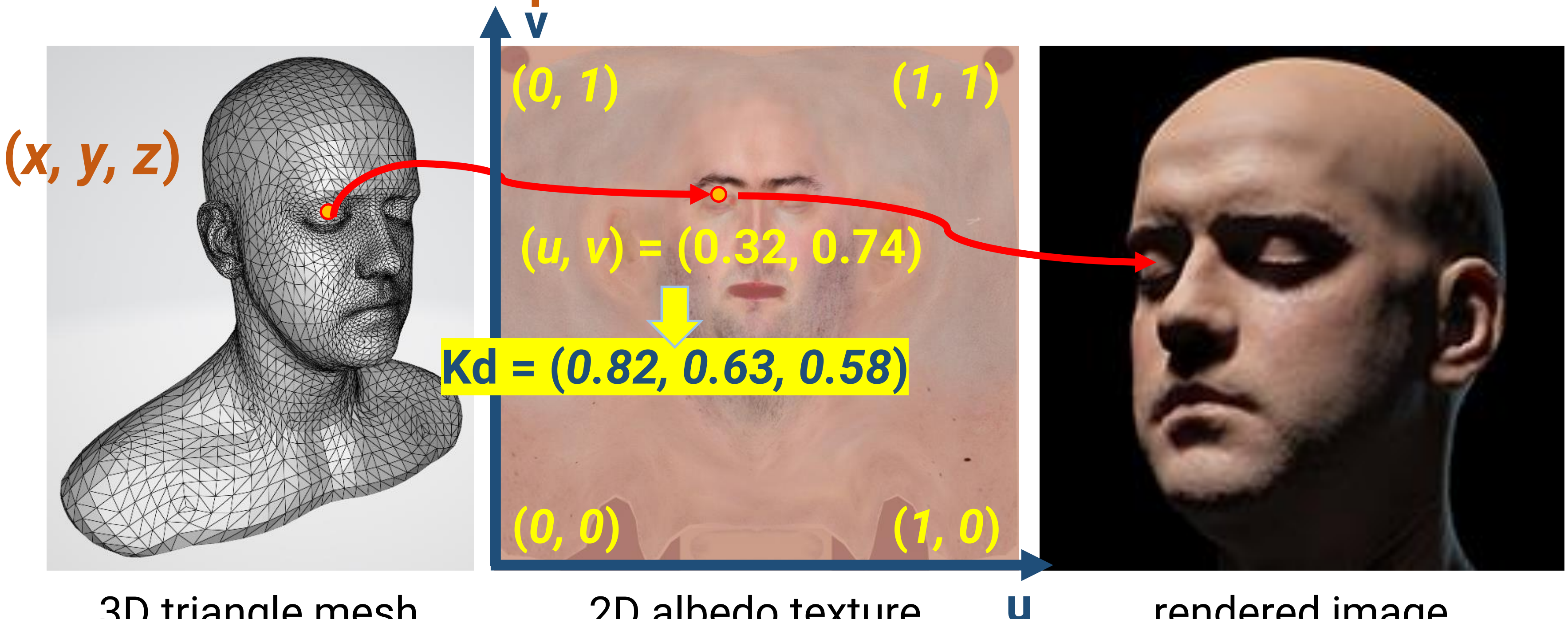
+

2D image texture

= **complex appearance**

Texture Coordinate

- A coordinate to look up the texture
- The way to map a point on an **arbitrary 3D surface** to a pixel (texel) on an **image** texture
  - Need **surface parameterization**



3D triangle mesh | 2D albedo texture | rendered image

Texture Coordinate (cont.)

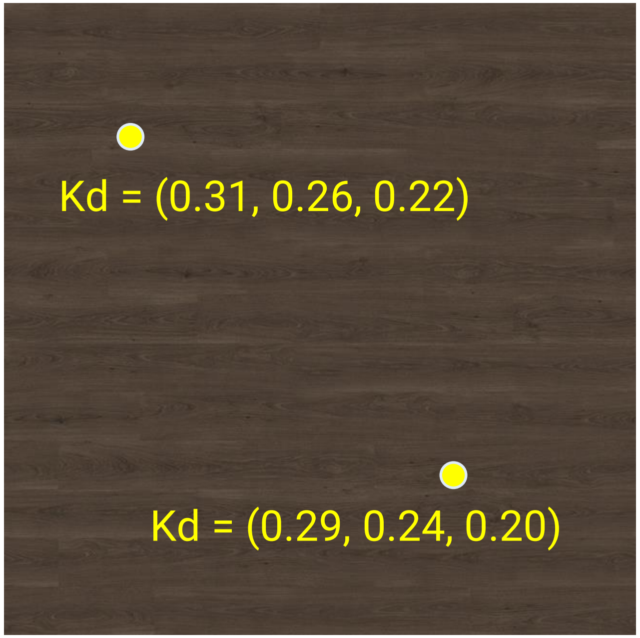
- A coordinate to look up the texture
- The way to map a point on an **arbitrary 3D surface** to a pixel (texel) on an **image** texture
  - Need **surface parameterization**
  - Usually produced by 3D artists

Types of Textures

- **2D image texture (most common)**
  - Material data (surface albedo, specularness, roughness)
  - Geometry data (surface bump, normals, height)
  - Lighting data (lightmap, ambient occlusion map)
- **3D volume texture**
  - Spatial data (participating media, collision detection)
- **Cubemap**
  - Spherical data (skybox, reflection probe)

Textures (cont.)

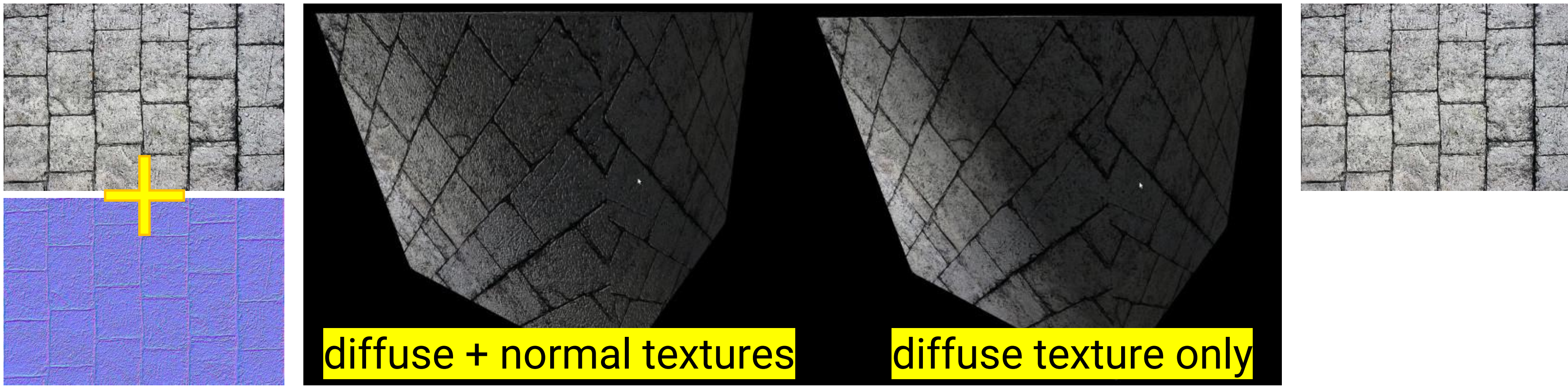
- 2D image texture for spatially-varying material

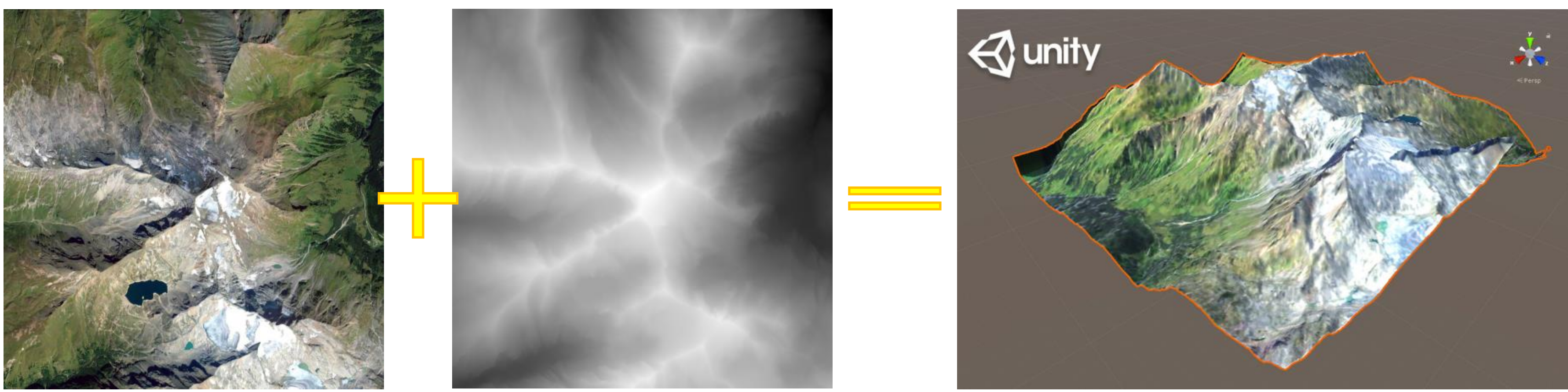


diffuse coefficient (Kd)

Types of Textures (cont.)

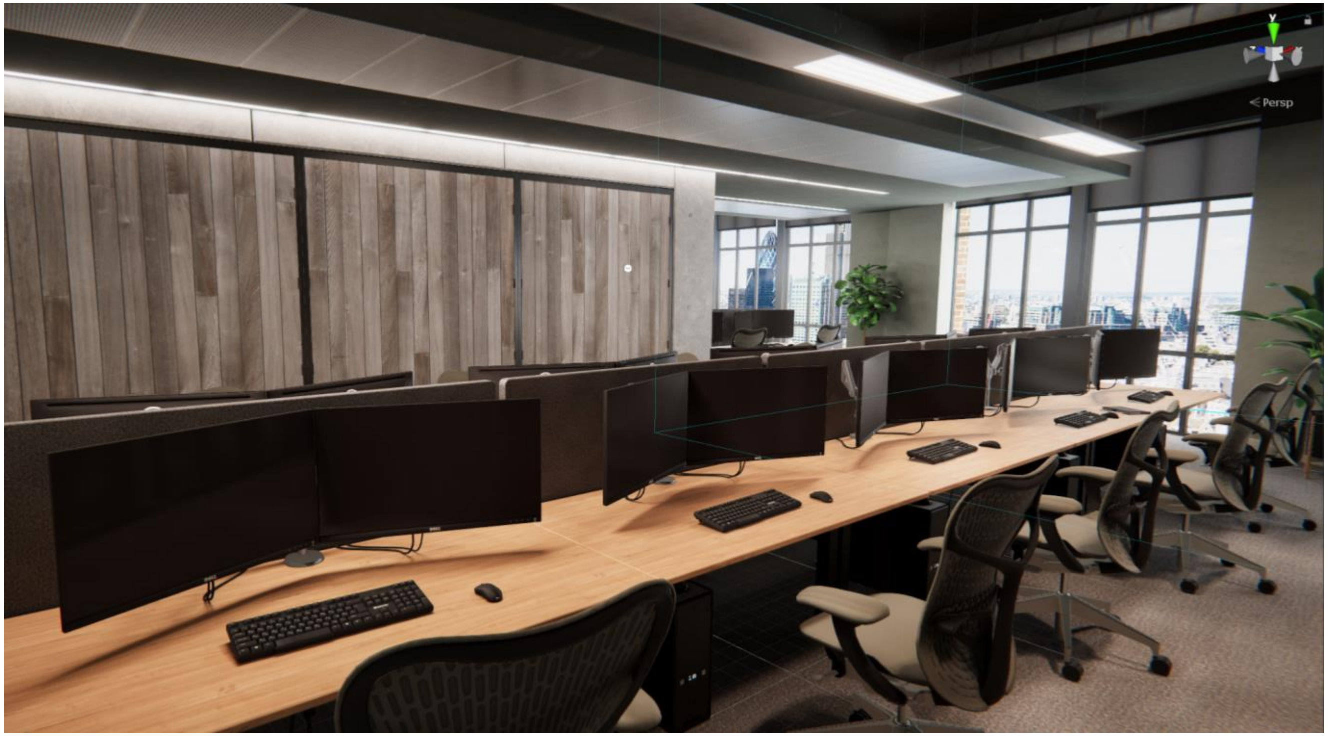
- 2D image texture for spatially-geometry data



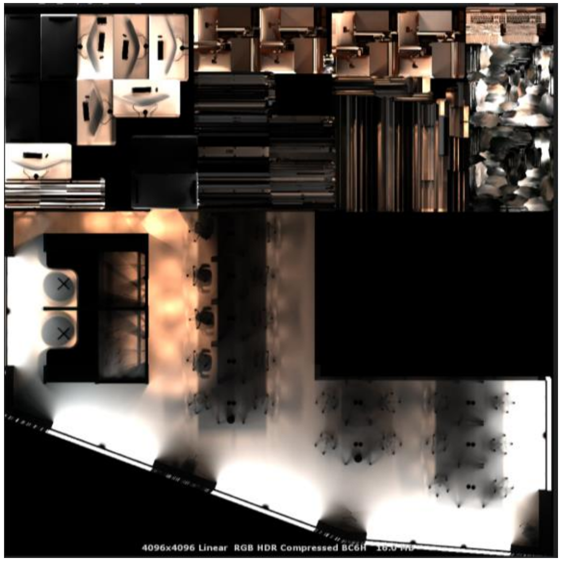


Types of Textures (cont.)

