Computer Graphics

7. Buffers and Mapping techniques

I-Chen Lin

National Yang Ming Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

Intended Learning Outcomes

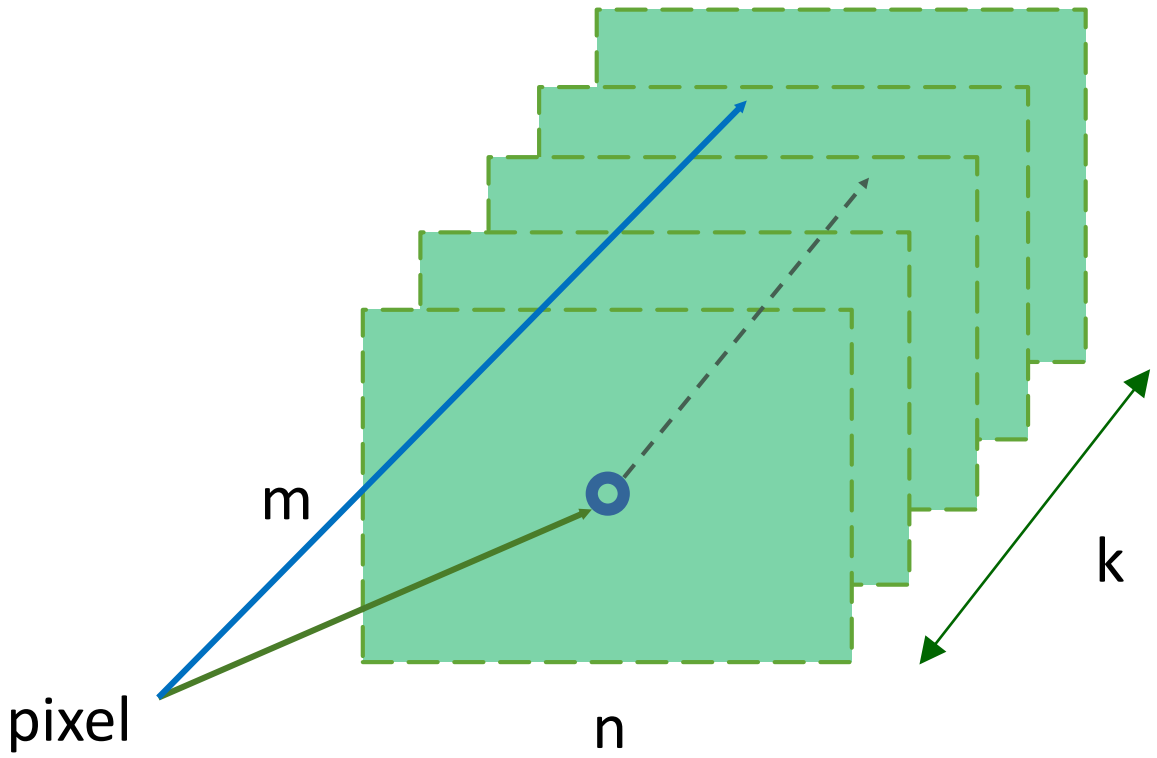
- On completion of this chapter, a student will be able to:
  - List the typical buffers used in graphics systems and their functions.
  - Express how texture images can be mapped onto polygons.
  - Explain the aliasing problem and describe anti-aliasing methods in graphics.
  - Present typical mapping methods and their applications.
  - Apply mapping methods with GLSL.

Outline

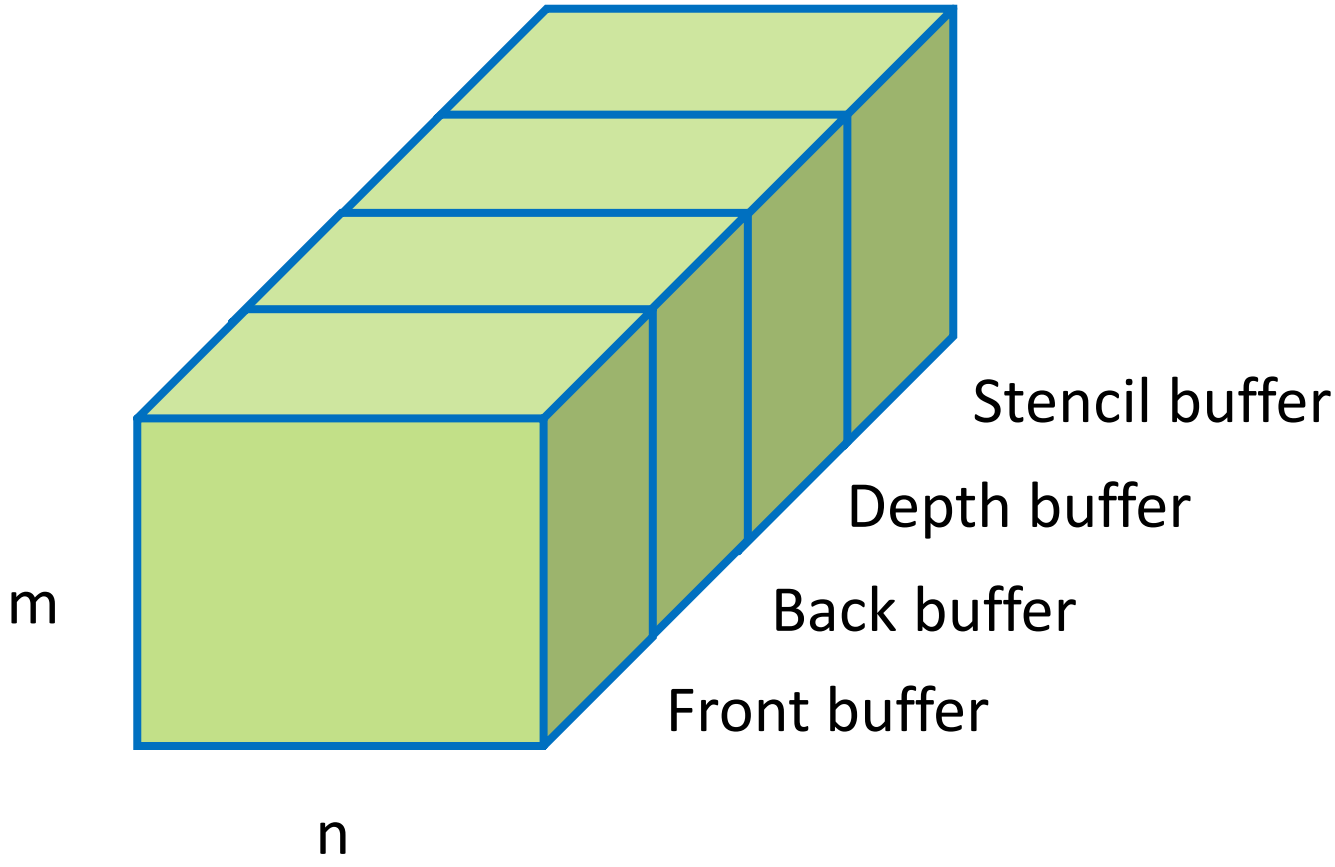
- Buffers
- Mapping techniques
- Anti-aliasing

Buffer

- Define a buffer by its spatial resolution (*n* x *m*) and its depth (or precision) *k*, the number of bits/pixel



OpenGL Frame Buffer



Buffers

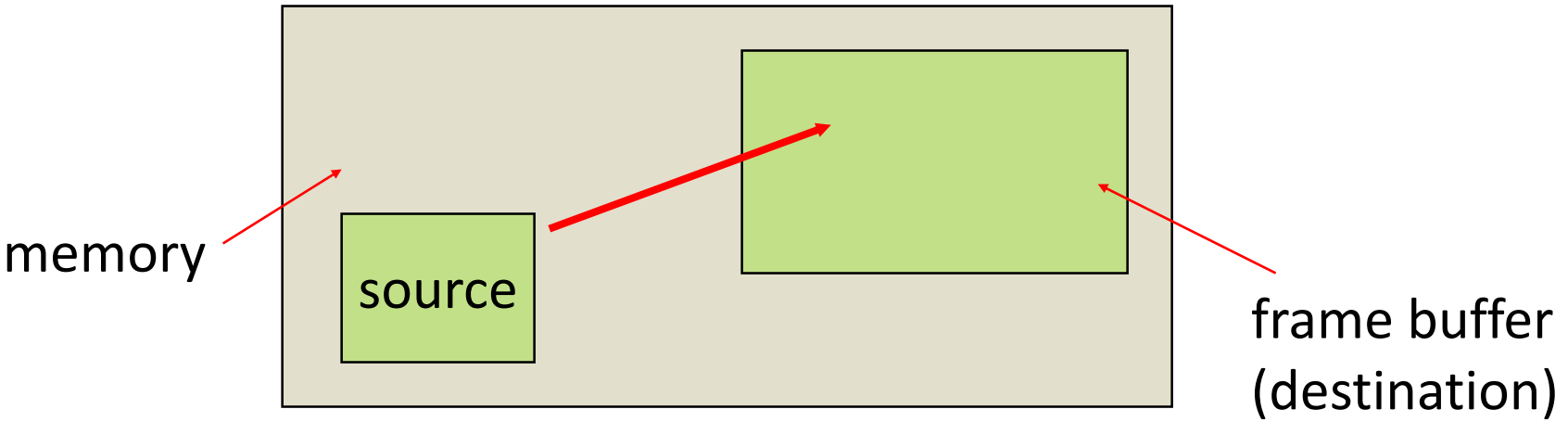
- Color buffers can be displayed
  - Front
  - Back
  - ……
- Depth
- Accumulation

Note: glAccum deprecated in newer OpenGL versions

- Stencil

Writing in Buffers

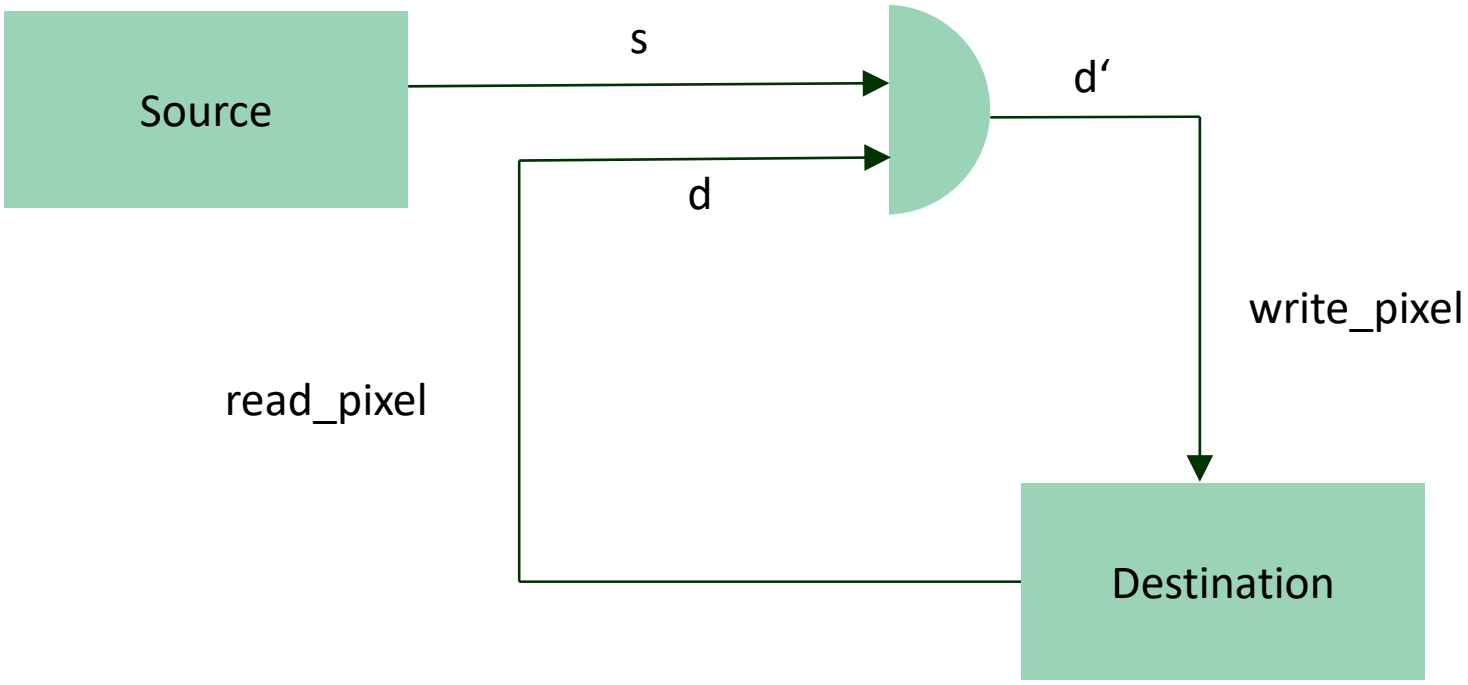
- Conceptually, we can consider all of memory as a large two-dimensional array of pixels
- We read and write rectangular block of pixels
  - Bit block transfer (bitblt) operations
- The frame buffer is part of this memory



writing into frame buffer

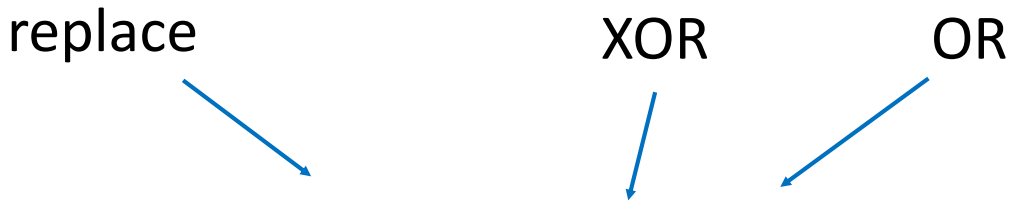
Writing Model

- Read destination pixel before writing source



Bit Writing Modes

- Source and destination bits are combined bitwise
- 16 possible functions (one per column in table)