- 2D image texture for precomputed lighting data



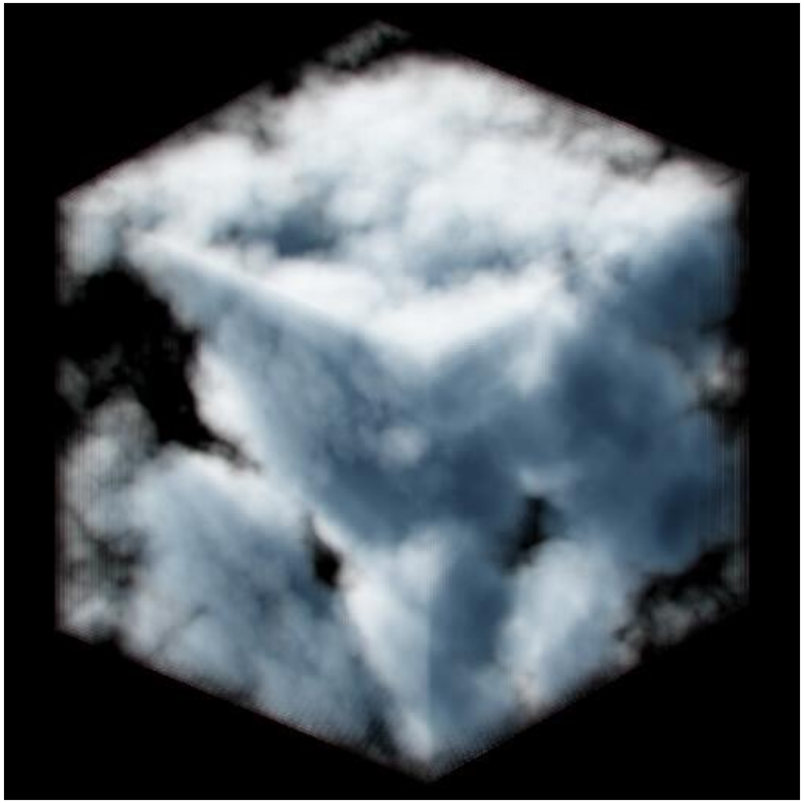
real-time rendered result

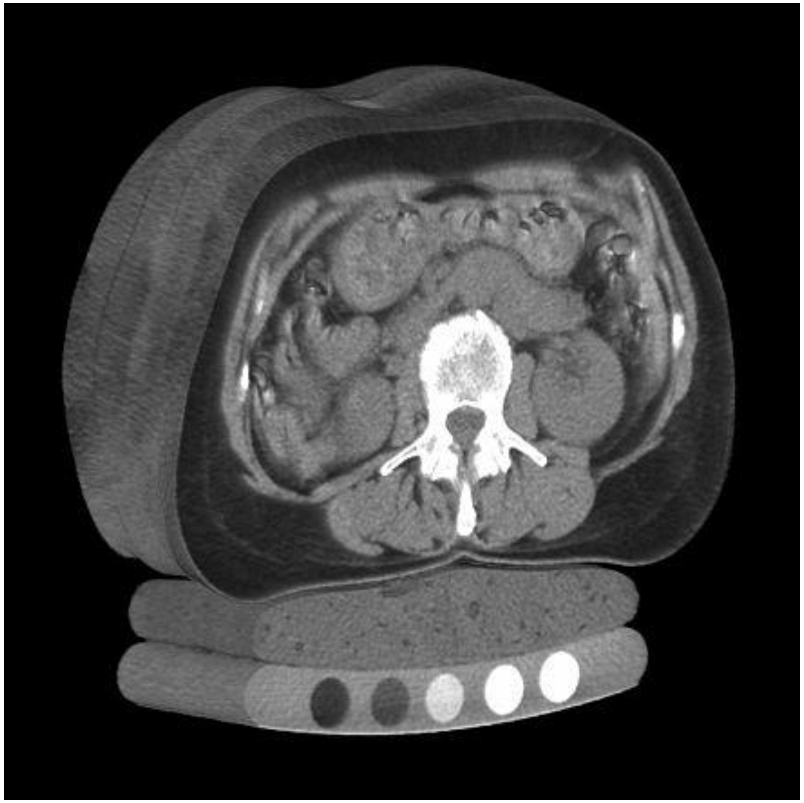


precomputed lightmaps

Types of Textures (cont.)

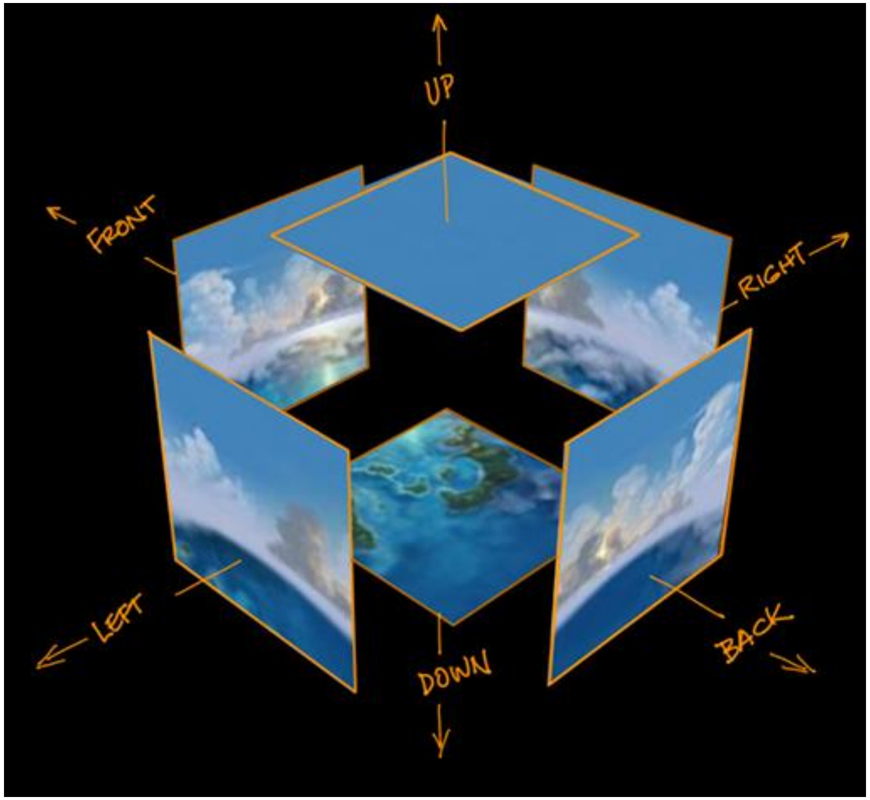
- 3D volume texture
  - Lookup by a 3D texture coordinate (u, v, s)

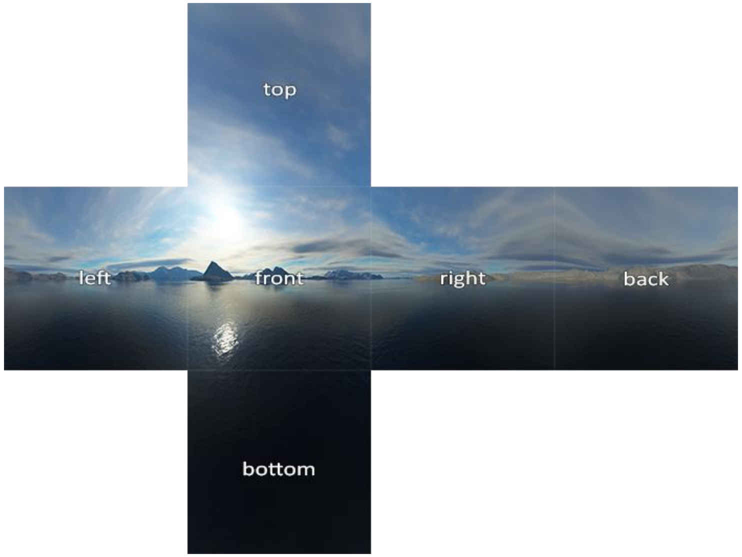




Types of Textures (cont.)

- Cubemap





Outline

- Overview
- **Texture data**
- Texture filtering
- Applications
- OpenGL implementation

Texture Data in Wavefront OBJ File

- TexCube.obj

TexCube.obj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

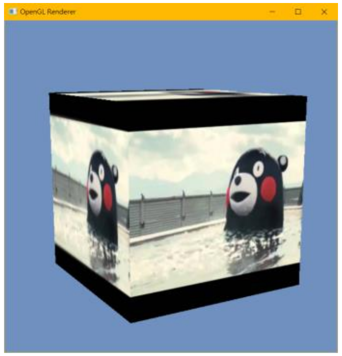
```
# Blender v2.76 (sub 0) OBJ File: ''
# www.blender.org
mtllib TexCube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000

vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0

vn 0.000000 -1.000000 0.000000
vn 0.000000 1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -0.000000 0.000000 1.000000
vn -1.000000 -0.000000 -0.000000
vn 0.000000 0.000000 -1.000000
```

vertex texture coordinate declaration

f P/T/N P/T/N P/T/N

```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

face data
(adjacency, submesh)



Texture Data in Wavefront OBJ File (cont.)

```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

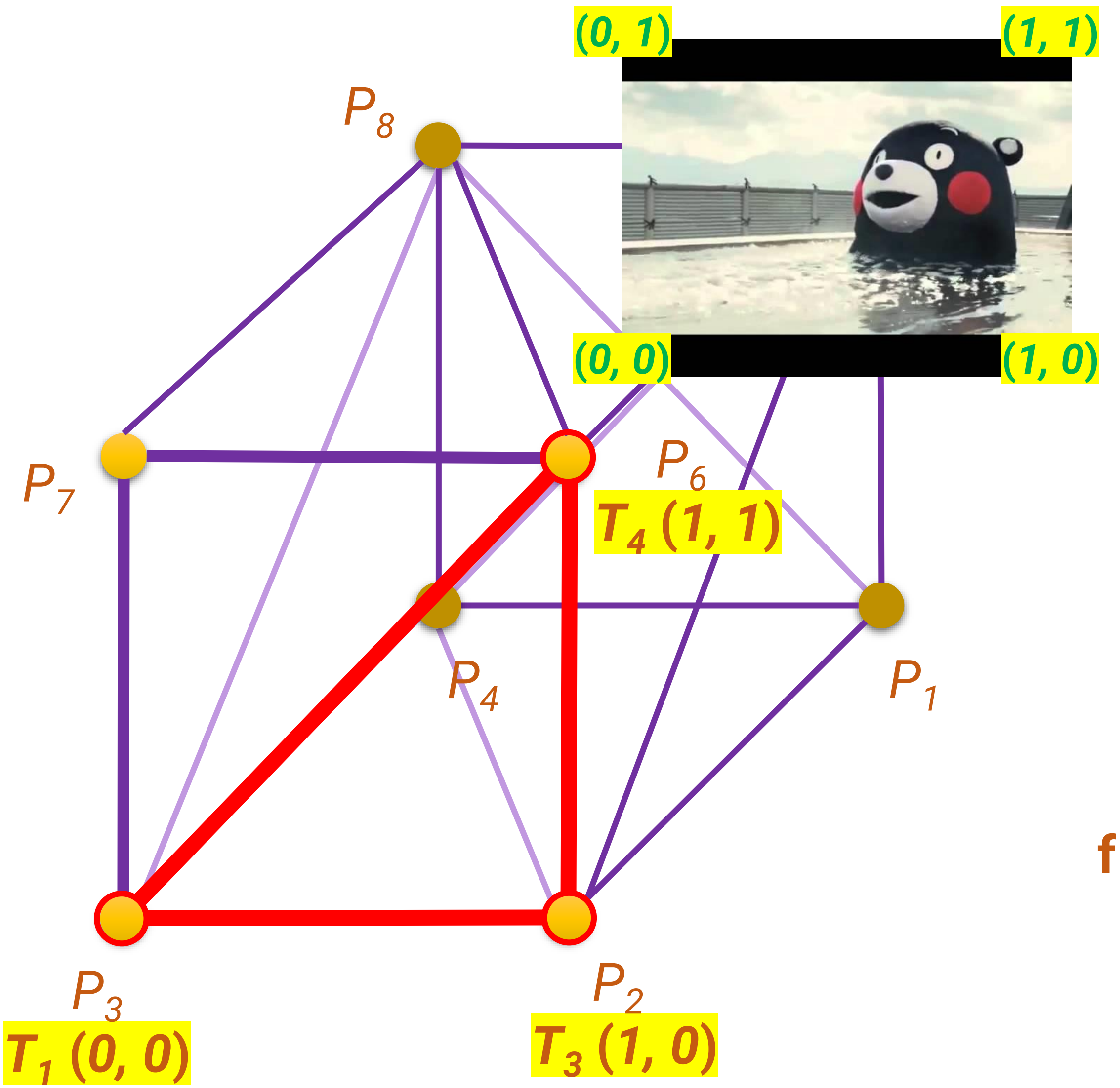
TexCube.mtl - 記事本

檔案(F) 編輯(E) 格式(O) 檢視

```
newmtl cubeMtl
  Ns 30.0000
  Ka 0.2 0.2 0.2
  Kd 1 1 1
  Ks 1 1 1
  map_Kd kumamon.jpg
```

kumamon.jpg

Interpret the Texture Data



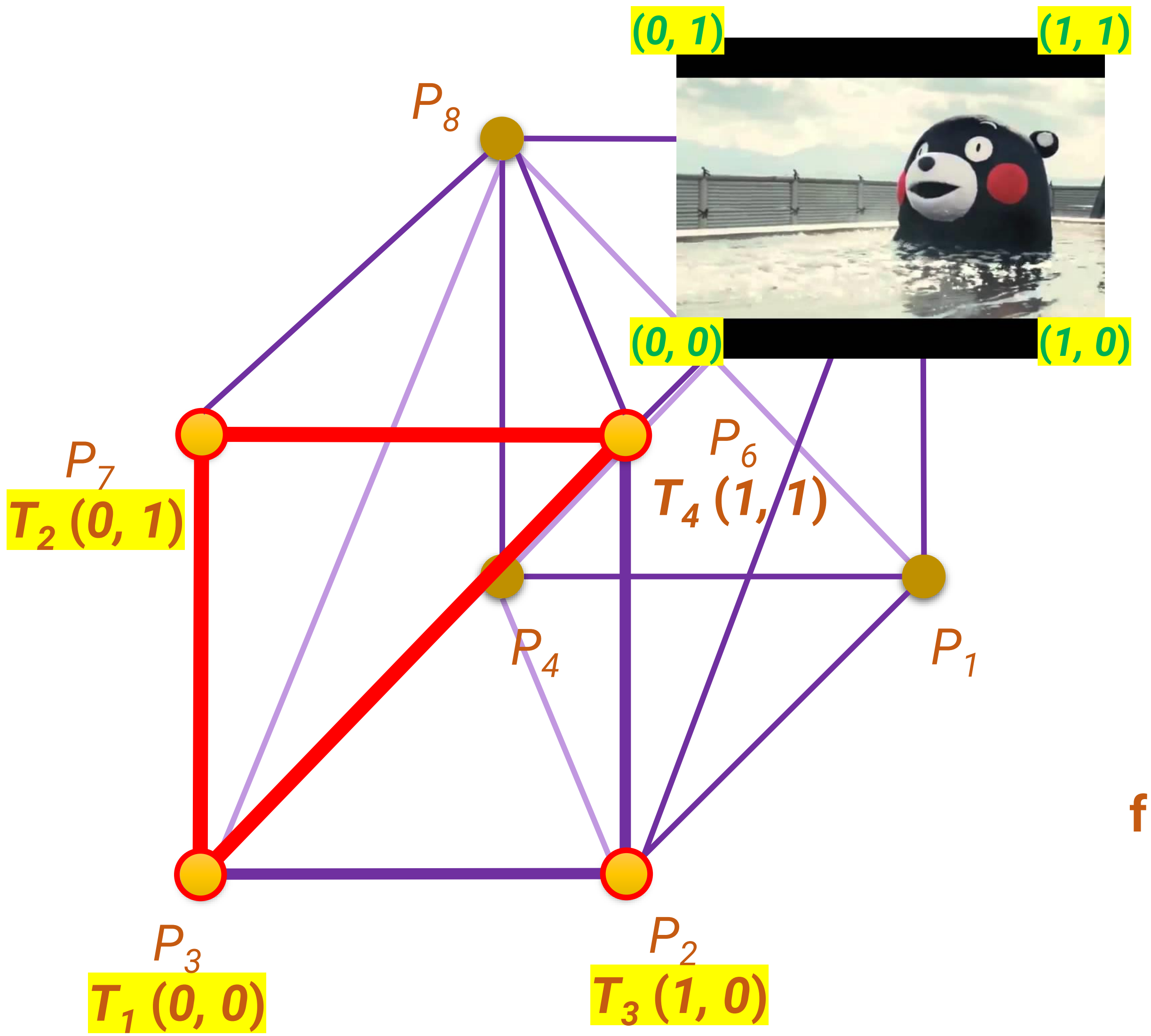

```
vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0
```

```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

	vertex1	vertex2	vertex3
f	**P/T/N**	**P/T/N**	**P/T/N**

P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Interpret the Texture Data (cont.)