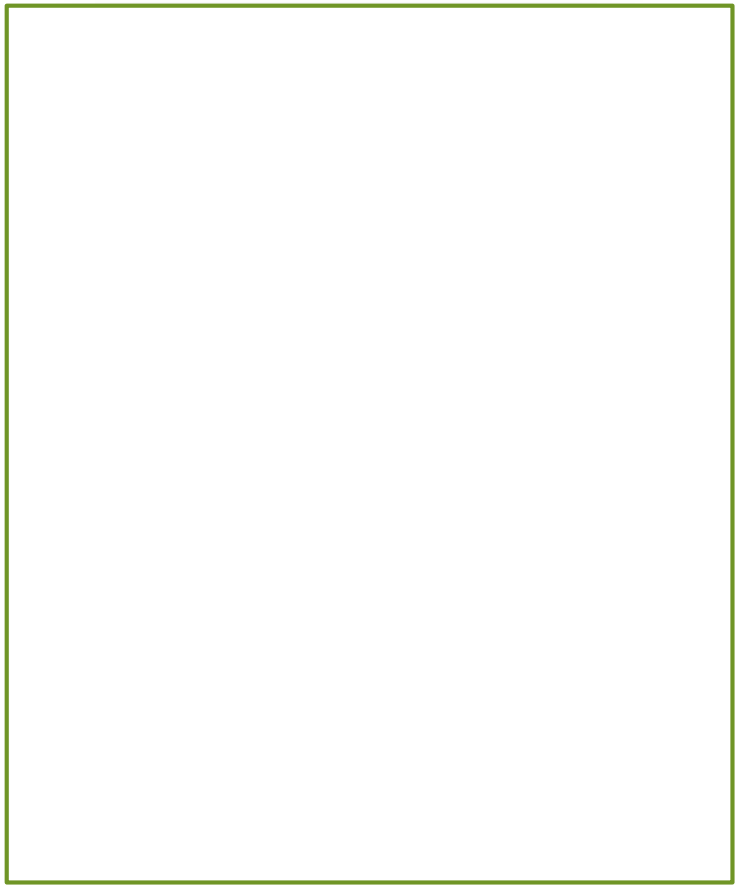
s	d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Mapping Methods

- Texture Mapping
- Environment Mapping
- Normal and Bump Mapping

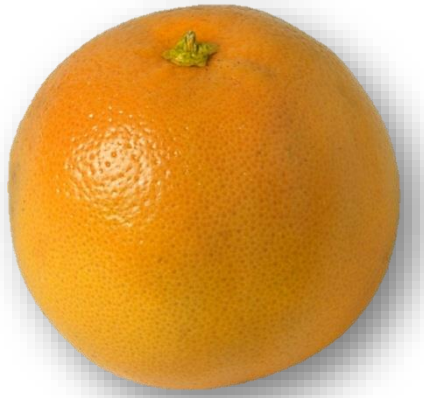
The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second, the number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin



Modeling an Orange

- Consider the problem of modeling an orange (the fruit)
- Start with an orange-colored sphere
  - Too simple



- Replace sphere with a more complex shape
  - Does not capture surface characteristics (small dimples)
  - Takes too many polygons to model all the dimples

Modeling an Orange (cont.)

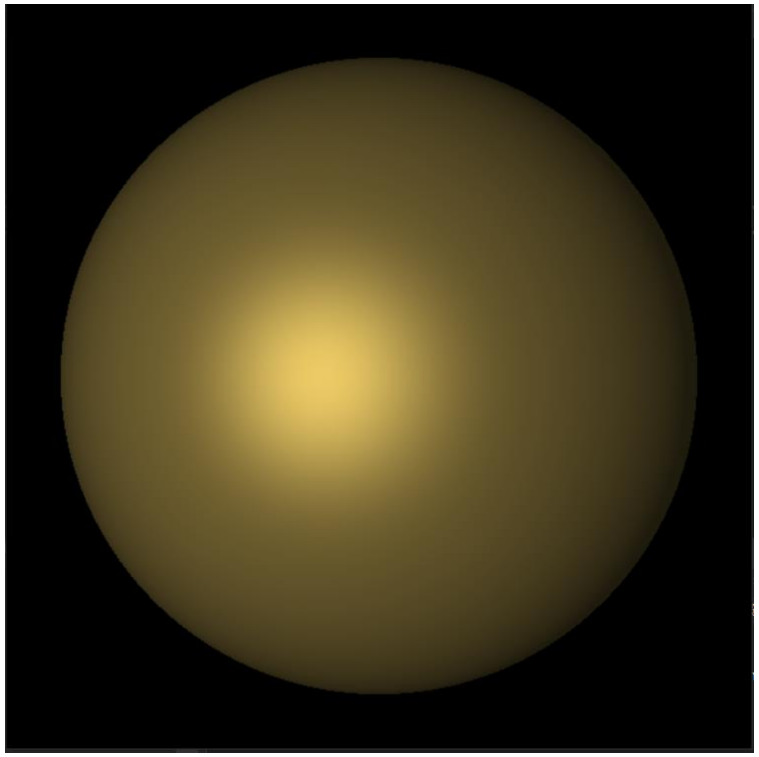
- Take a picture of a real orange, scan it, and “paste” onto simple geometric model
  - This process is known as *texture mapping*

- Still might not be sufficient because the resulting surface will be smooth
  - Need to change local shape
  - Bump mapping

Three Typical Types of Mapping

- Texture Mapping
  - Uses images to fill inside of polygons
- Environment (reflection mapping)
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
- Bump mapping
  - Emulates altering normal vectors during the rendering process
- ……

Texture Mapping

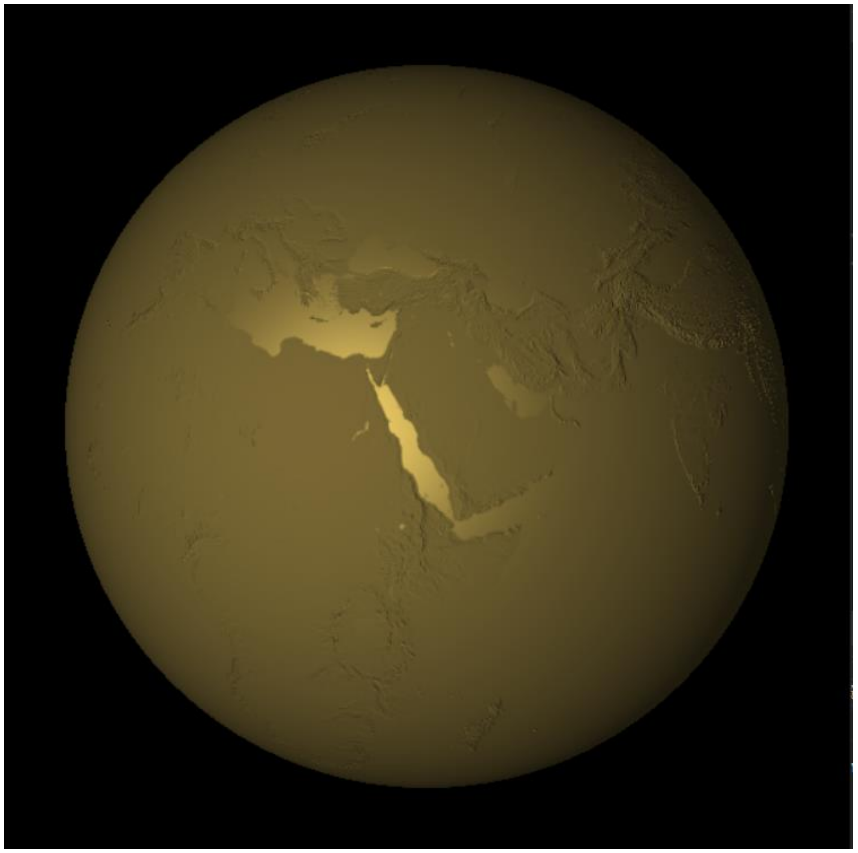


Geometric model

Texture-mapped

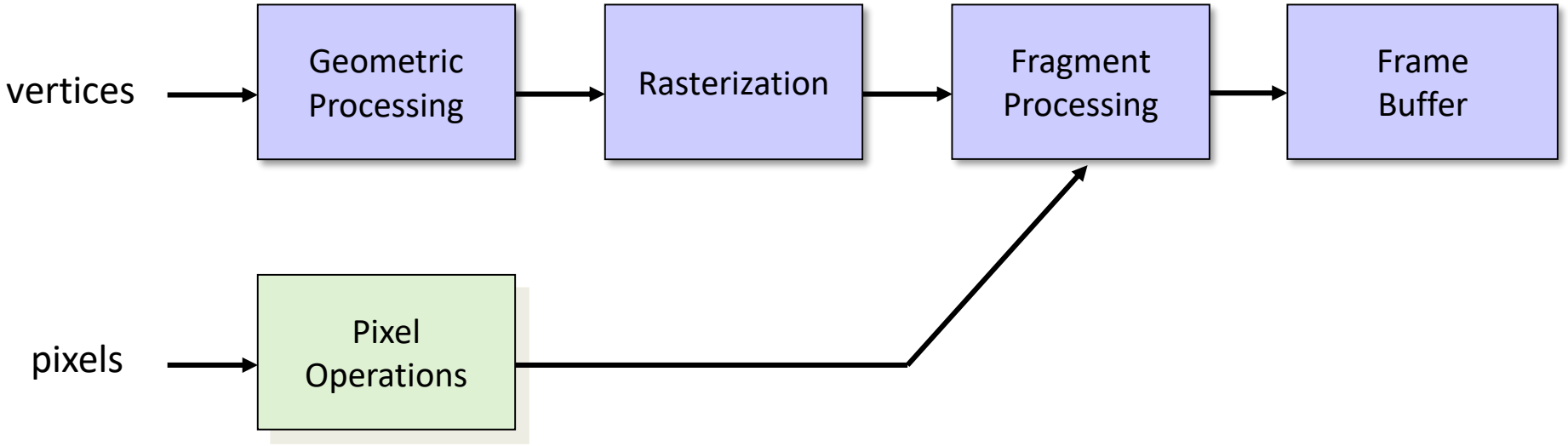
Environment Mapping

Bump Mapping



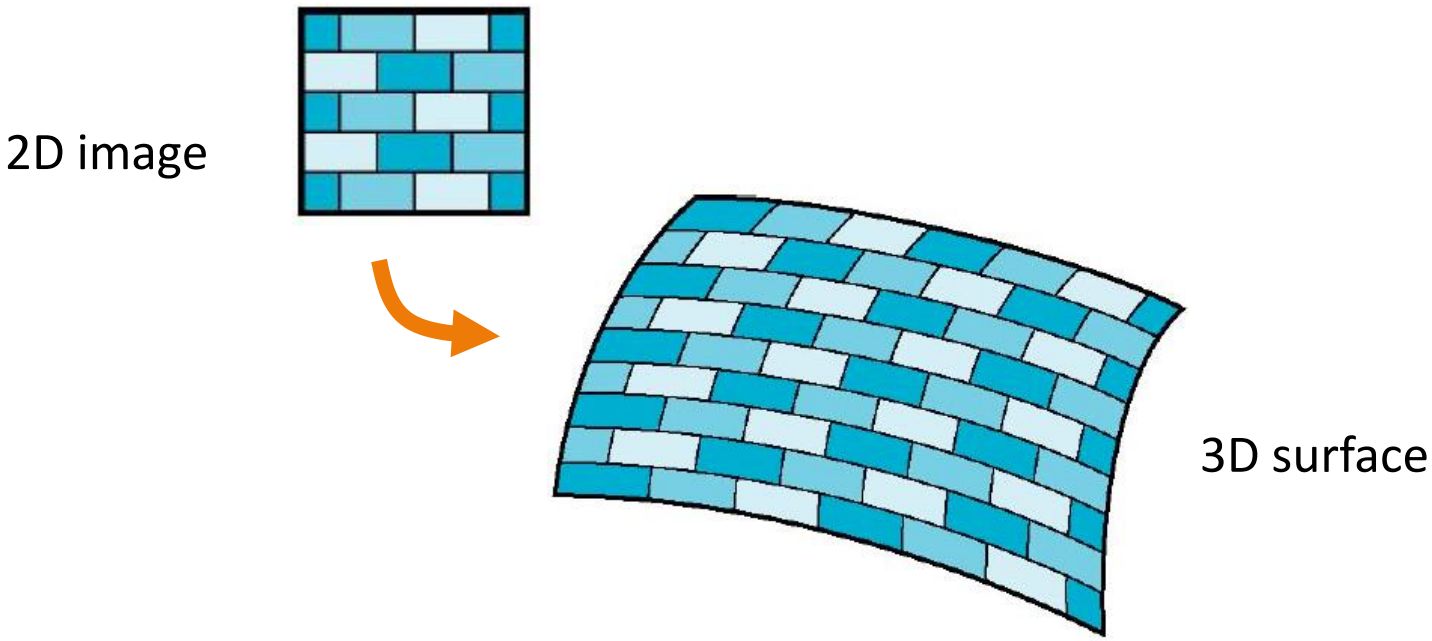
Where Does Mapping Take Place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons make it past the clipper



Is it Simple?