```
vt 0.0 0.0
vt 0.0 1.0
vt 1.0 0.0
vt 1.0 1.0
```

```
usemtl cubeMtl
f 8/2/2 7/1/2 6/3/2
f 5/4/2 8/2/2 6/3/2
f 2/4/1 3/2/1 4/1/1
f 1/3/1 2/4/1 4/1/1
f 2/3/4 6/4/4 3/1/4
f 6/4/4 7/2/4 3/1/4
f 5/4/3 6/2/3 2/1/3
f 1/3/3 5/4/3 2/1/3
f 3/3/5 7/4/5 8/2/5
f 4/1/5 3/3/5 8/2/5
f 5/2/6 1/1/6 8/4/6
f 1/1/6 4/3/6 8/4/6
```

	vertex1	vertex2	vertex3
f	P/T/N	P/T/N	P/T/N

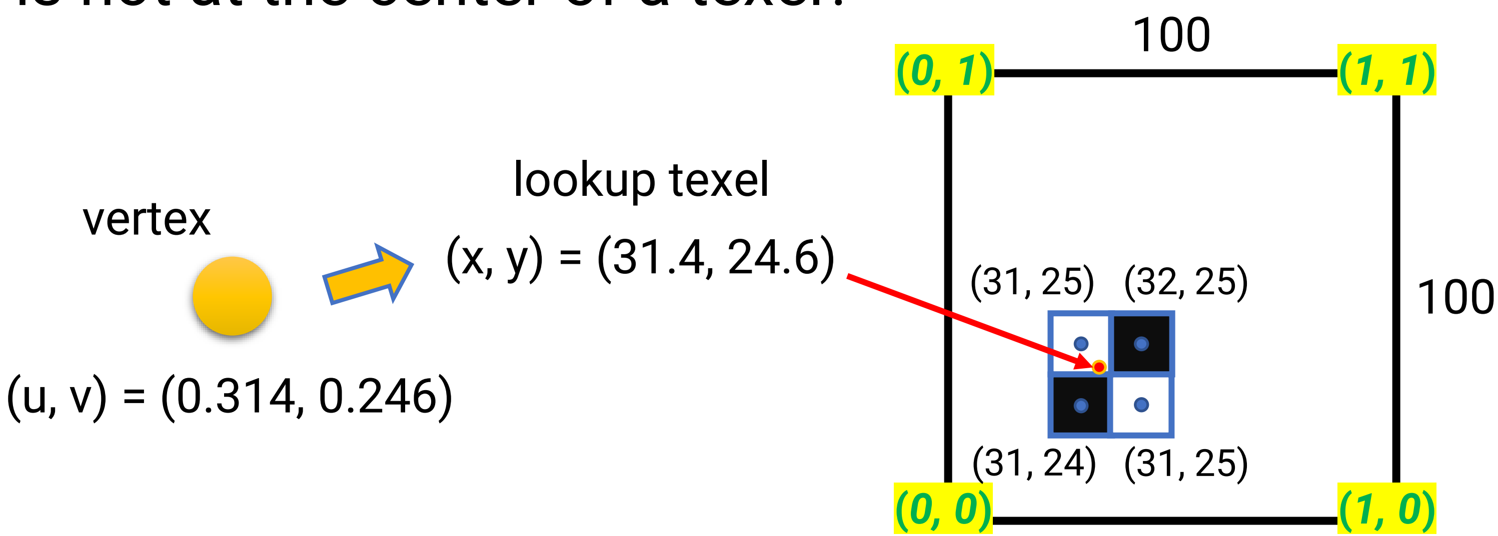
P: index of vertex position
T: index of texture coordinate
N: index of vertex normal

Outline

- Overview
- Texture data
- **Texture filtering**
- Applications
- OpenGL implementation

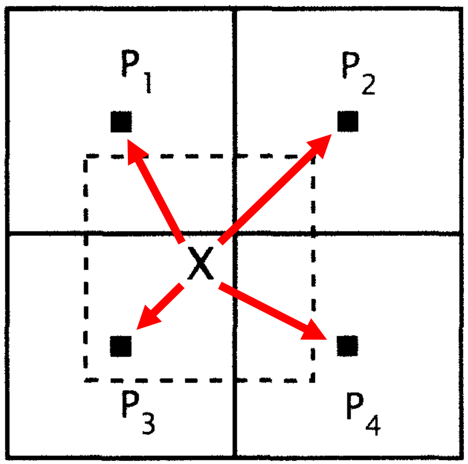
Texture Filtering

- Like an image, the content in a 2D texture is **discretely** represented by texels
- The texture coordinates can be **continuous** (especially after interpolation by the rasterization)
- How to determine the texture value if the lookup point is not at the center of a texel?



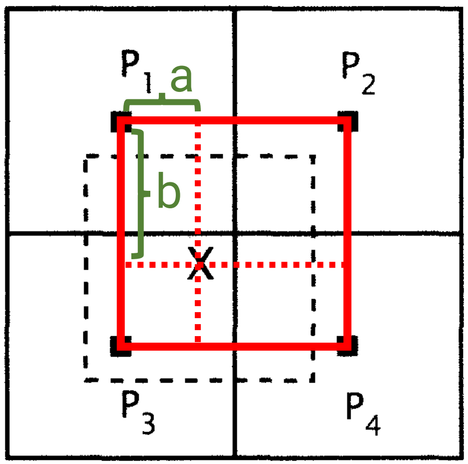
Texture Filtering (cont.)

- **Strategies**
  - **Nearest neighbor**
  - **Bilinear interpolation**



nearest neighbor

P_3 is closest
Use P_3's pixel value

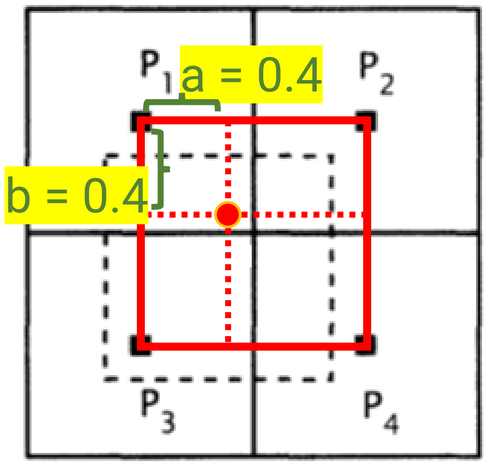


bilinear interpolation

$$(1-a)(1-b)P_1 + (a)(1-b)P_2 + (1-a)(b)P_3 + (a)(b)P_4$$

Texture Filtering (cont.)

- Example

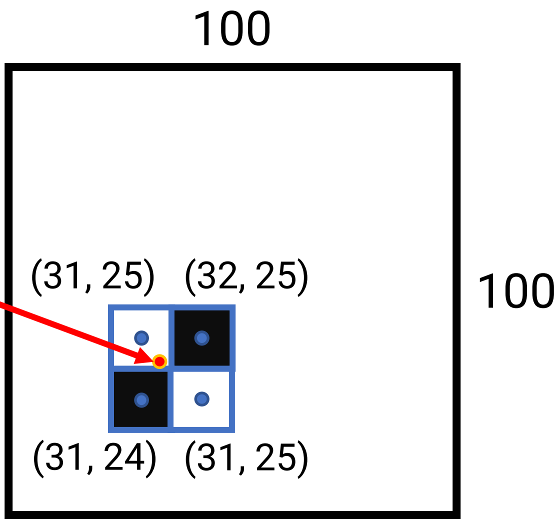


bilinear interpolation

$(1\text{-}a)(1\text{-}b)P_1 + (a)(1\text{-}b)P_2 + (1\text{-}a)(b)P_3 + (a)(b)P_4$

lookup texel

(x, y) = (31.4, 24.6)



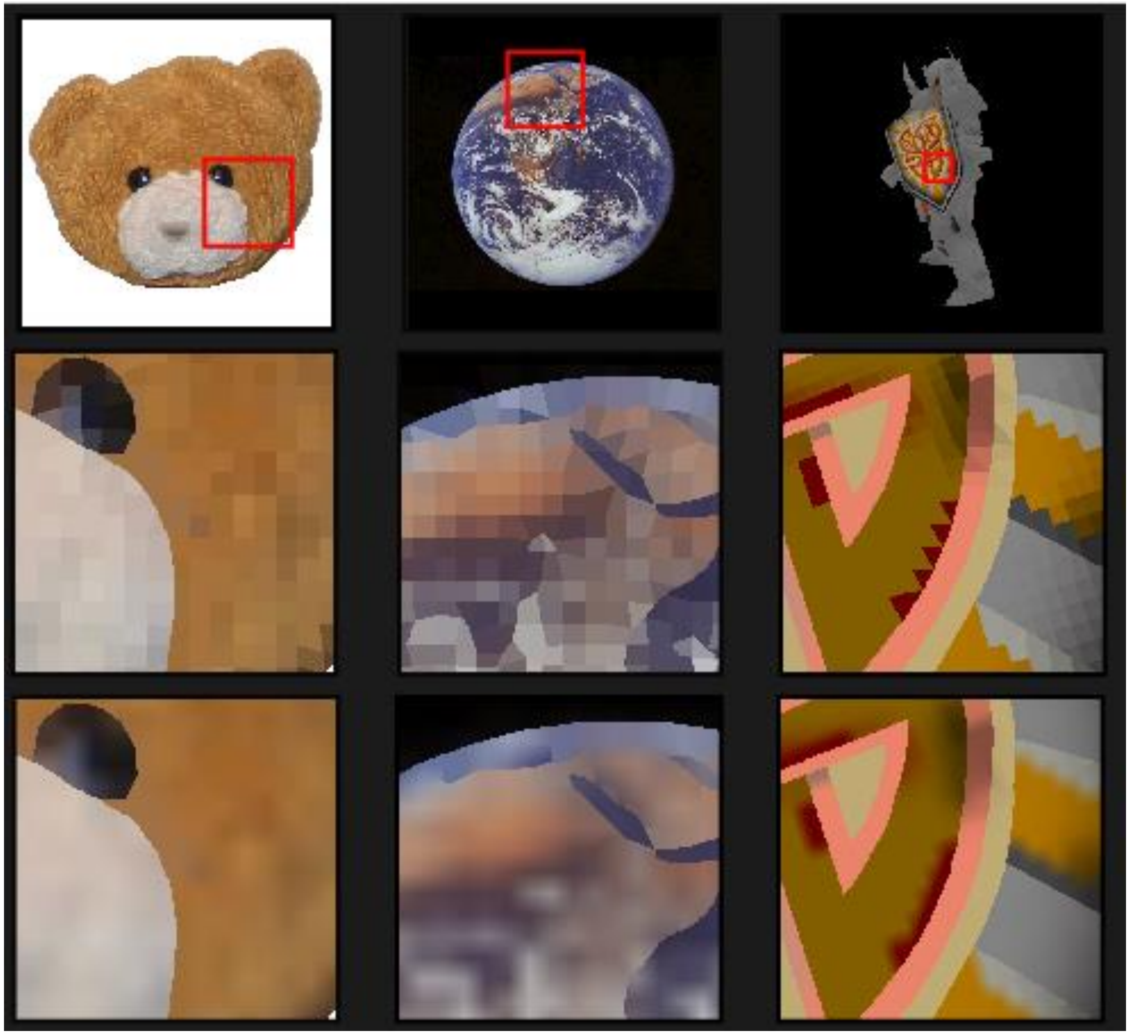
nearest neighbor: use color of (31,25)

bilinear: compute

$(1\text{-}a)(1\text{-}b)P_1 + (a)(1\text{-}b)P_2 + (1\text{-}a)(b)P_3 + (a)(b)P_4$