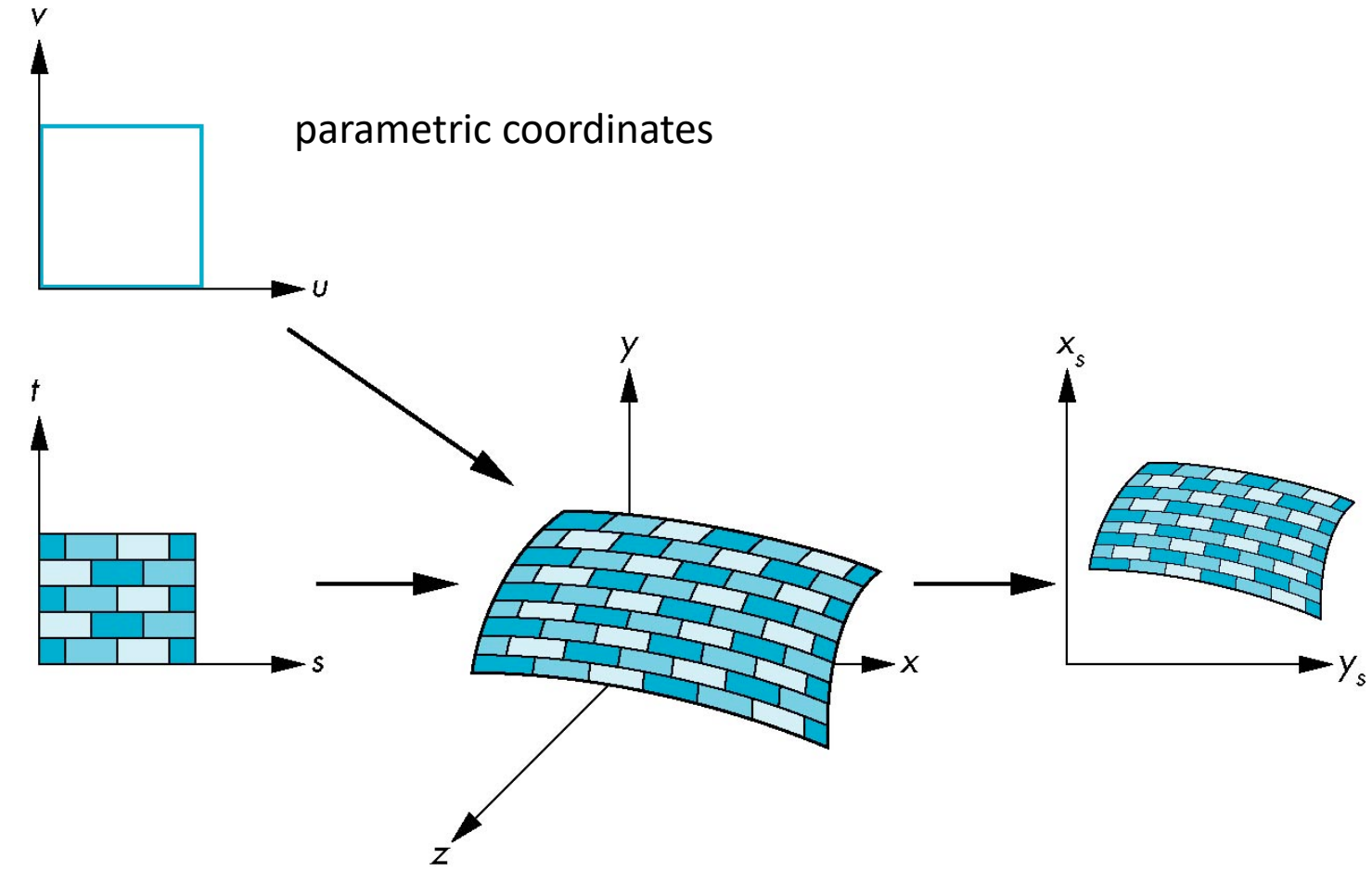
- Although the idea is simple
  - map an image to a surface---there are 3 or 4 coordinate systems involved



Coordinate Systems

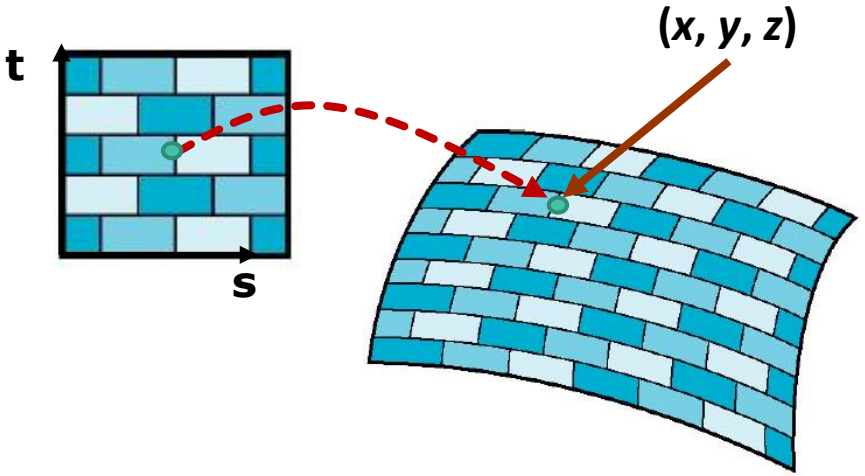
- Parametric coordinates
  - May be used to model curves and surfaces
- Texture coordinates
  - Used to identify points in the image to be mapped
- Object or World Coordinates
  - Conceptually, where the mapping takes place
- Window Coordinates
  - Where the final image is really produced

Texture Mapping



Mapping Functions

- The basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions
  - x = x(s,t)
  - y = y(s,t)
  - z = z(s,t)



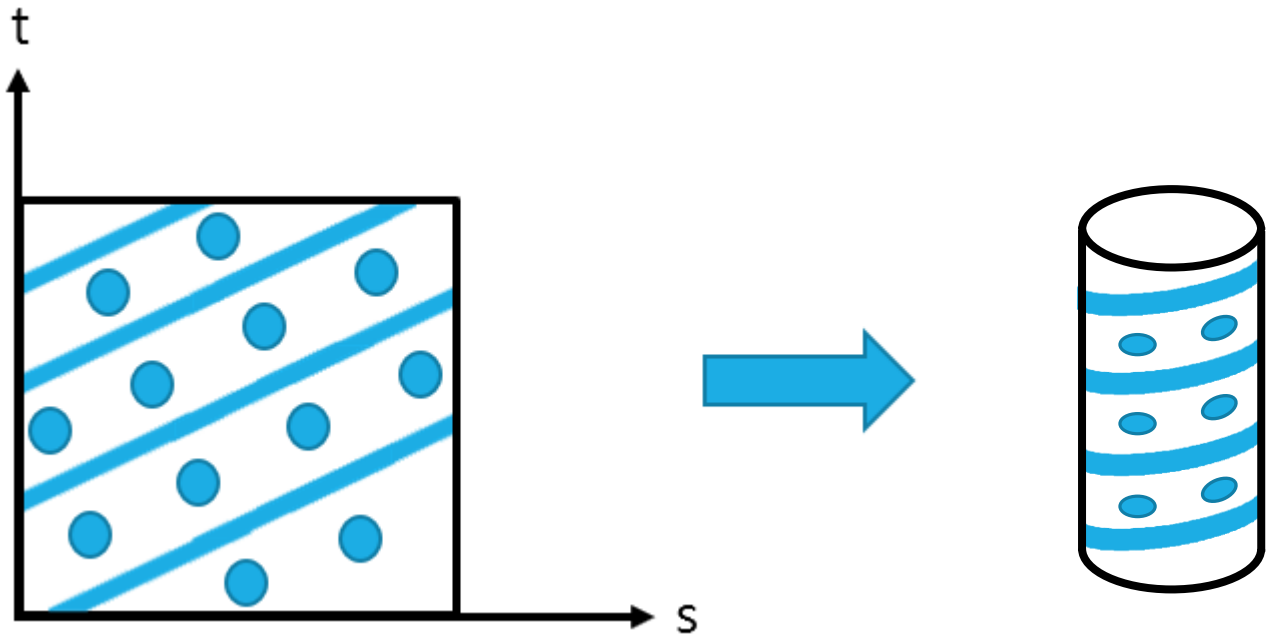
- But we really want to go the other way

Backward Mapping

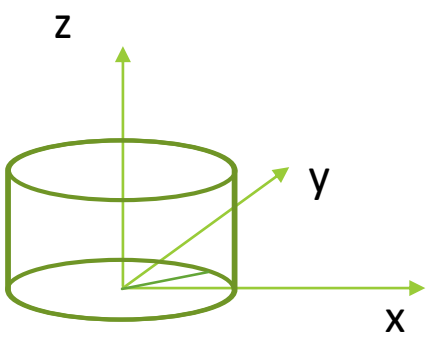
- We really want
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a backward map of the form
  - $s = \mathbf{s}(x, y, z)$
  - $t = \mathbf{t}(x, y, z)$
- Such functions are difficult to find in general

Two-part Mapping

- One solution to the mapping problem is to first map the texture to a simple intermediate surface
  - Example: map to cylinder



Cylindrical Mapping



- parametric cylinder
  - x = r cos 2πu
  - y = r sin 2πu
  - z = vh

u, v: 0~1

- maps rectangle in *u*,*v* space to cylinder of radius r and height h in world coordinates
  - s = u
  - t = v

- maps from texture space

Spherical Map

- We can use a parametric sphere
  - x = r sin 2πu cos 2πv
  - y = r sin 2πu sin 2πv
  - z = r cos 2πu
- in a similar manner to the cylinder but have to decide where to put the distortion
- *Spheres are used in environmental maps*

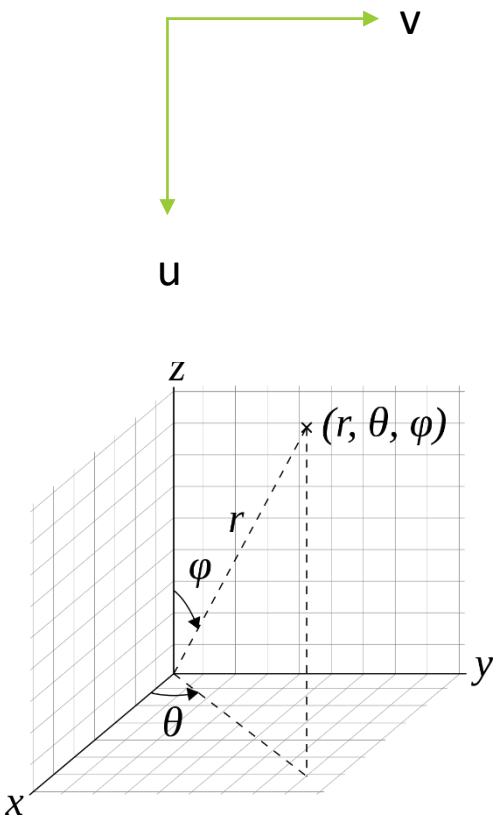
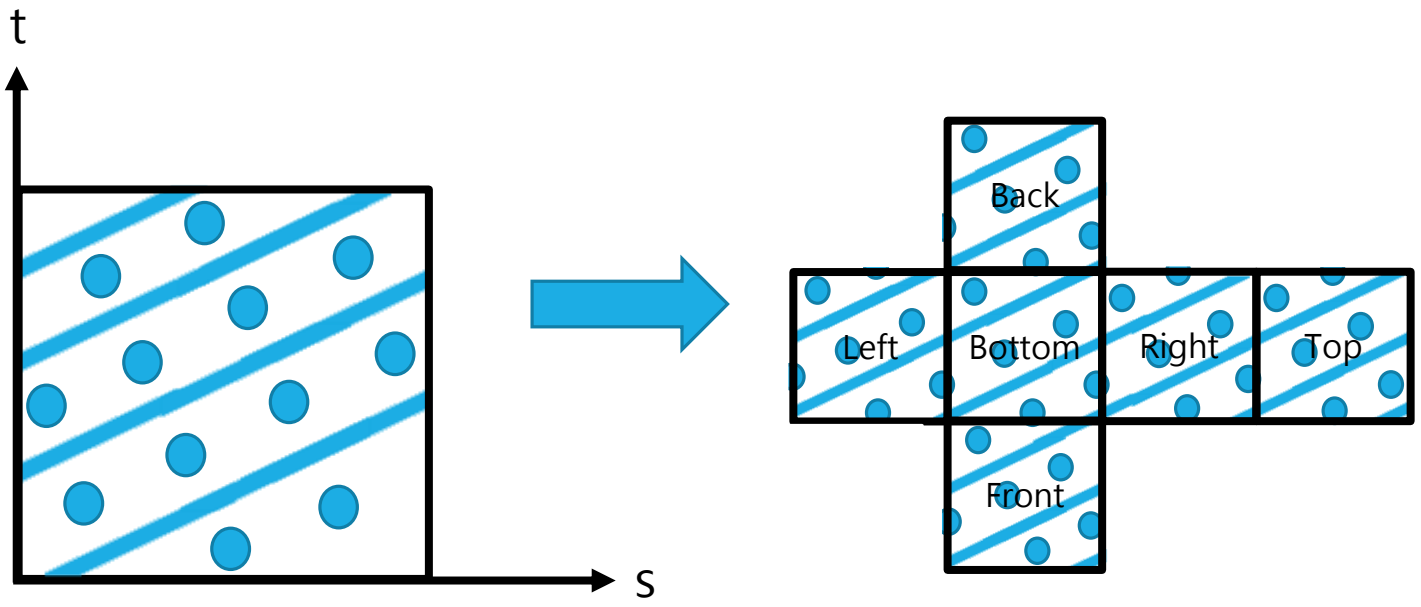


Fig. from en.wikipedia.org/wiki/Spherical_coordinate_system

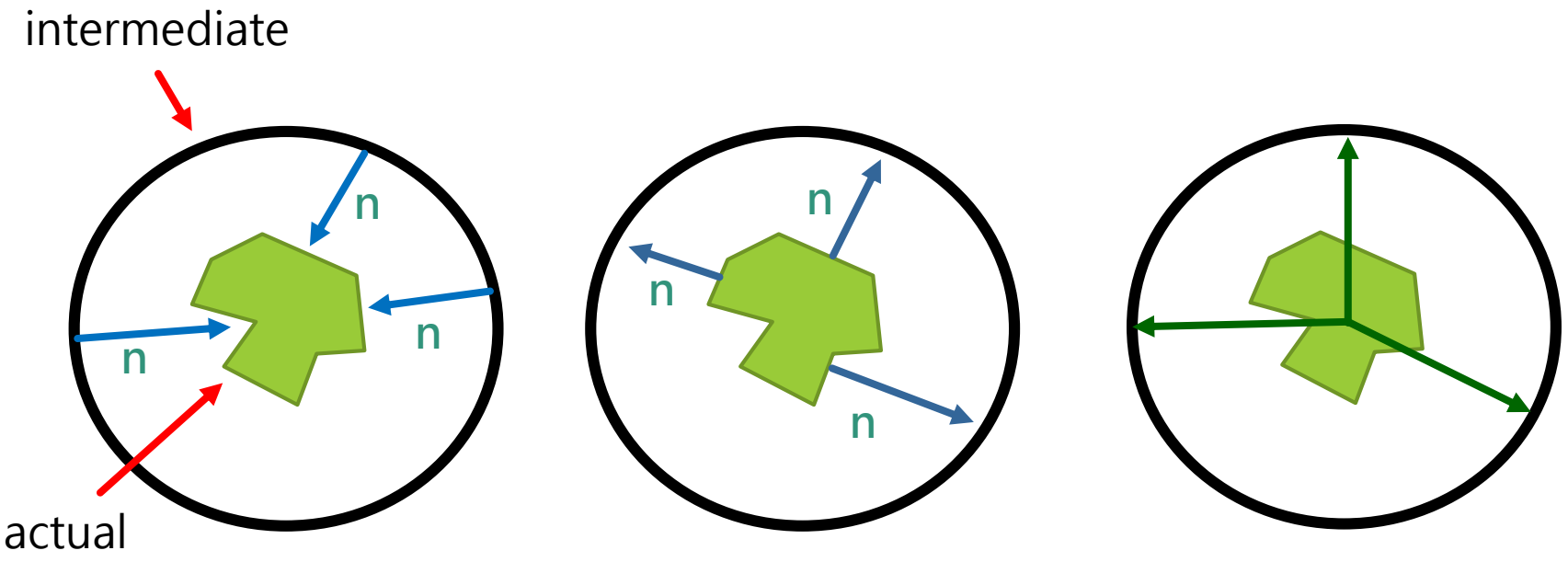
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps (Cube mapping)

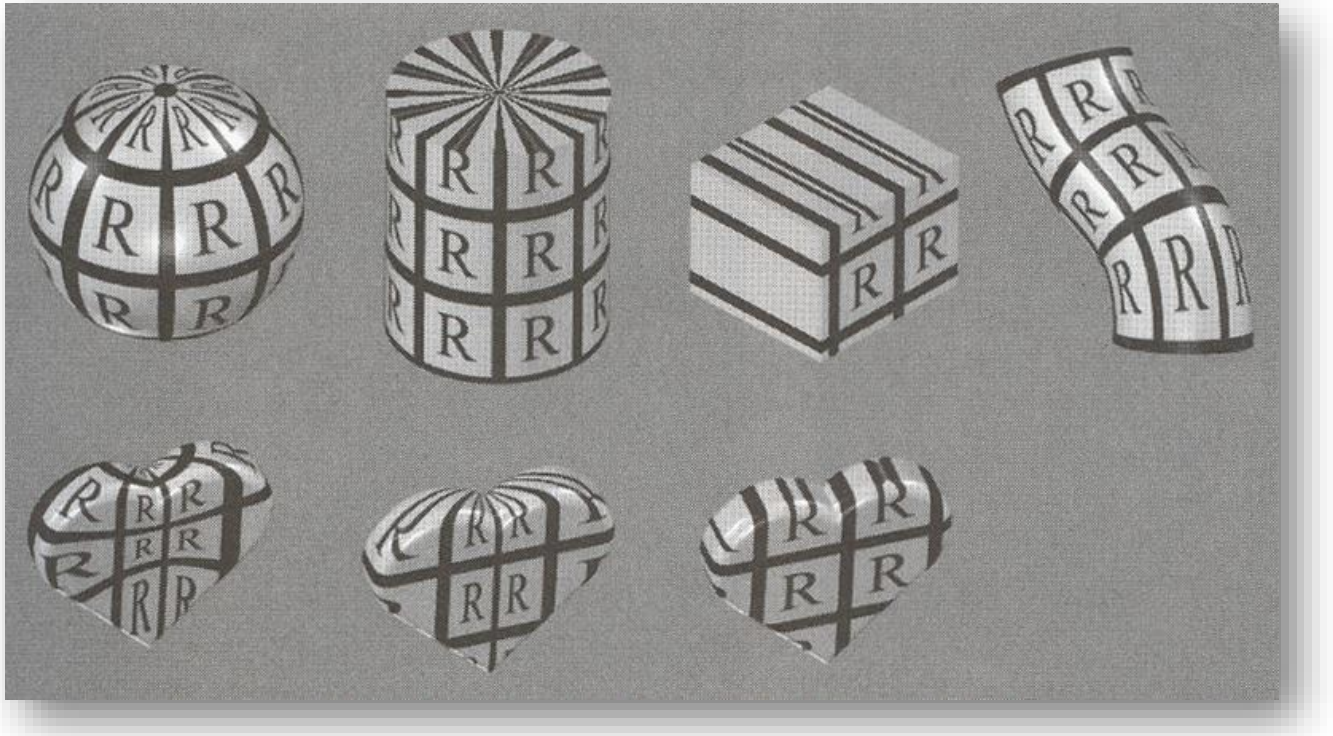


Second Mapping

- Map from an intermediate object to an actual object
  - Normals from the intermediate to the actual
  - Normals from the actual to the intermediate
  - Vectors from the center of the intermediate



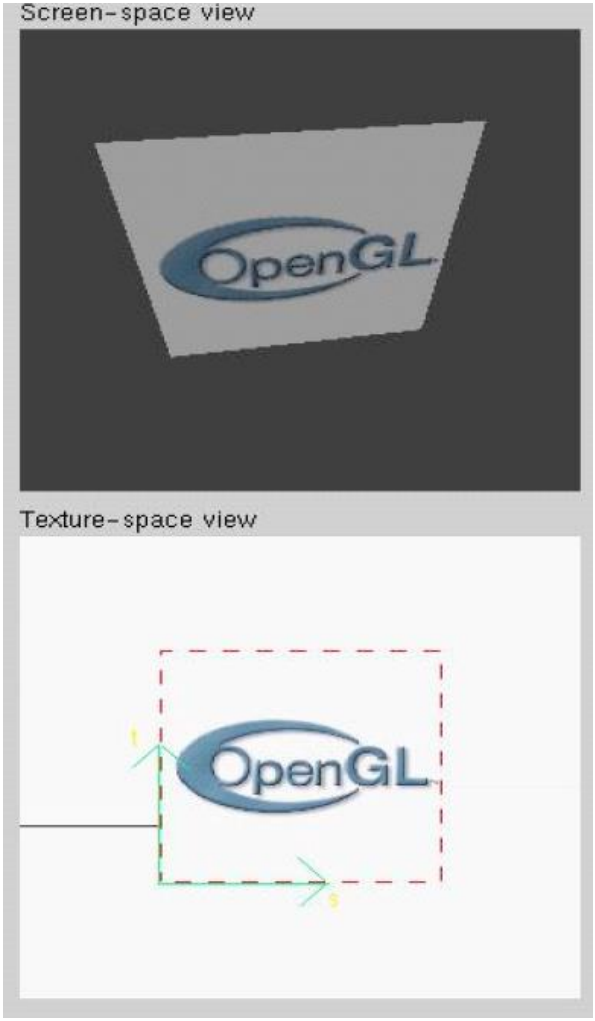
Two-part Mapping



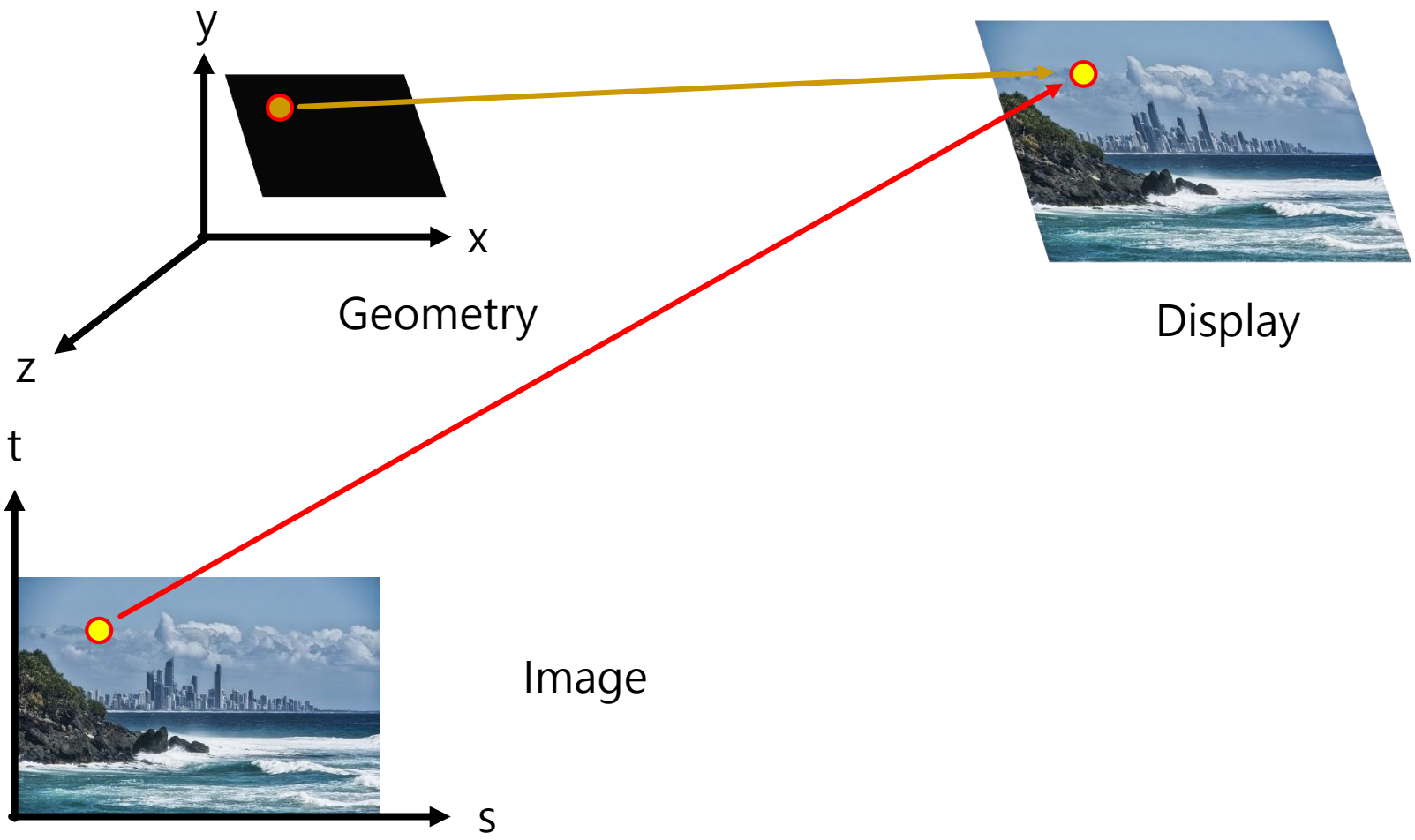
Texture Example

- The texture (below) is a 256 x 256 image, mapped to a rectangular polygon which is viewed in perspective.