0.36 0.24 0.24 0.16

Texture Filtering (cont.)

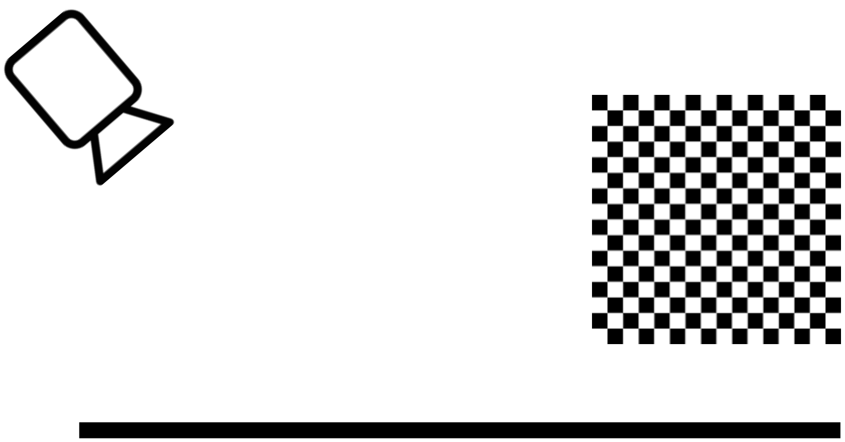


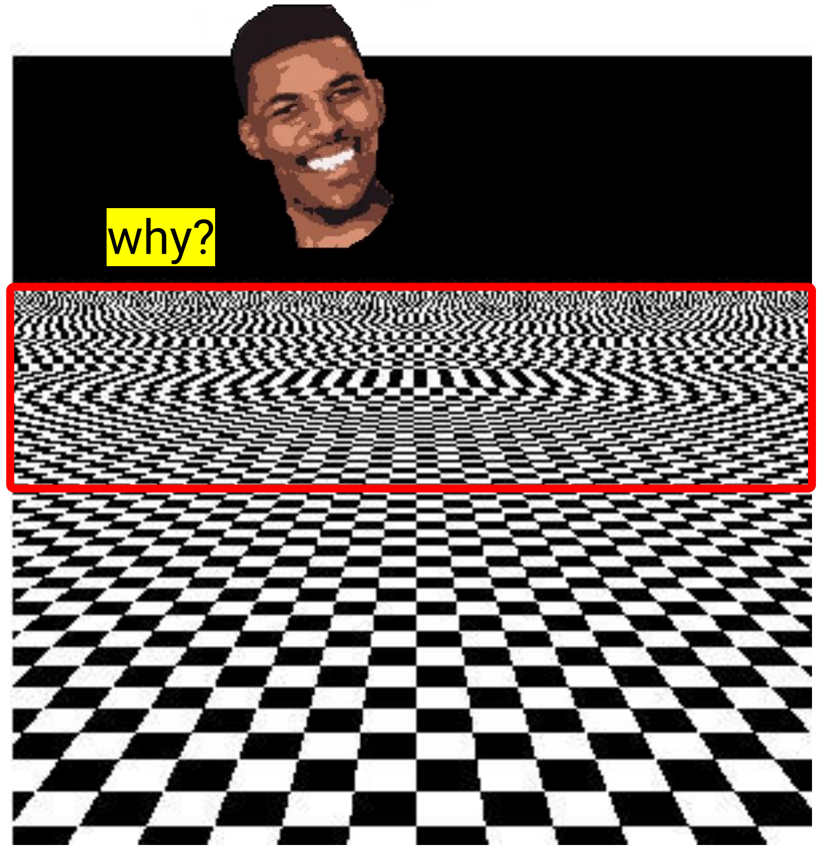
nearest neighbor

bilinear interpolation

Problems with Texture Mapping

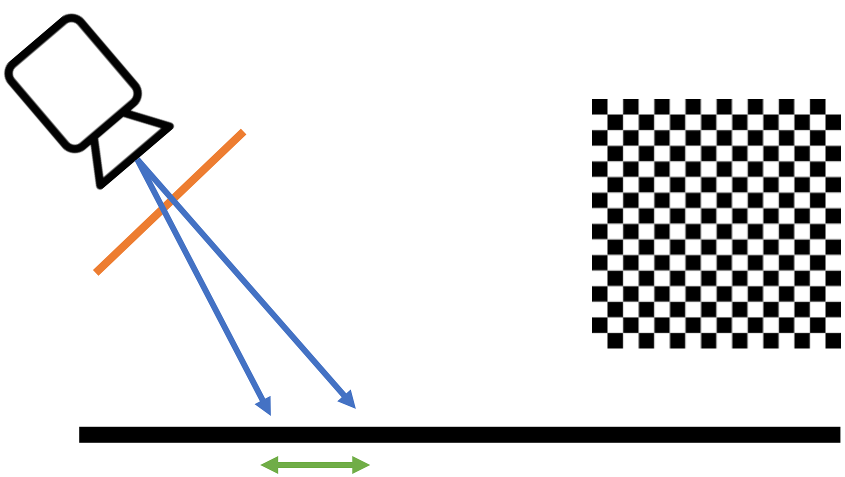
- Consider the following plane with a check-board pattern texture

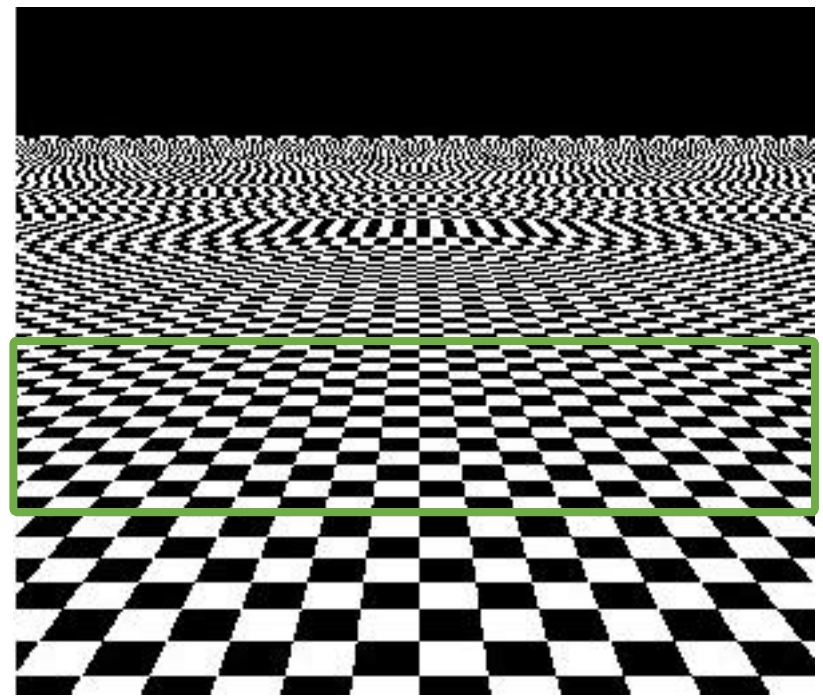




Texture Aliasing (cont.)

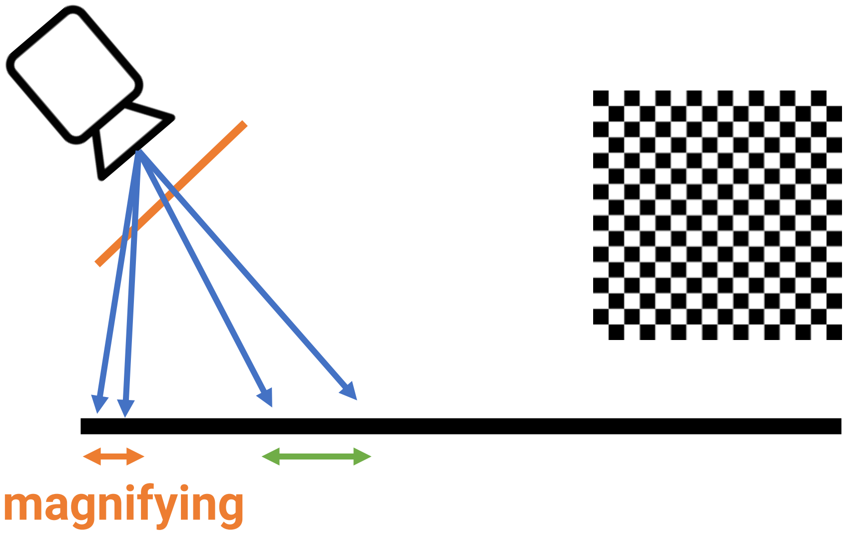
- Example
  - For the **green** area, one pixel covers a surface that is roughly one texel in the texture

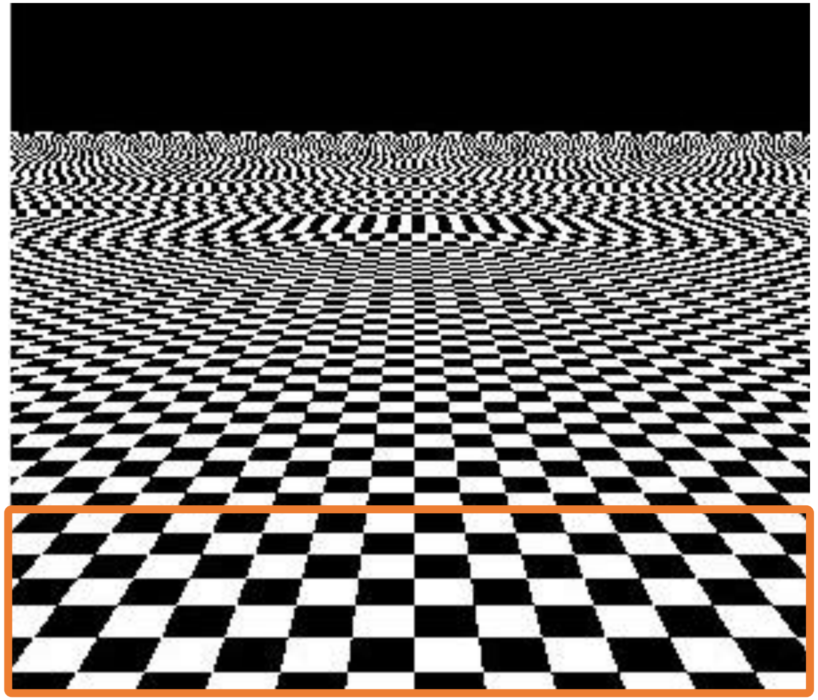




Texture Aliasing (cont.)

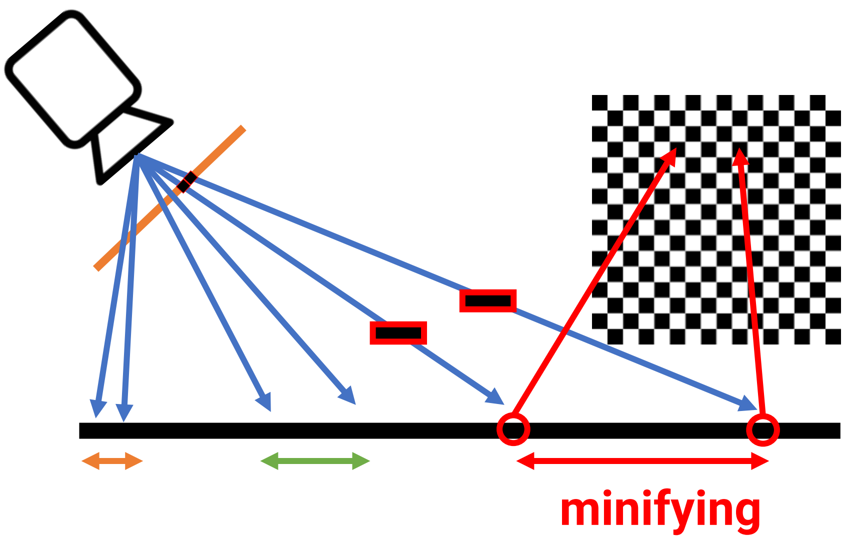
- Example
  - For the **orange** area, one pixel covers a surface that is **smaller** than one texel in the texture
  - Called **magnification**

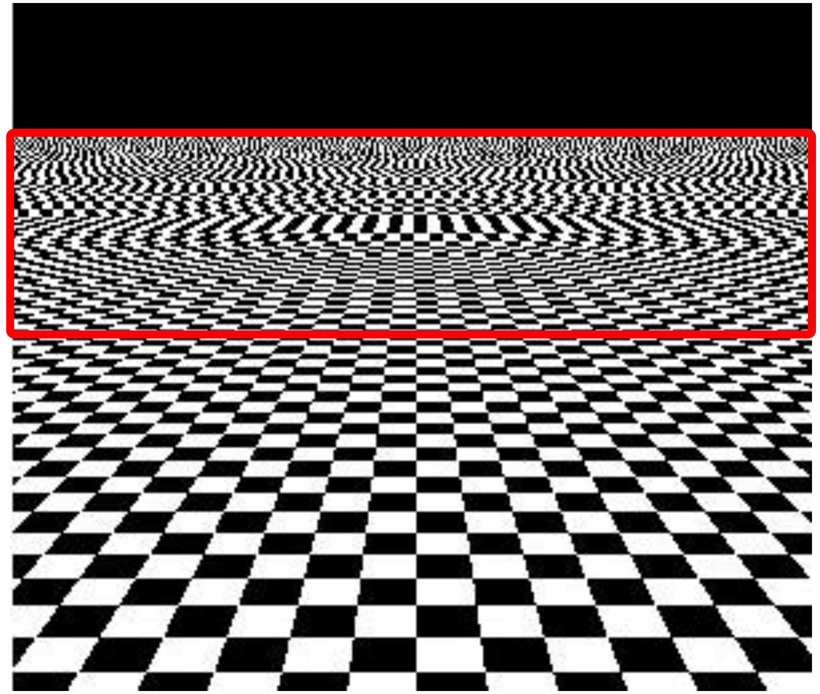




Texture Aliasing (cont.)

- Example
  - For the **red** area, one pixel covers a surface that is **larger** than one texel in the texture
  - Called **minification**



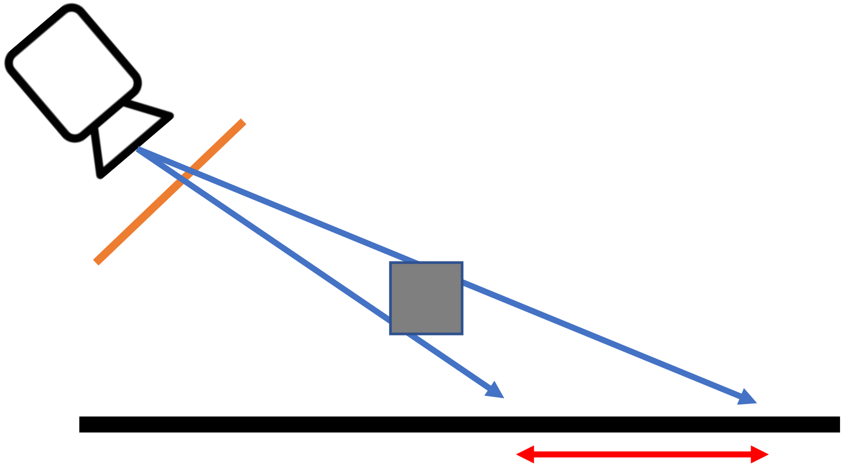


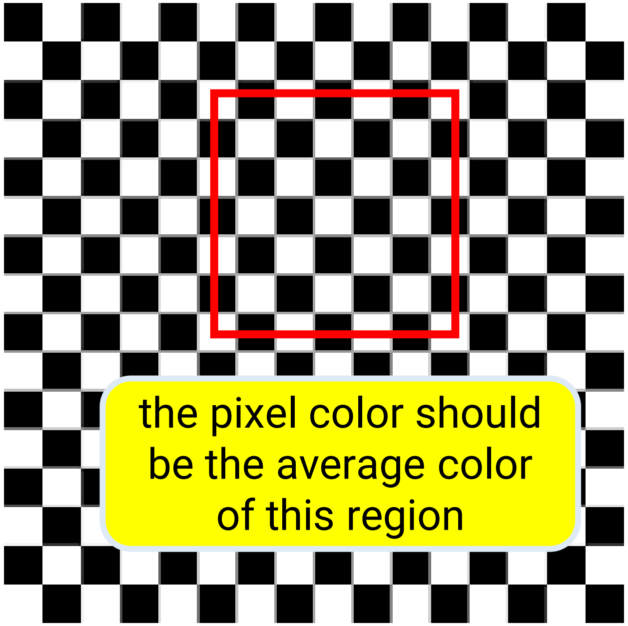
Texture Aliasing (cont.)

- Example
  - For the **red** area, one pixel covers a surface that is **larger** than one texel in the texture
  - Called **minification**
  - Might produce **flickering** for distant objects

Mipmap

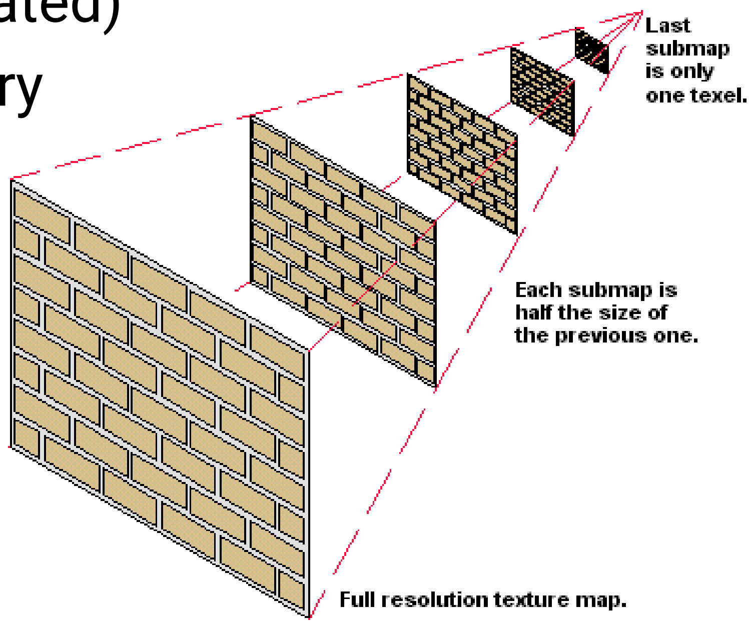
- To avoid aliasing, we should determine the regions a pixel covers (footprint) and average all the texture values inside the regions
- Time-consuming to do this in the run time!



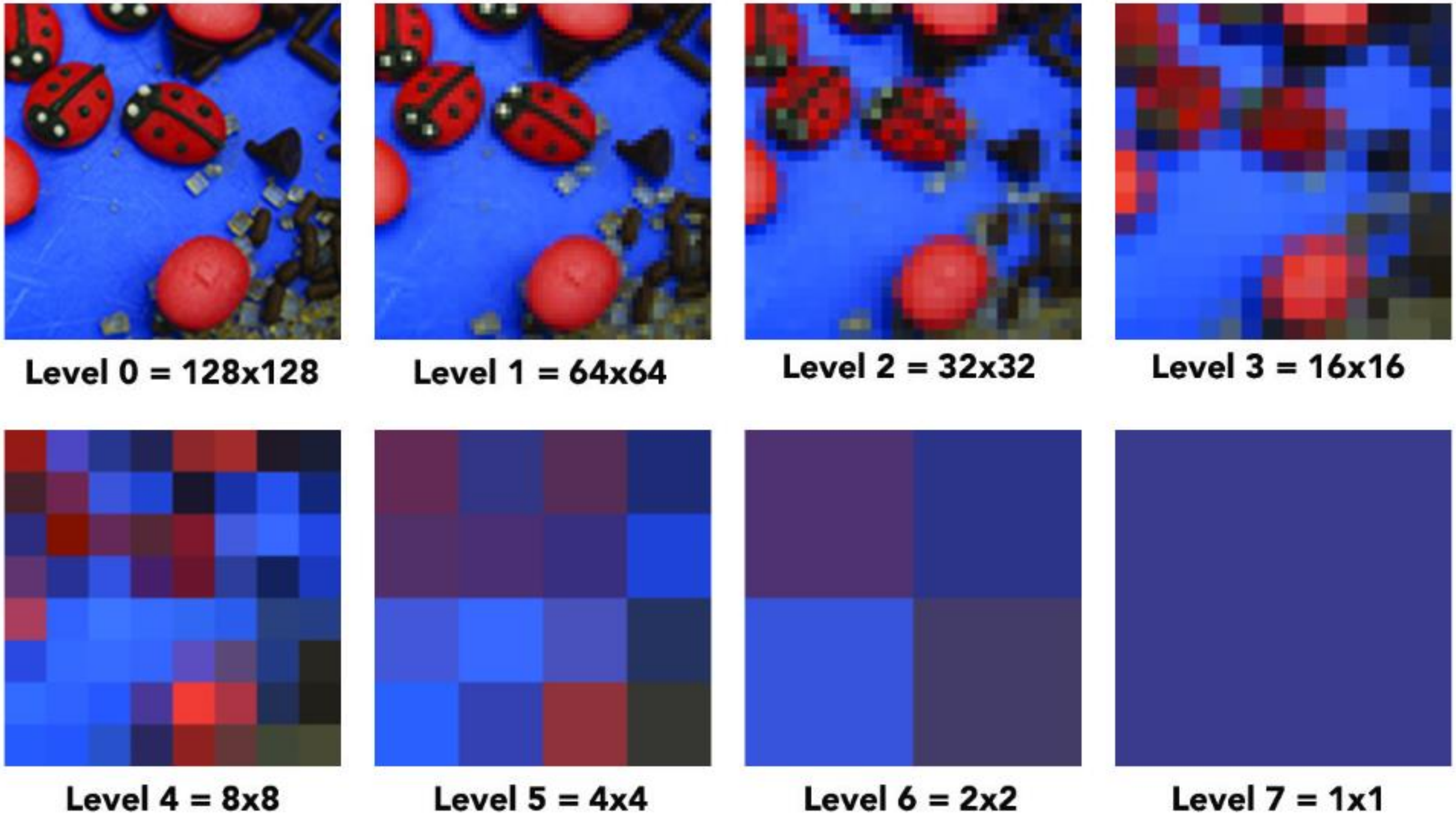


Mipmap (cont.)

- Mipmap provides a clever way to solve this problem
- **Pre-process**