- OpenGL requires texture dimensions to be powers of 2

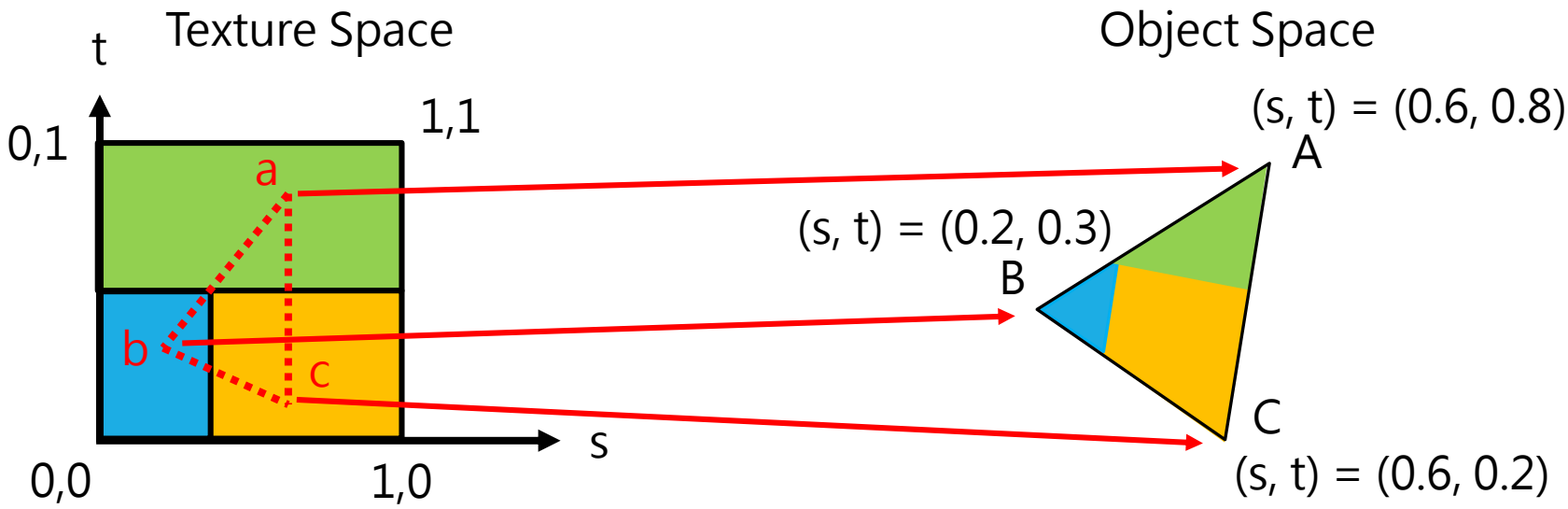


Texture Mapping



Texture Mapping for Polygons

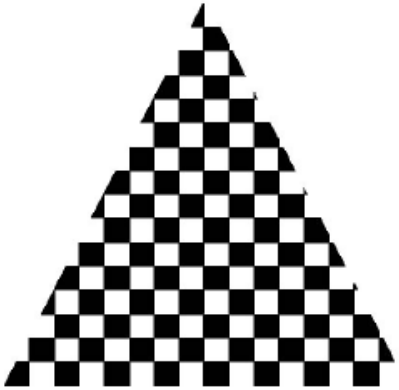
- Based on parametric texture coordinates
  - glTexCoord*() specified at each vertex



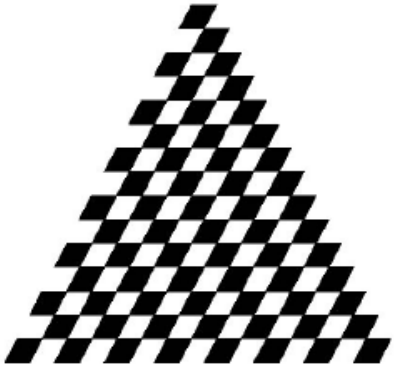
Interpolation

- OpenGL uses interpolation to find proper texels from specified texture coordinates
  - Can be distortions

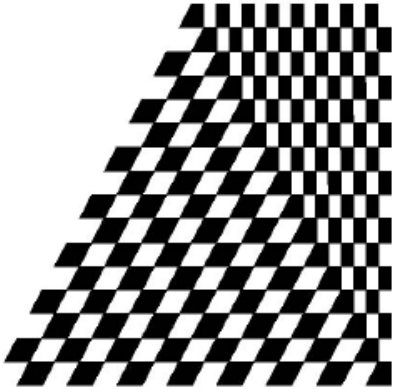
good selection
of tex coordinates



poor selection
of tex coordinates

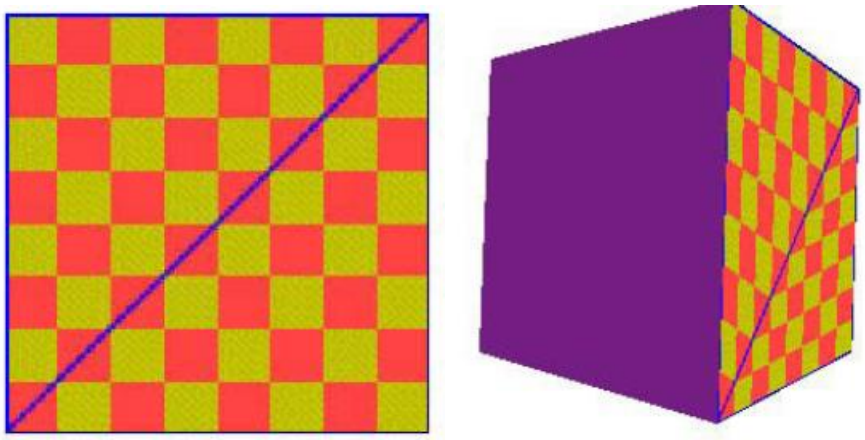


texture stretched
over trapezoid
showing effects of
bilinear interpolation



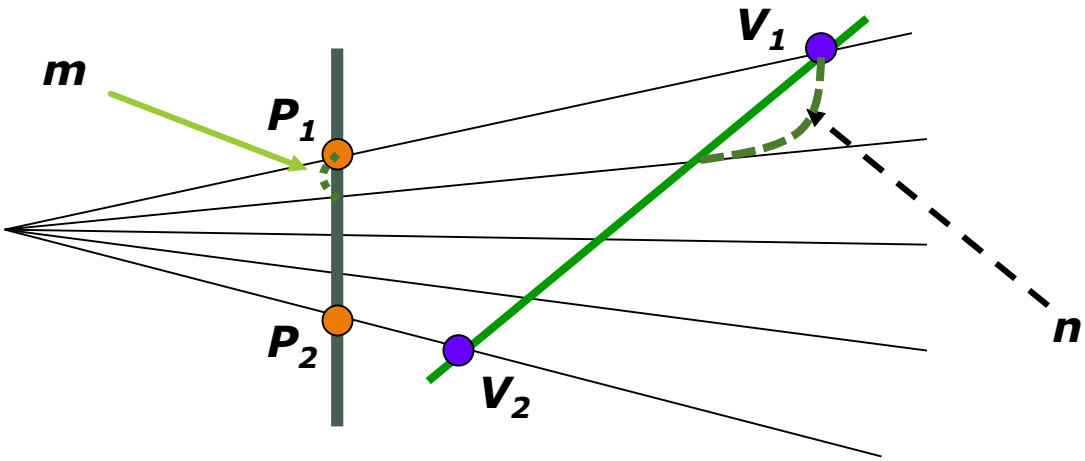
Interpolation

- Can we directly use projected x, y for texture coordinate interpolation?



Pictures from lecture notes of “Computer Graphics”, UNC

Reminder: Screen Space vs. 3D space



- Interpolation in screen space
  - $P(m) = P_1 + m(P_2 - P_1)$
- Interpolation in 3D space
  - $V(n) = V_1 + n(V_2 - V_1)$
  - $P_y(n) = V_y(n) / V_z(n)$

Mapping from Screen Space to 3D Space

$$P_y = \frac{y_1}{z_1} + m\left(\frac{y_2}{z_2} - \frac{y_1}{z_1}\right) = \frac{y_1 + n(y_2 - y_1)}{z_1 + n(z_2 - z_1)}$$

n in terms of ***m***

$$n = \frac{mz_1}{z_2 + m(z_1 - z_2)}$$

$$T(n) = T_1 + n(T_2 - T_1)$$

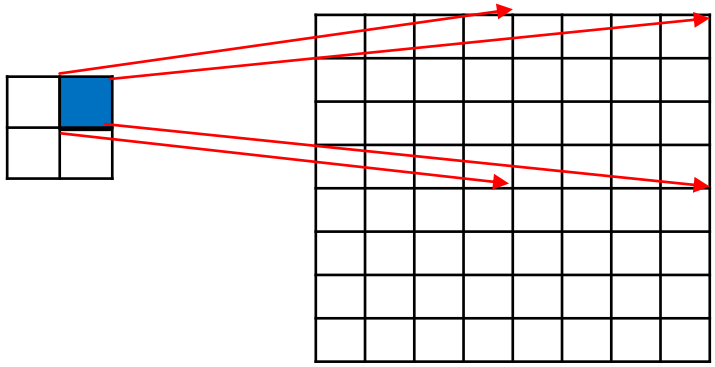
$$\begin{aligned} T(n) &= (1-n)T_1 + n\,T_2 \\ &= \frac{(1-m)z_2}{(1-m)z_2 + mz_1} T_1 + \frac{mz_1}{(1-m)z_2 + mz_1} T_2 \\ &= \frac{(1-m)\frac{1}{z_1}}{(1-m)\frac{1}{z_1} + m\frac{1}{z_2}} T_1 + \frac{m\frac{1}{z_2}}{(1-m)\frac{1}{z_1} + m\frac{1}{z_2}} T_2 \end{aligned}$$

$$M_{normpers} = \begin{bmatrix} -z_{near}\dfrac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & 0 \\ 0 & -z_{near}\dfrac{2}{yw_{\max} - yw_{\min}} & 0 & 0 \\ 0 & 0 & \dfrac{z_{near} + z_{far}}{z_{near} - z_{far}} & \dfrac{-2z_{near}z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note: Graphics systems (e.g. OpenGL) can take the z from w of (x, y, z, w) for perspective correct interpolation.

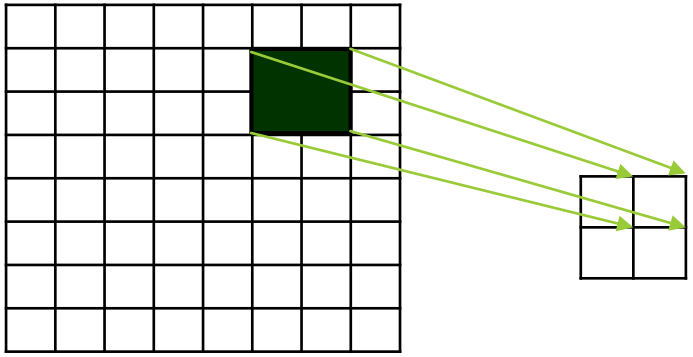
Magnification and Minification

- Minification
  - More than one texel can cover a pixel
- Magnification
  - More than one pixel can cover a texel



Texture Polygon

Magnification



Texture Polygon

Minification

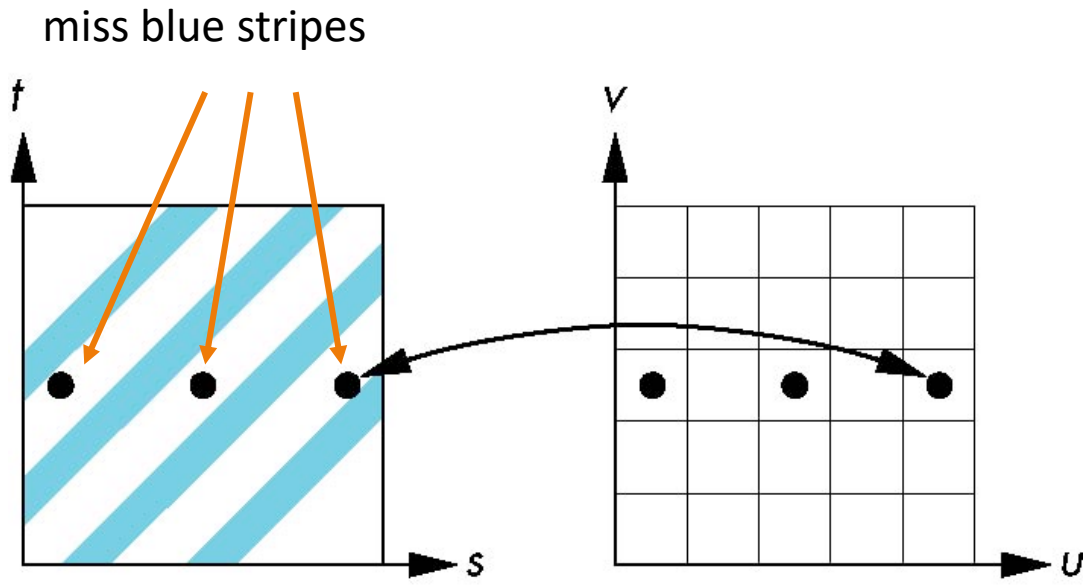
point sampling (nearest texel) is the most efficient approach, but …