  - Build a **hierarchical representation** of the texture image
  - Each level has a half resolution of its previous level (generated by linearly interpolated)
  - Take at most **1/3** more memory



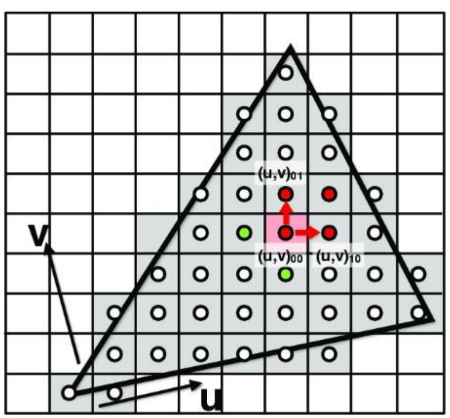
Mipmap (cont.)


Level 0 = 128x128
Level 1 = 64x64
Level 2 = 32x32
Level 3 = 16x16
Level 4 = 8x8
Level 5 = 4x4
Level 6 = 2x2
Level 7 = 1x1

Mipmap (cont.)

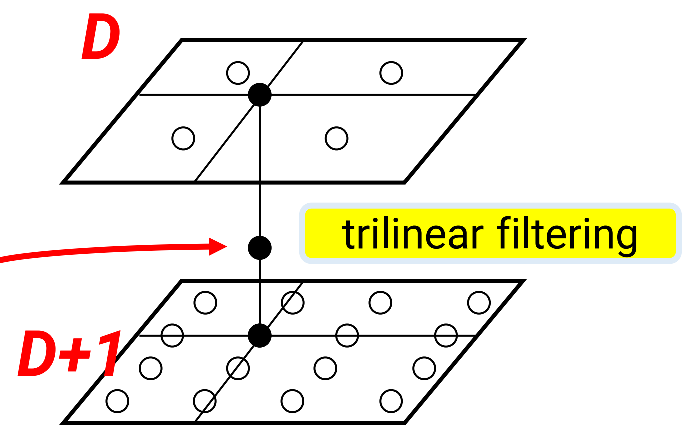
- **Run-time lookup**
  - Use **screen-space texture coordinate** to estimate its footprint in the texture space
  - Choose two levels ***D*** and ***D+1*** based on the footprint
  - Perform linear interpolation at level ***D*** to obtain a value V_D
  - Perform linear interpolation at level ***D+1*** to obtain V_{D+1}
  - Perform linear interpolation between V_D and V_{D+1}

trilinear

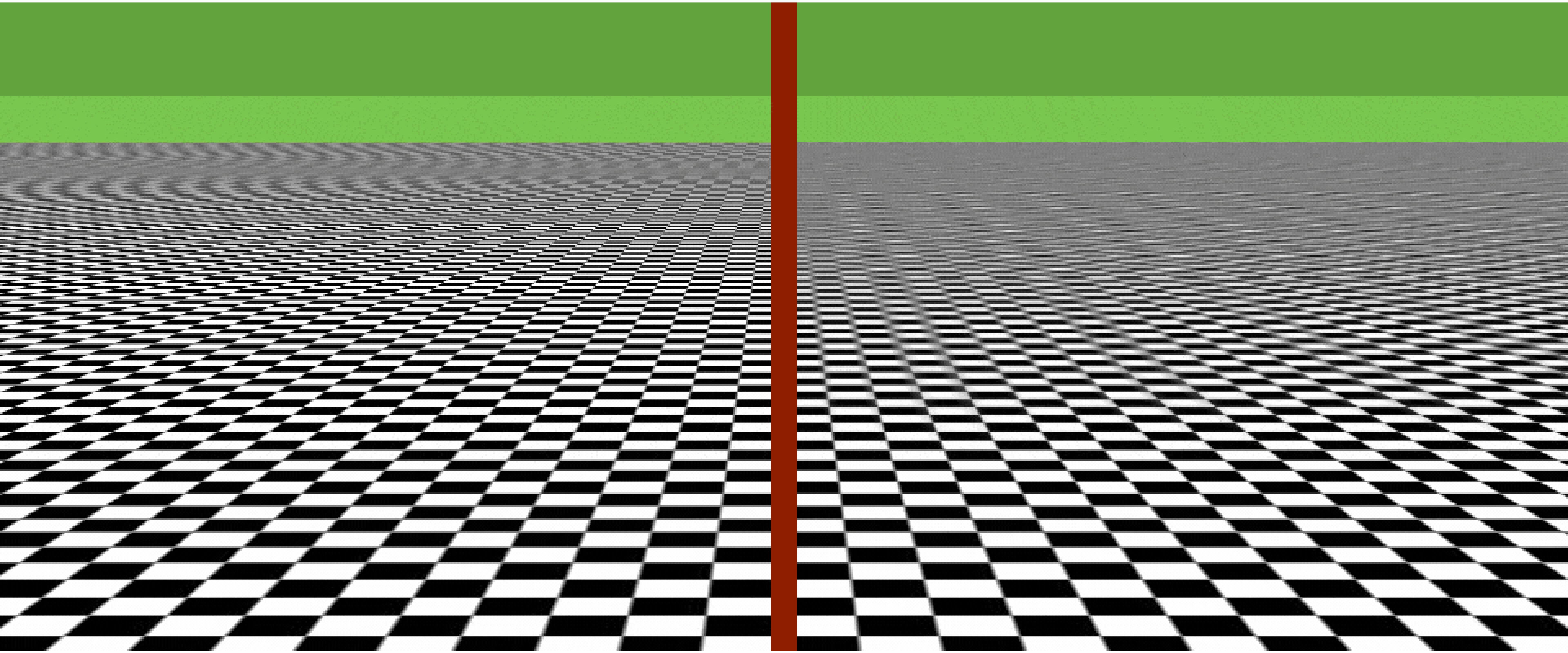


$$\frac{1}{W} = 2^{n-1-l}$$

$$l = n - 1 + \log w$$



Mipmap (cont.)



without mipmap

with mipmap

Mipmap (cont.)


tom's hardware guide
No Mipmapping
With Mipmapping
tom's hardware guide
Colored Mipmaps

Outline

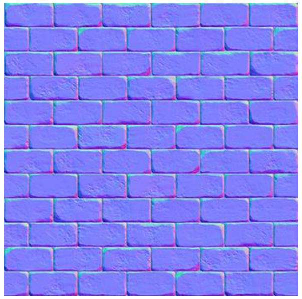
- Overview
- Texture data
- Texture filtering
- **Applications**
- OpenGL implementation

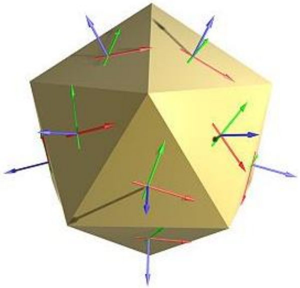
Normal Mapping

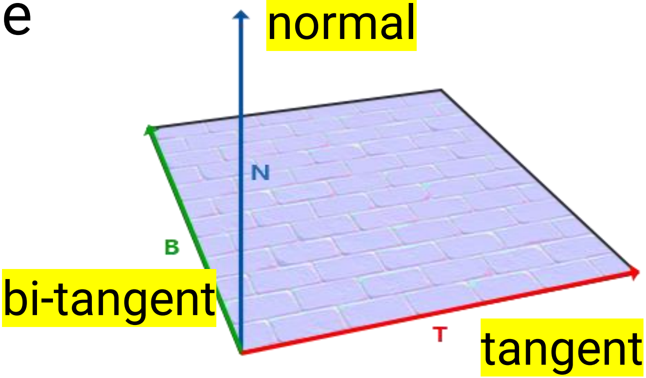
- Improve geometry details without adding vertices and triangles
  - Reduce the time of geometry processing
  - Only increase shading cost
  - Can also shorten the efforts of producing assets

Normal Mapping (cont.)

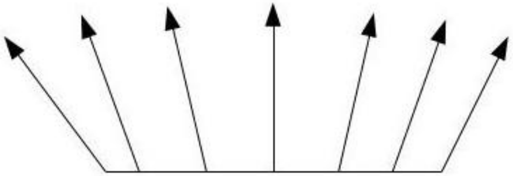
- Encode normal as texture color
  - (nx, ny, nz) = normalize(2 * TexColorRGB – 1)
  - The normal is defined in ***TBN*** space

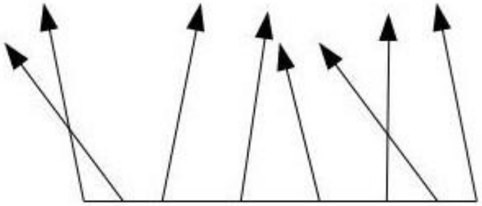






- During rendering, use shading normal instead of geometry normal





Normal Mapping (cont.)

- Recap: build camera matrix with viewing direction, right vector, and up vector

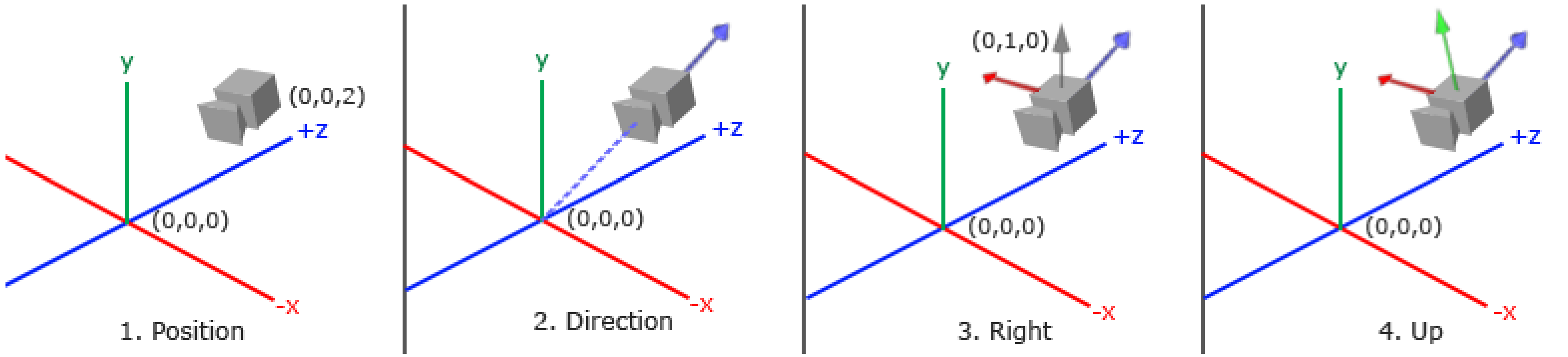
right vector

up vector

viewing vector

$$\begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation matrix translation matrix



Normal Mapping (cont.)

- Implementation
  - Calculate vertex tangent and bitangent as new vertex attributes
    - Calculate **per-face tangent** and **bi-tangent** and obtain **per-vertex tangent** and **bi-tangent** by averaging the face tangents of all adjacent faces
  - In the shader, build a ***TBN*** matrix and use it to transform the geometry normal

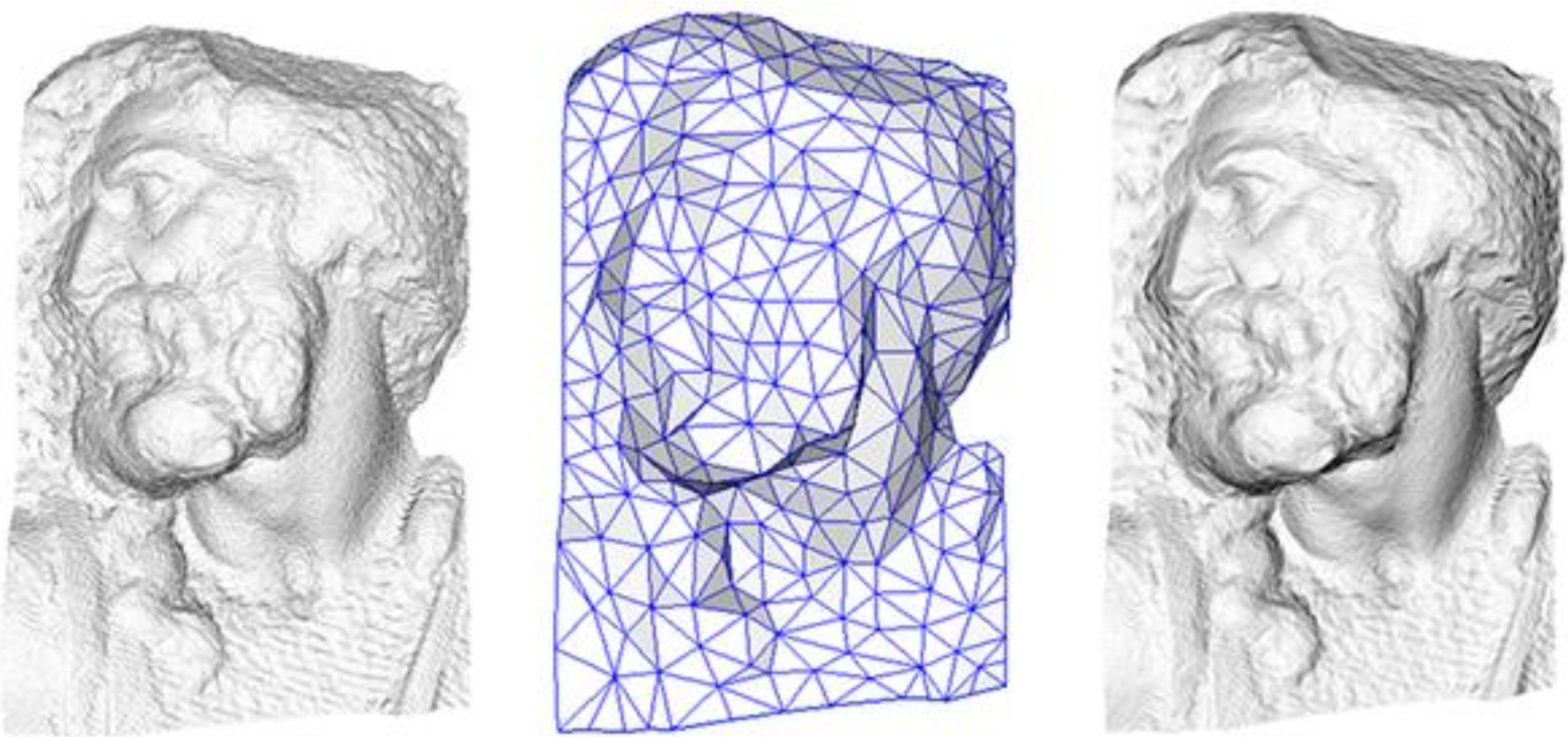
tangent vector

bi-tangent vector

normal vector

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

Normal Mapping (cont.)



original mesh
4M triangles

simplified mesh
500 triangles

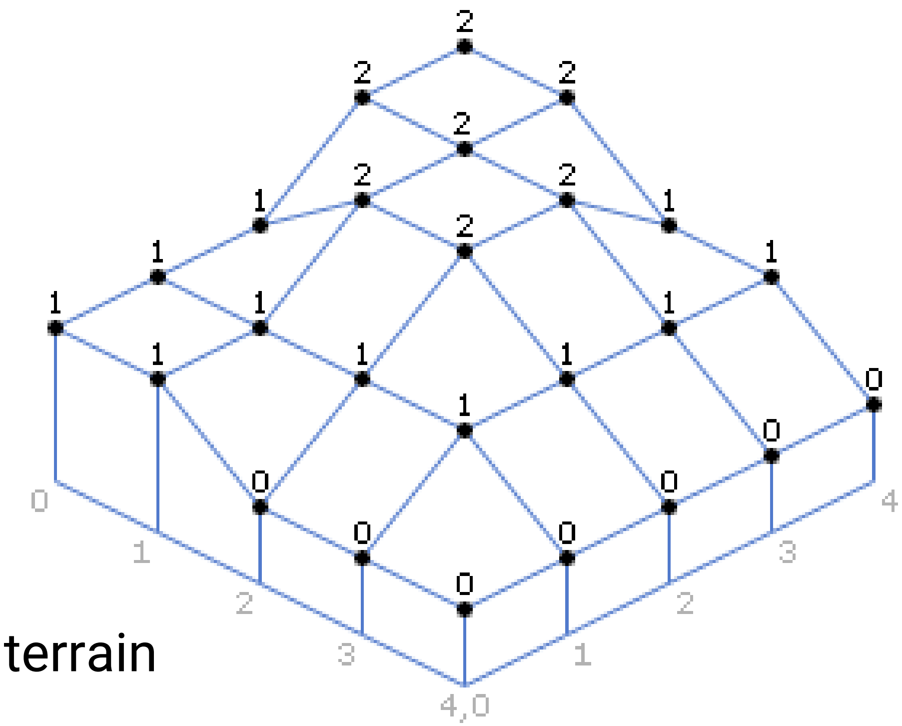
simplified mesh
and normal mapping
500 triangles

Height Map

- Use a scalar texture to represent the **vertex displacement** along the surface normal of a **base mesh**
- Widely used for **terrain** design

	0	1	2	3	4
0	1	1	1	2	2
1	1	1	2	2	2
2	0	1	2	2	1
3	0	1	1	1	1
4	0	0	0	0	0

heightmap

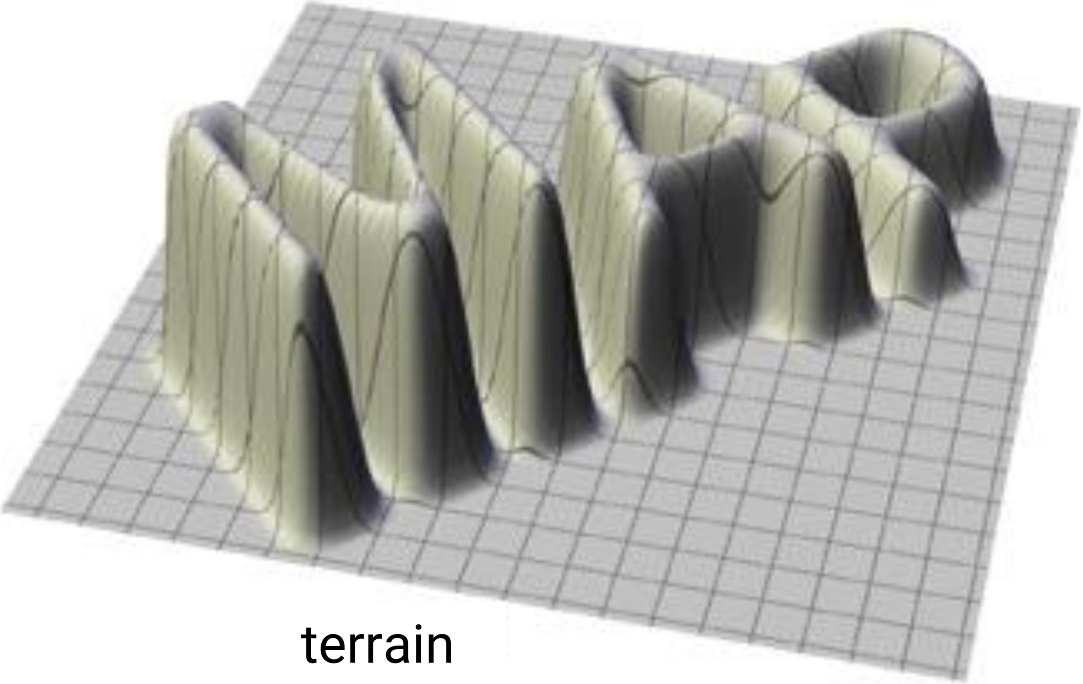


terrain

Height Map (cont.)

- Use a scalar texture to represent the **vertex displacement** along the surface normal of a **base mesh**
- Widely used for **terrain** design

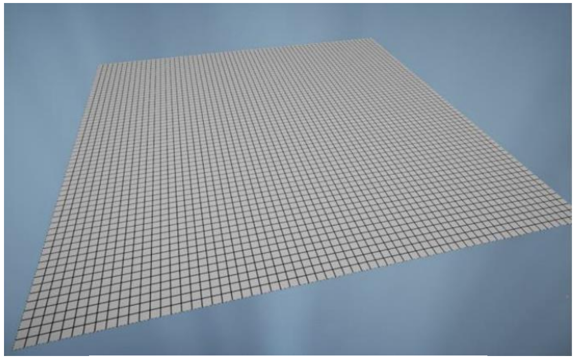
heightmap



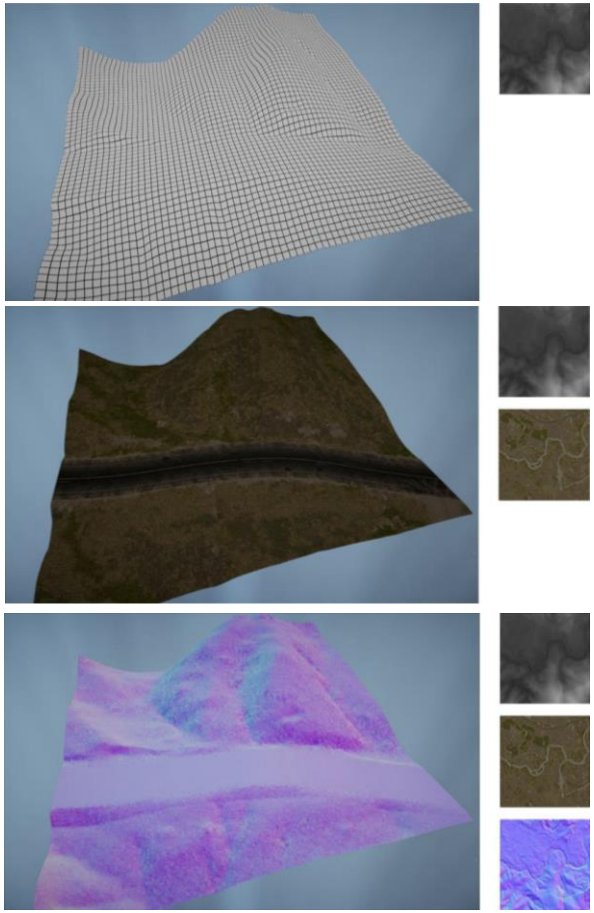