Aliasing

Ref: www.relisoft.com/Science/Graphics/alias.html

Aliasing

- Point sampling of the texture can lead to aliasing errors

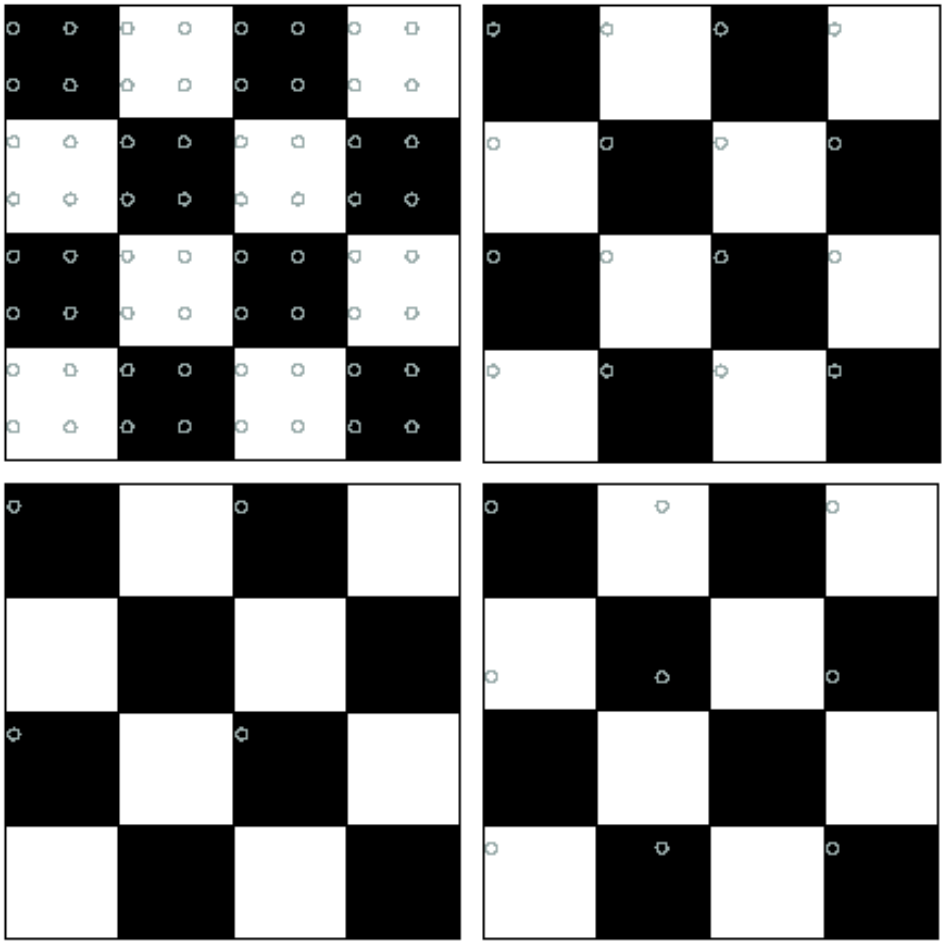


point samples in texture space

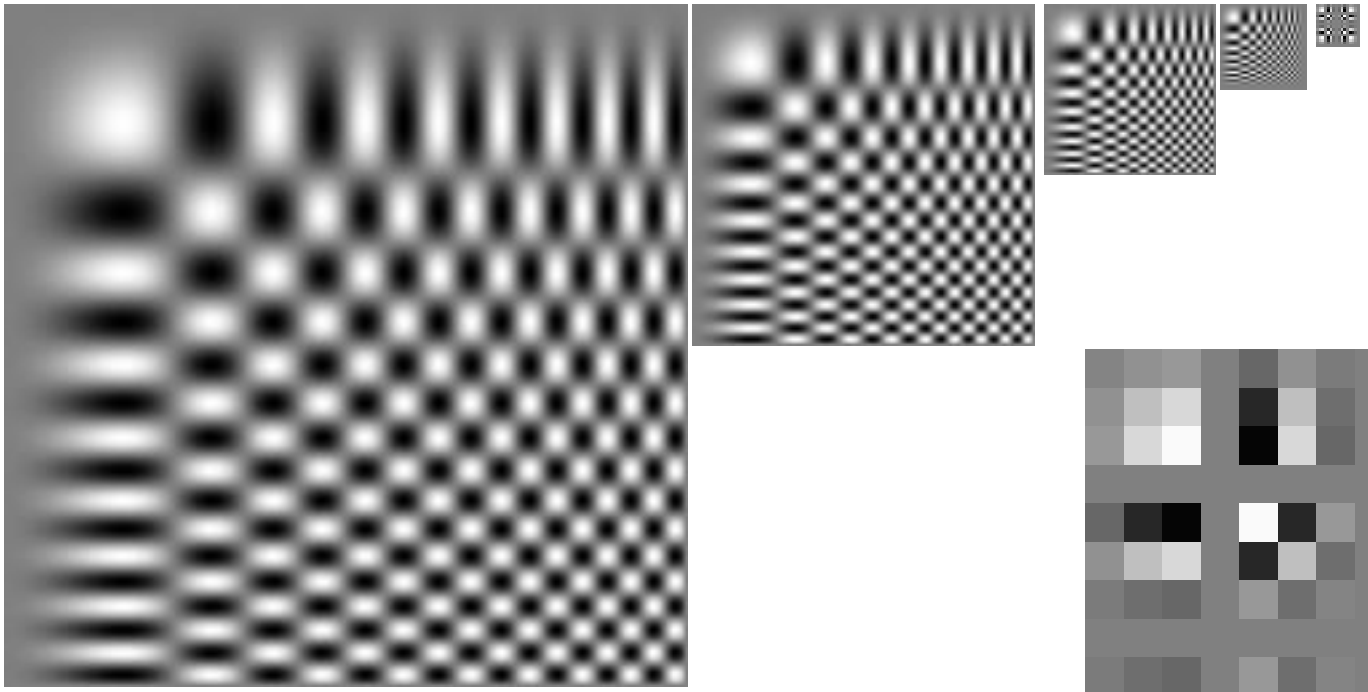
point samples in u,v (or x,y,z) space

Re-sampling

- Resample the checkerboard by taking one sample at each circle.



Simple sampling, but …



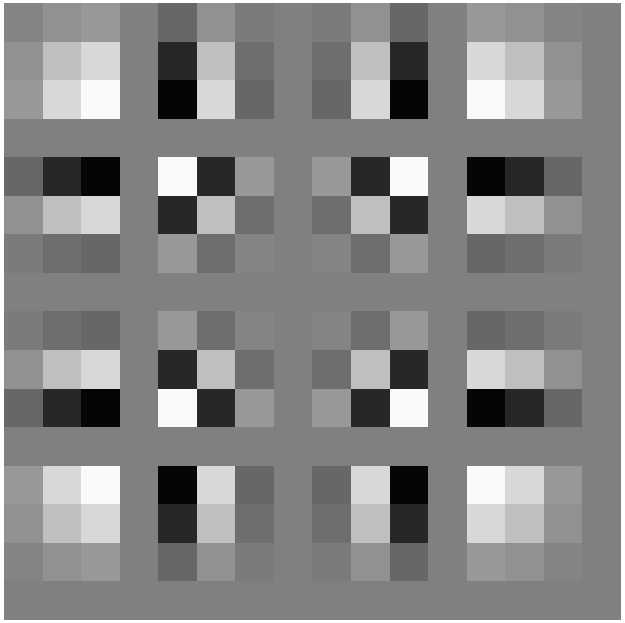
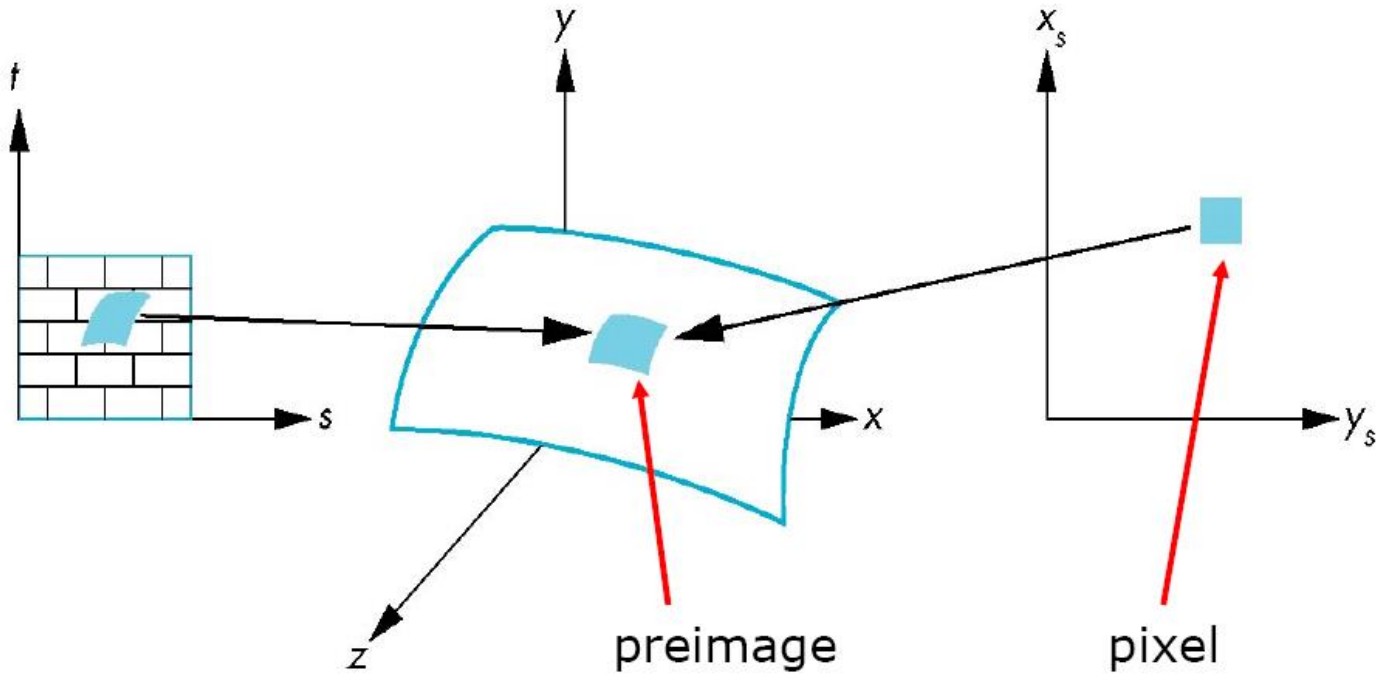


Fig. from: David A. Forsyth and Jean Ponce, Computer Vision: A Modern Approach, Prentice Hall, New Jersey, 2003.

Area Averaging

- A better but slower option is to use area averaging



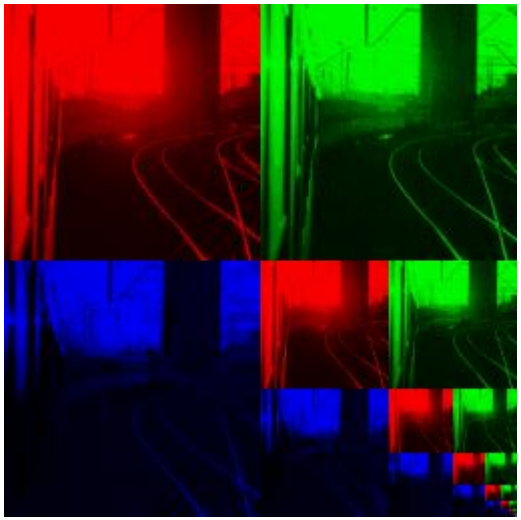
Area Averaging

www.relisoft.com/Science/Graphics/alias.html

Mipmapped Textures

- On-line processing or pre-filtering?
- Mipmapping allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects

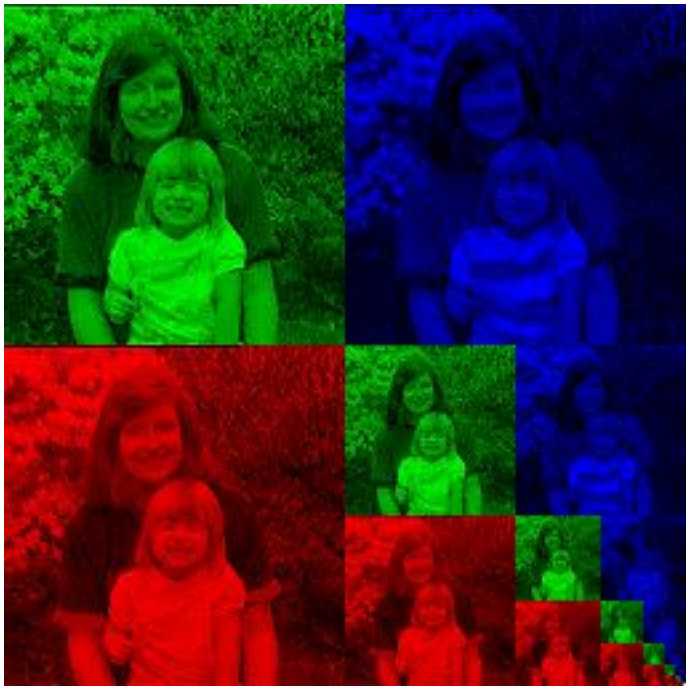
MipMap

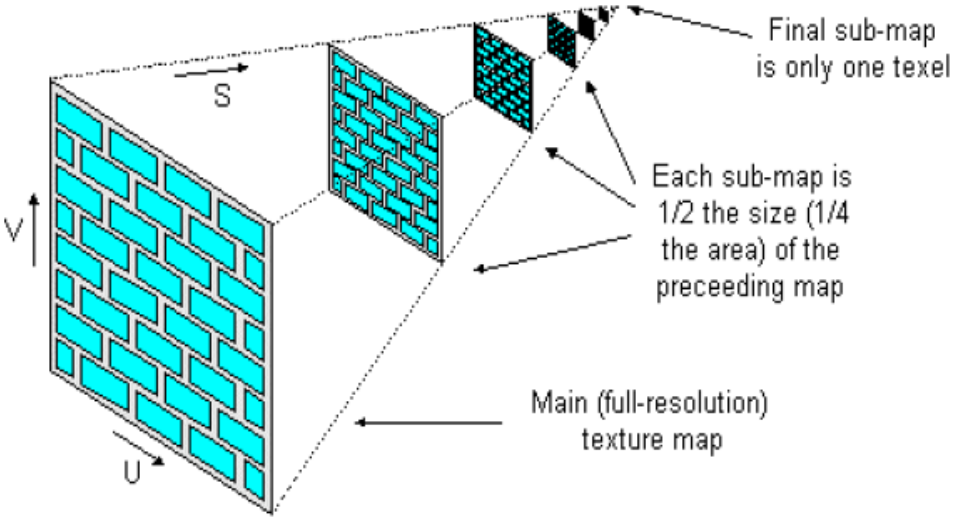


storage

Mipmapping

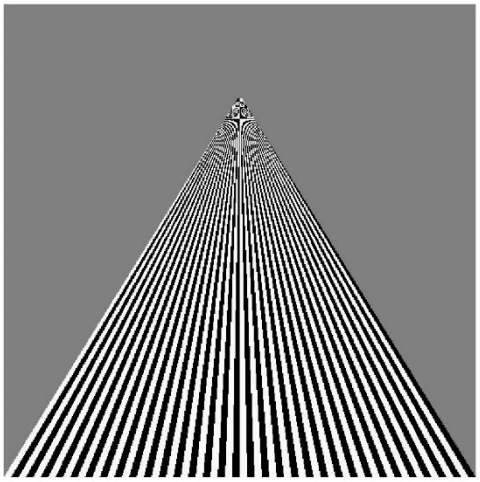
- 1/3 overhead of maintaining the MIP map.