terrain

Height Map (cont.)

- Usually combined with an albedo texture and a normal map for shading



base mesh



rendered terrain

Height Map (cont.)

- Terrain management in ***FarCry 5***

Mountains and cliffs

High detail shading and geometry close to camera

Integration with other world elements (rocks, trees, grass)

Height Map (cont.)

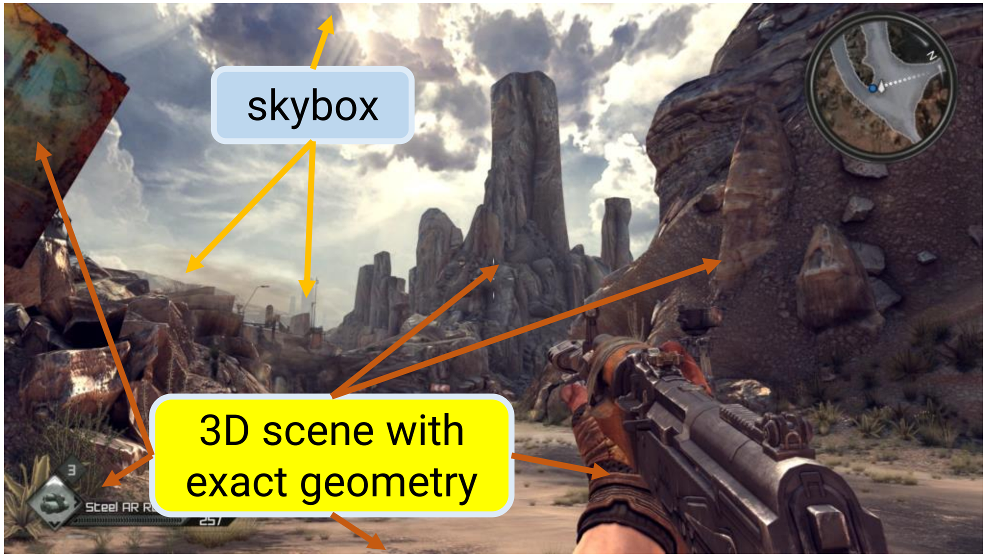
- Implementation
  - For each vertex in the base mesh, lookup the **height map** to displace the vertex (in the Vertex Shader)

    *new vertex position = original vertex position + normal * height*

  - For each fragment, lookup the **normal map** for the detailed shading normal and the **albedo texture** for the material property (in the Fragment Shader)

Skybox

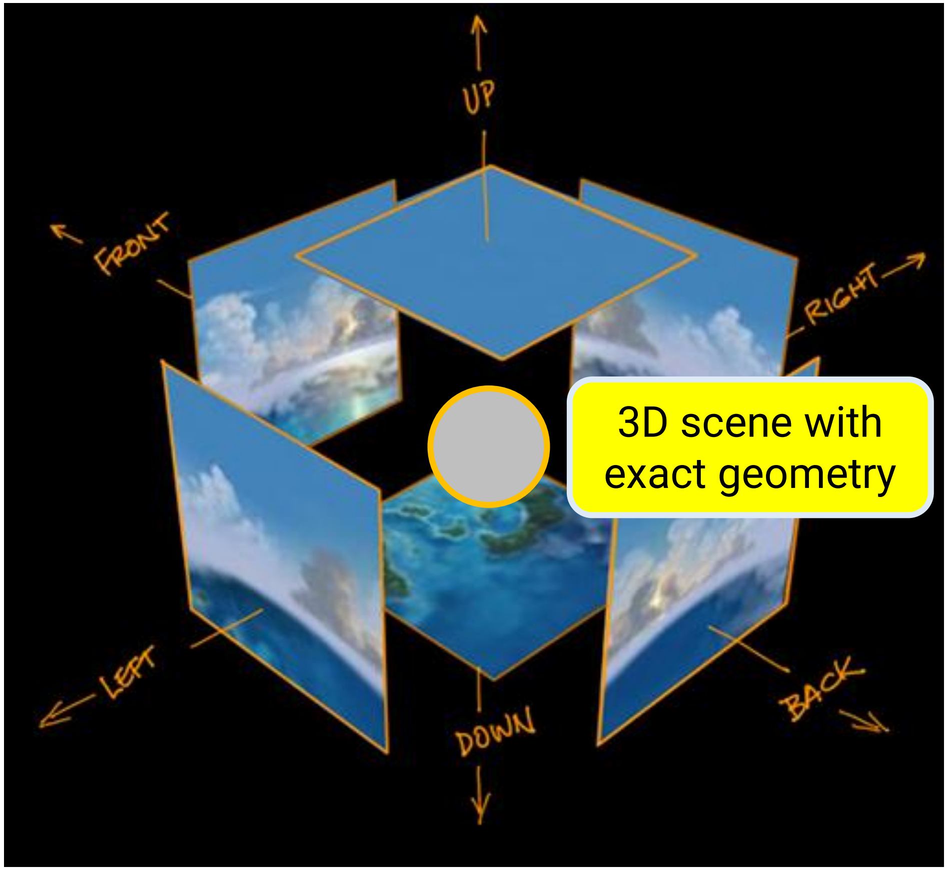
- Use a texture-mapped simple proxy geometry to represent far-away objects

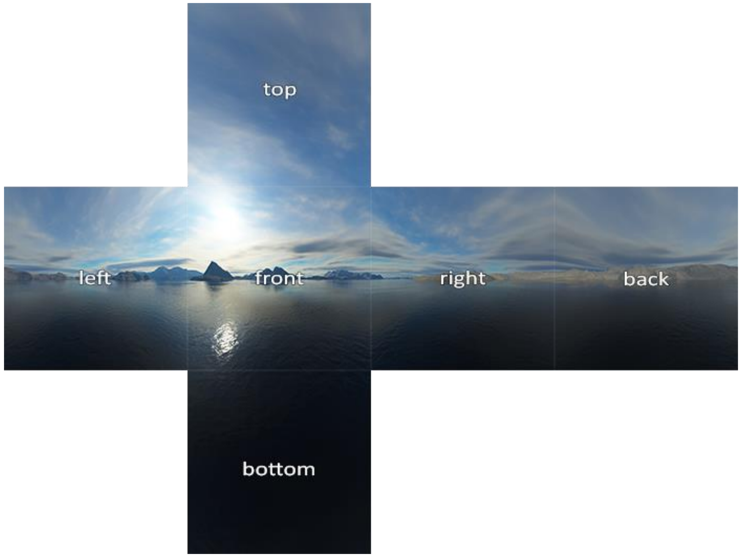


- Two approaches
  - Cube + **cube map** texture
  - Sphere + **longitude-latitude** image

Skybox (cont.)

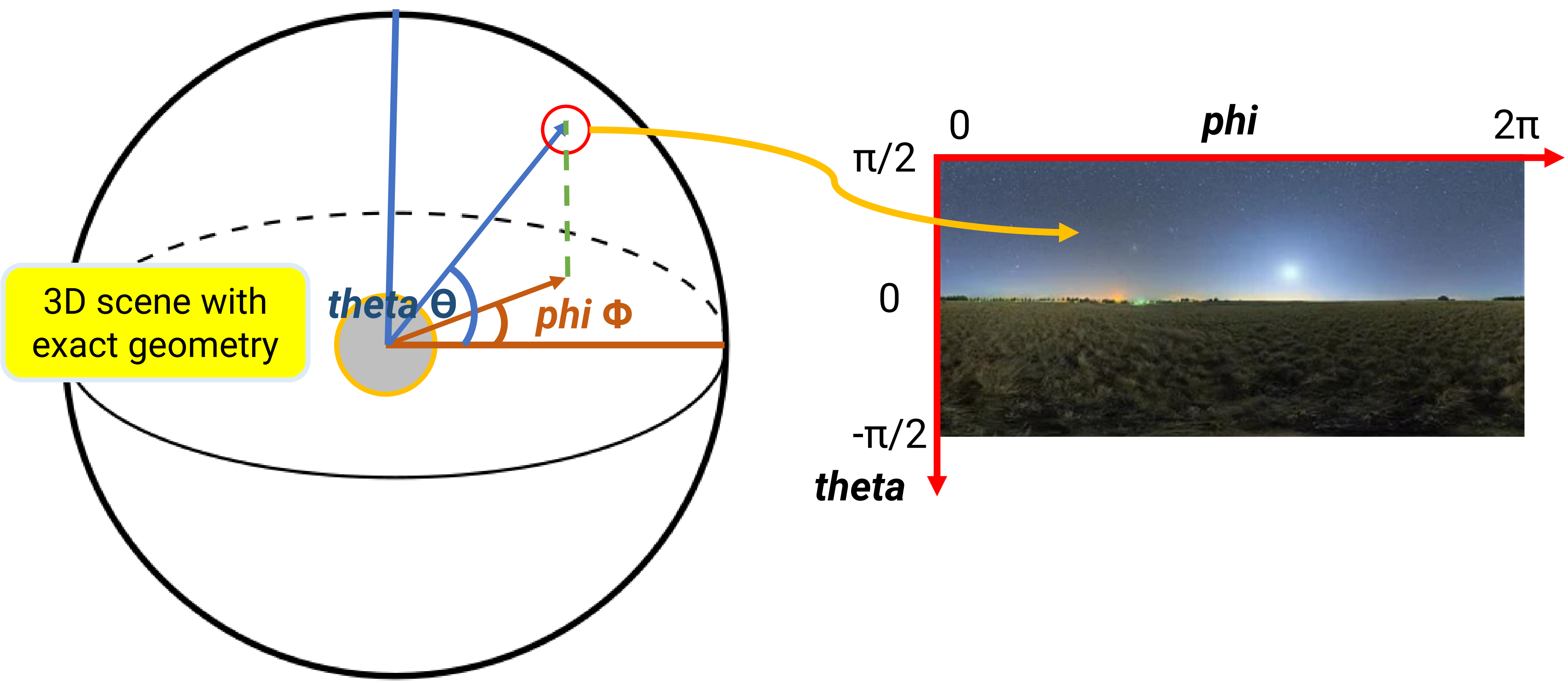
- Cube + **cube map** texture
  - Centered at world-space origin, with a significant long extent





Skybox (cont.)

- Sphere + **longitude-latitude** image
  - Centered at world-space origin, with a significant large radius



Reflection of the Skybox

- When rendering the scene, compute a reflected direction based on the viewing direction
- Use the reflected direction to lookup the skybox texture and obtain the reflected contribution
- Add the reflected contribution to the surface color

Reflection (cont.)

Ray Traced

Environment Map


END OF PRESENTATION
ANY QUESTIONS?