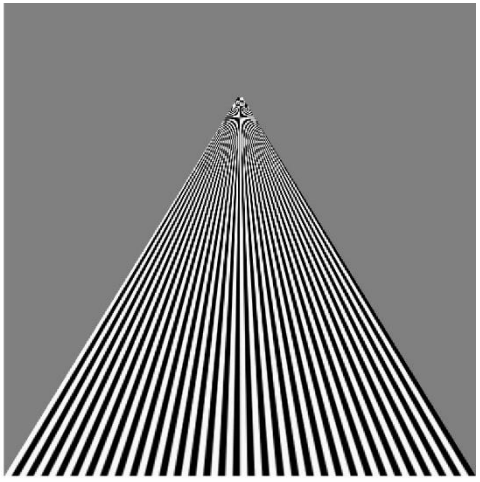




Example

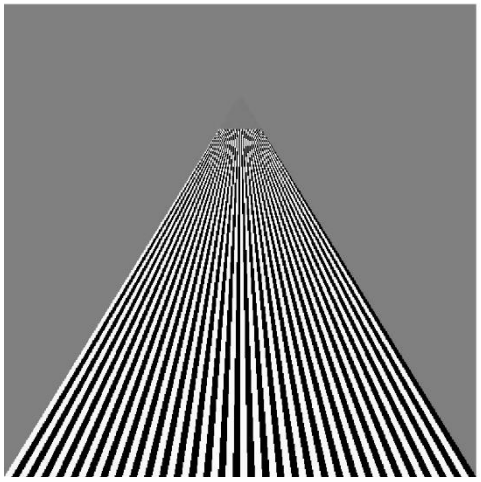
point sampling

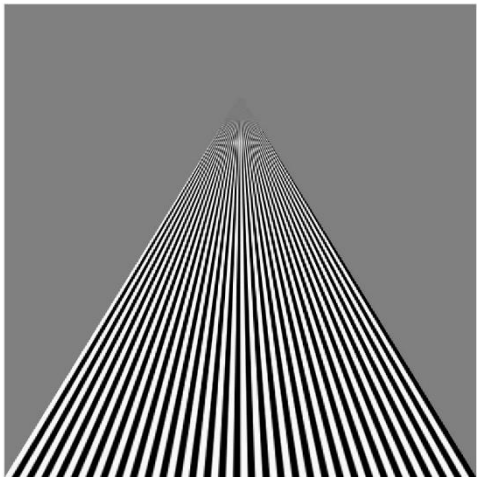




linear filtering

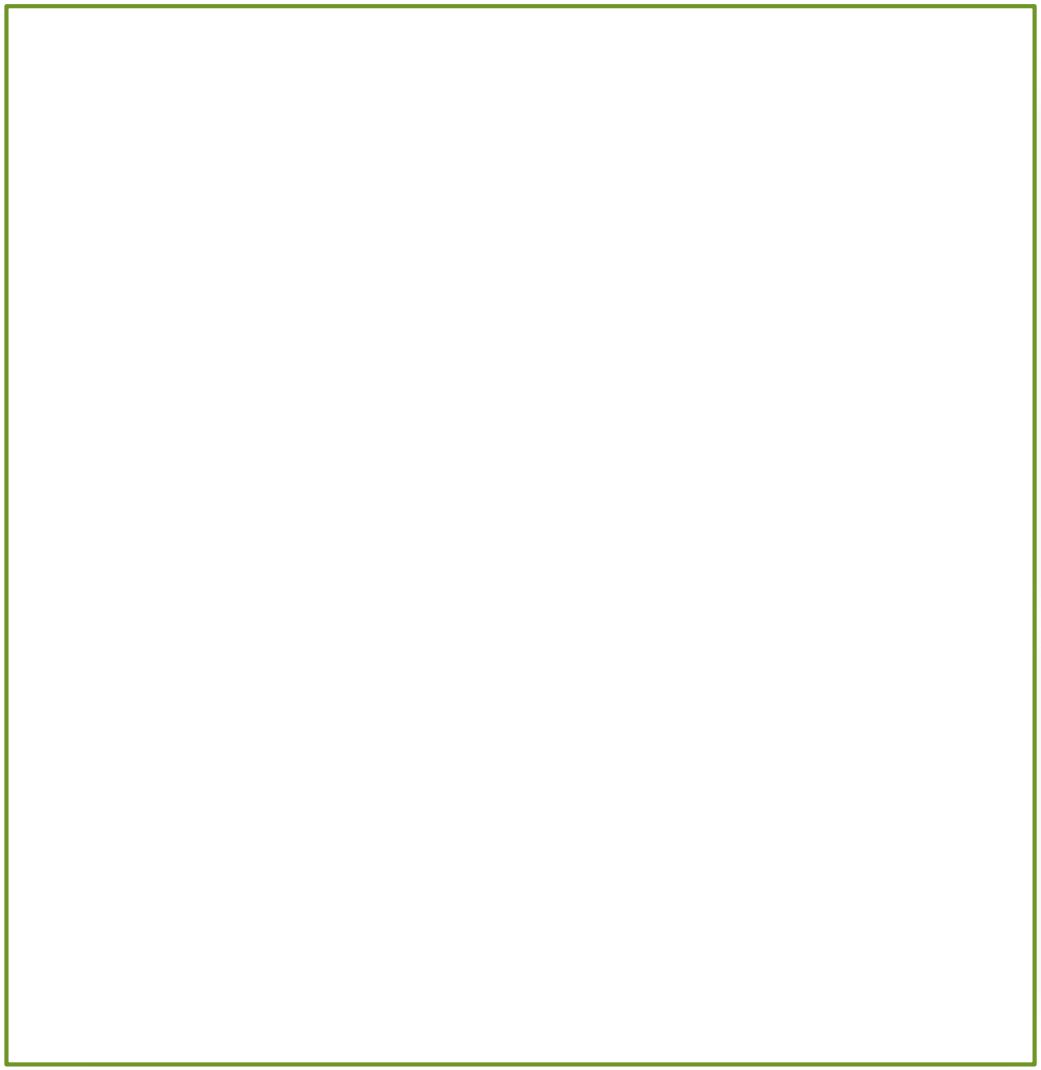
mipmapped point sampling





mipmapped linear filtering

Examples of Highly Reflected Models



T1000 from movie “Terminator 2”

Silver Surfer from movie “Fantastic 4:Rise of the Silver Surfer”

How to Handle Highly Specular Surfaces?

- How to render a flat mirror?
- How to render a mirror-like object in a virtual scene?
- How about rendering such an object in a real scene?

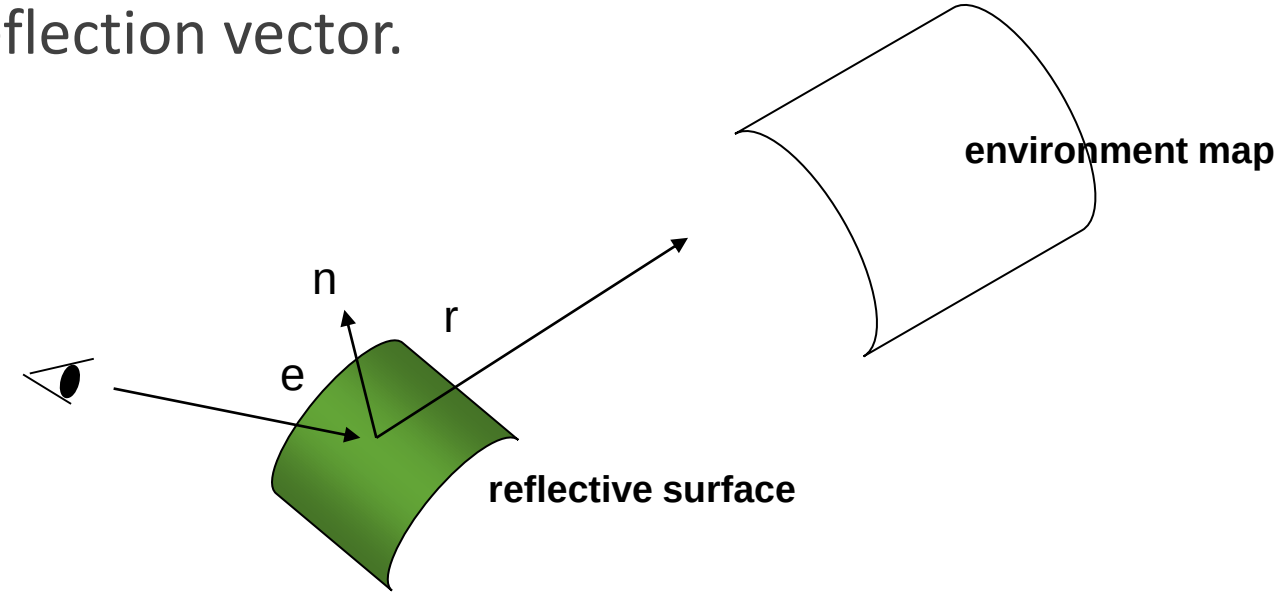
Environment Mapping

- For real-time applications
- A.k.a reflection mapping
- First proposed by Blinn and Newell.
- A efficient way to create reflections on curved surfaces
  - can be implemented using texture mapping supported by graphics hardware

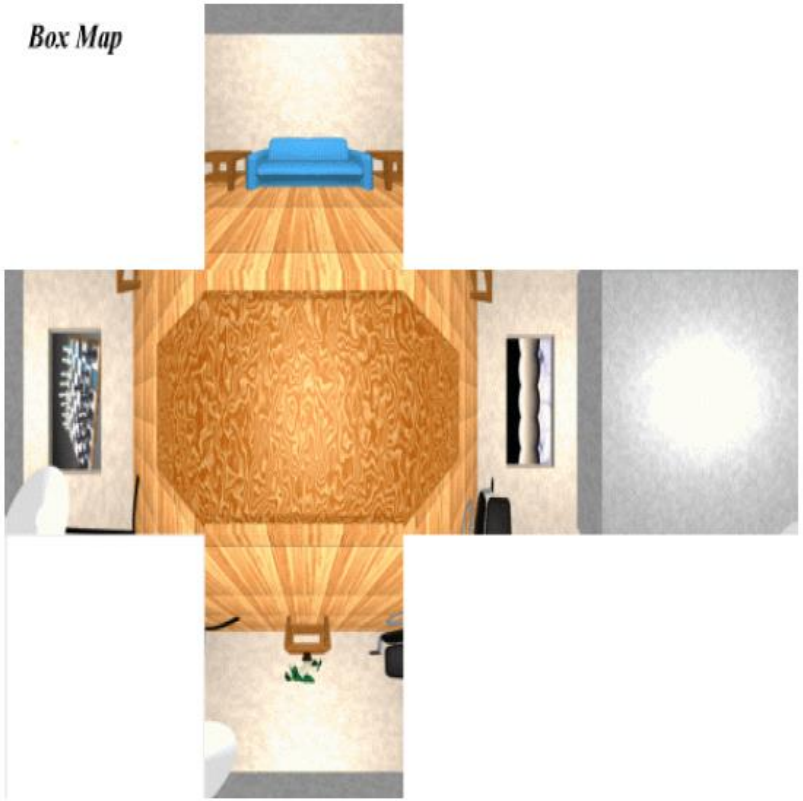
Environment Mapping

- Assume the environment is far away and there's no self-reflection

- The reflection at a point can be solely decided by the reflection vector.



Environment Mapping



Pictures from lecture notes of Computer Graphics course, UNC

Sphere Mapping

- The image texture is taken from a perfectly reflective sphere.
- Assume the size of the sphere →0. Map the rays to the environment
- Using orthogonal projection.

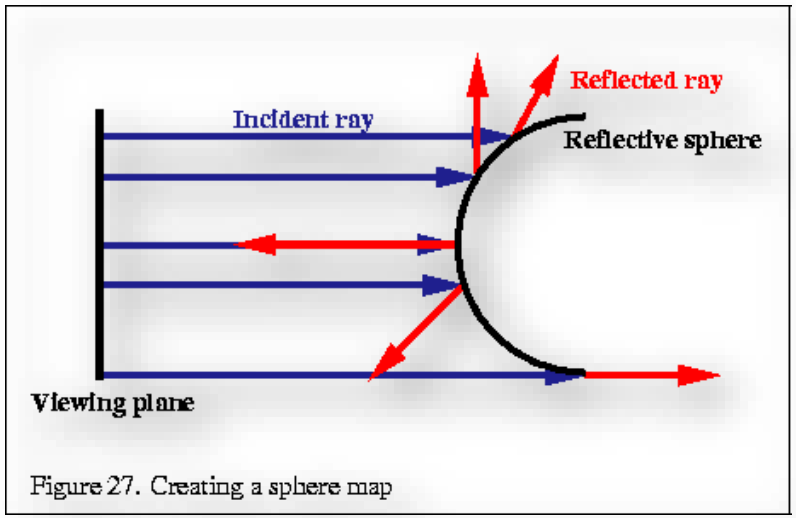
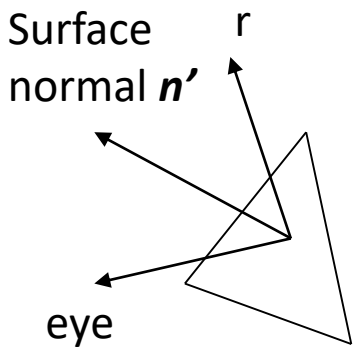


Figure 27. Creating a sphere map


Figure 31. Sphere map generated from image cube faces in Figure 30

Pictures from OpenGL tutorial. http://www.opengl.org

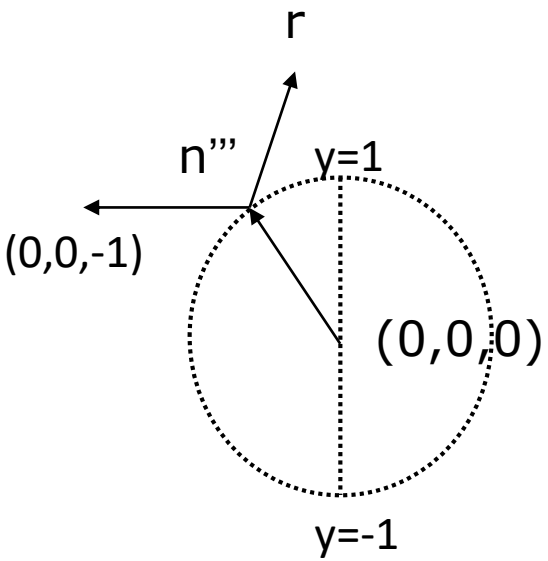
Sphere Mapping



- To access the sphere map texture
  - Compute the reflection vector *r* on the object surface by *e* and *n'*.

    ($r = (r_x, r_y, r_z) = -e' + 2(n' \cdot e')n'$)
  - Access the texture: compute the sphere normal in the local space

    $n'' = (r_x, r_y, r_z) + (0,0,-1)$



$$n''' = \left(\frac{r_x}{m}, \frac{r_y}{m}, \frac{r_z - 1}{m} \right) \qquad m = \sqrt{r_x^{\,2} + r_y^{\,2} + (r_z - 1)^2}$$

- Normalized the screen space from [-1,1] to [0,1]

$$s = \frac{r_x}{2m} + \frac{1}{2} \qquad t = \frac{r_y}{2m} + \frac{1}{2}$$

- (s, t) is the target texture coordinate

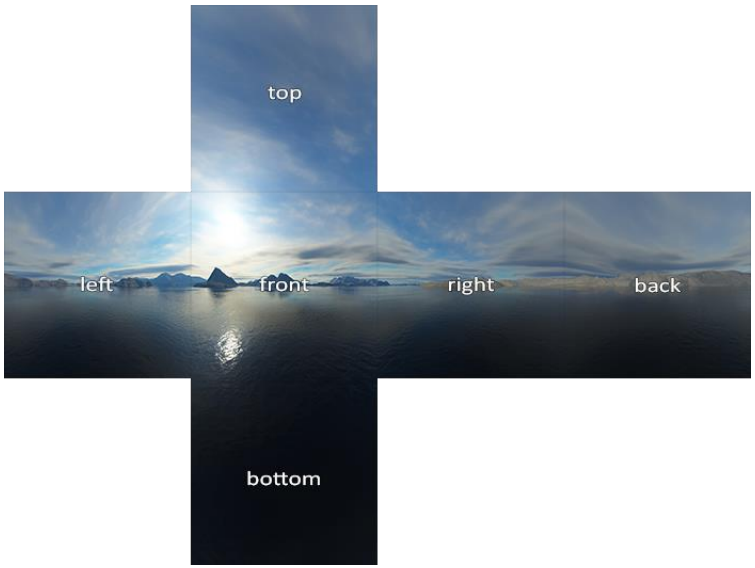
Sphere Mapping

Samples from DirectX SDK

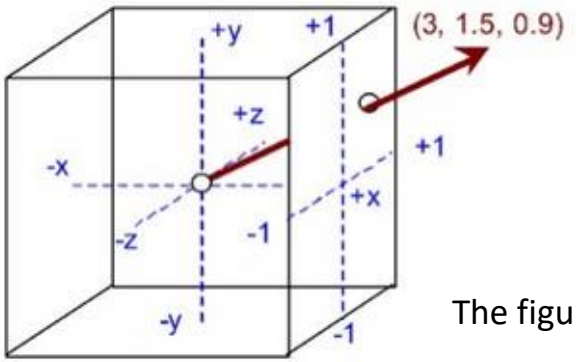
Cubemap in OpenGL

- In modern OpenGL, A special kind of texture, Cube Map, consists of six images, can be indexed by (s, t, r).

```
glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
……
for(unsigned int i = 0; i < 6; i++) {
    glTexImage2D(
        GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
        0, GL_RGB, width, height, 0, GL_RGB,
        GL_UNSIGNED_BYTE, data[i]    ); }
```



The example is extracted from leanopengl.com



The figure is from Mark Kilgard, Computer Graphics Slides, Univ. of Texas, Austin

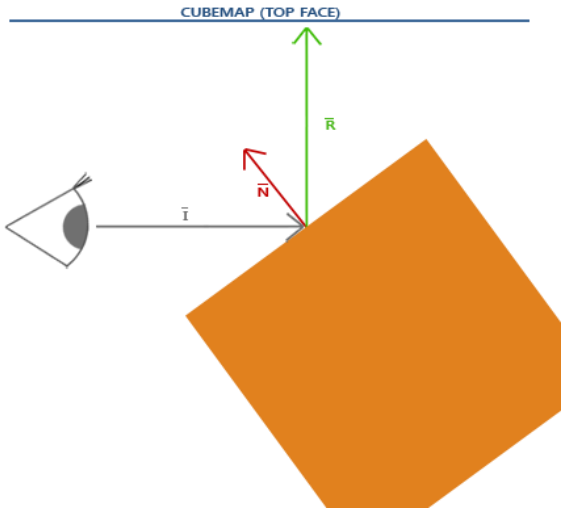
Cubemap for Environment Mapping

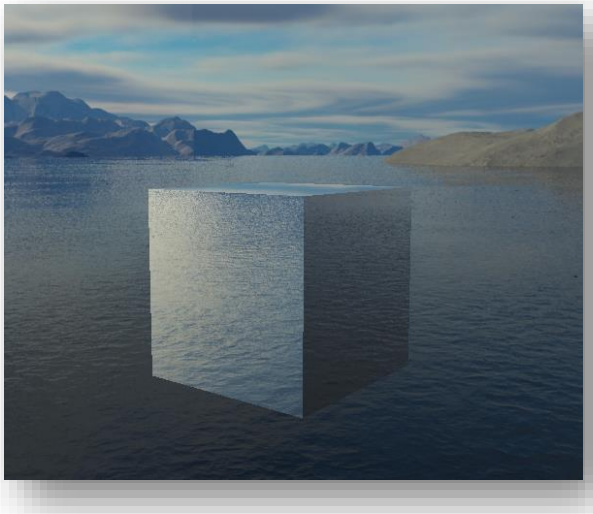
```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 Position;

uniform vec3 cameraPos;
uniform samplerCube skybox;

void main()
{
    vec3 I = normalize(Position - cameraPos);
    vec3 R = reflect(I, normalize(Normal));
    FragColor = vec4(texture(skybox, R).rgb, 1.0);
}
```





The example is extracted from leanopengl.com

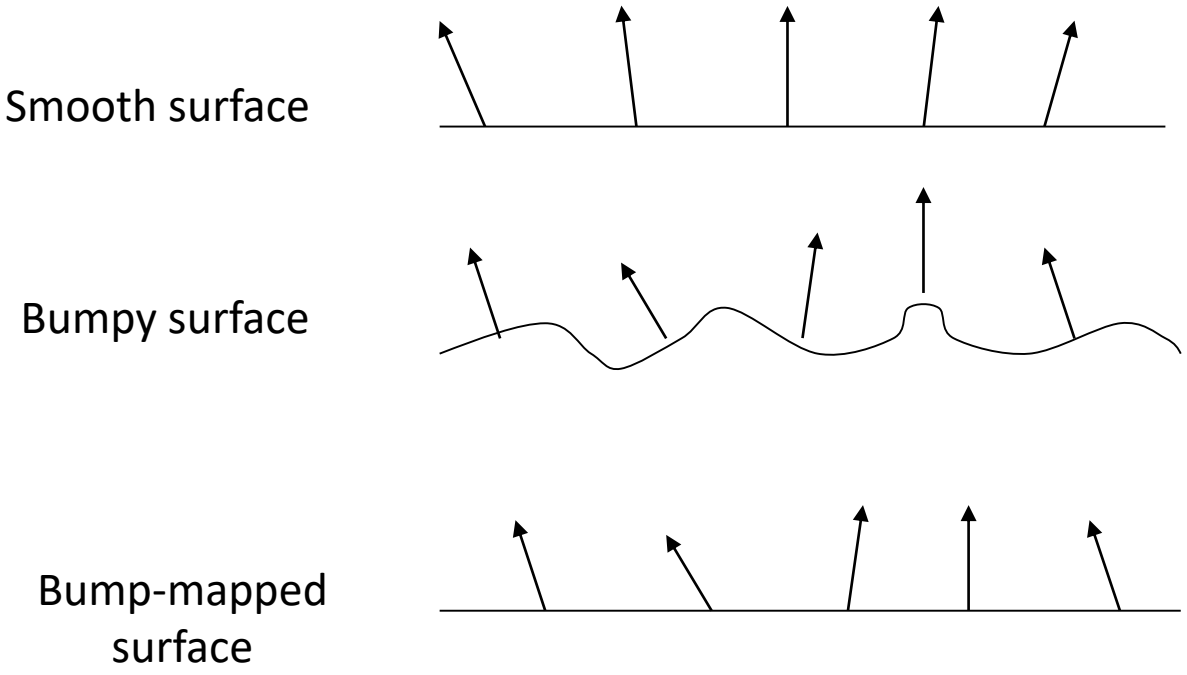
Bump and Normal Mapping

- Represent surface details and avoid heavy geometric computation.

http://www.ozone3d.net/tutorials/bump_mapping.php

Bump and Normal Mapping

- Calculate reflection (Phong Shading) with a normal map. [normal mapping]
- Or with a height map. [bump mapping]

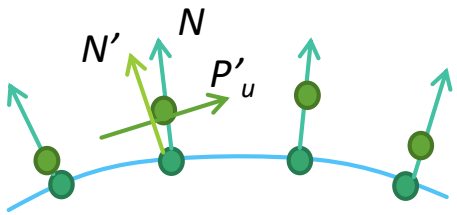


Bump Mapping

- Let $P = P(u,v)$ be a smooth parametric surface, with normals $N = N(u,v)$.
- Apply a bump map $b = b(u,v)$:

$$P' = P + bN$$

$$N' = P'_u \times P'_v$$



$$P'_u = \frac{\partial}{\partial u}(P + bN) = P_u + b_u N + bN_u \approx P_u + b_u N$$

$$P'_v = \frac{\partial}{\partial v}(P + bN) = P_v + b_v N + bN_v \approx P_v + b_v N$$

P_u – Tangent at P in u direction
P_v – Tangent at P in v direction

Bump Mapping (cont.)

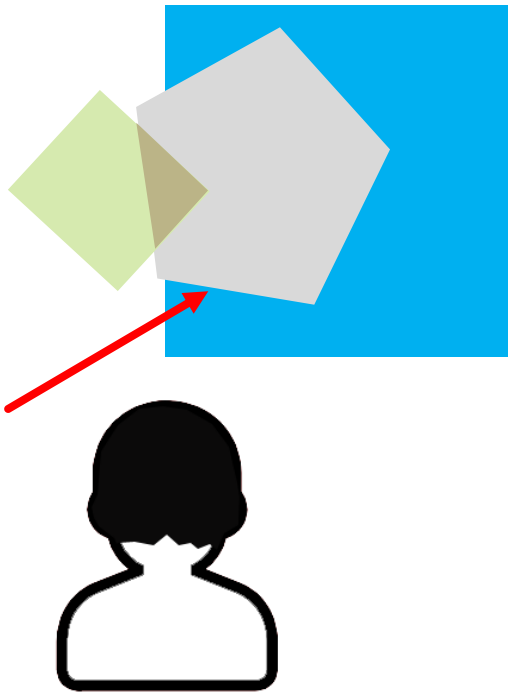
$$N' \approx (P_u + b_u N) \times (P_v + b_v N)$$
$$= P_u \times P_v + b_u (N \times P_v) + b_v (P_u \times N) + b_u b_v (N \times N)$$
$$= N + b_u (N \times P_v) + b_v (P_u \times N)$$

E.g.
When N = (0, 0, 1), N x P_v = (-1, 0, 0), P_u x N = (0, -1, 0),
N' becomes ($-b_u$, $-b_v$, 1)

Compositing, Blending and Accumulation Buffer

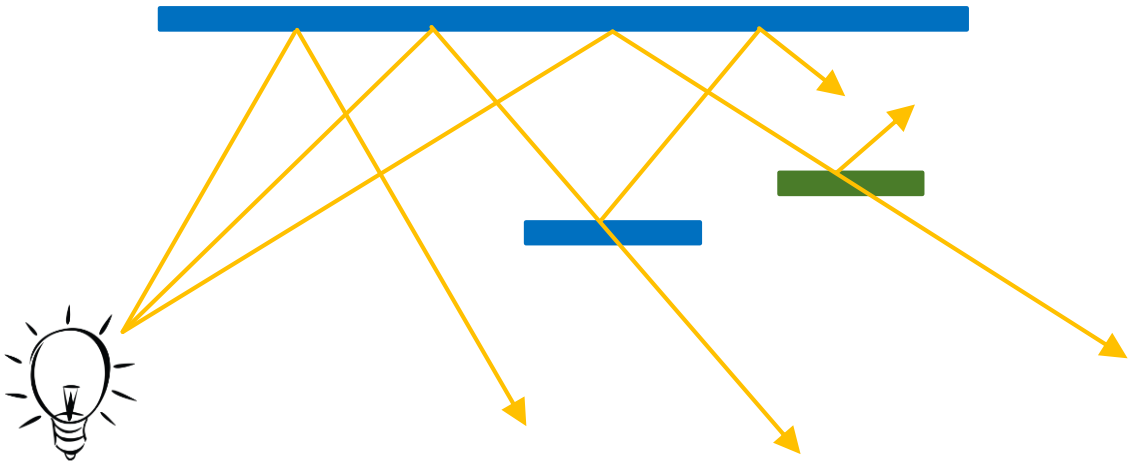
Opacity and Transparency

- Opaque surfaces permit no light to pass through
- Transparent surfaces permit all light to pass
- Translucent surfaces pass some light
  - translucency = 1 – opacity (α)



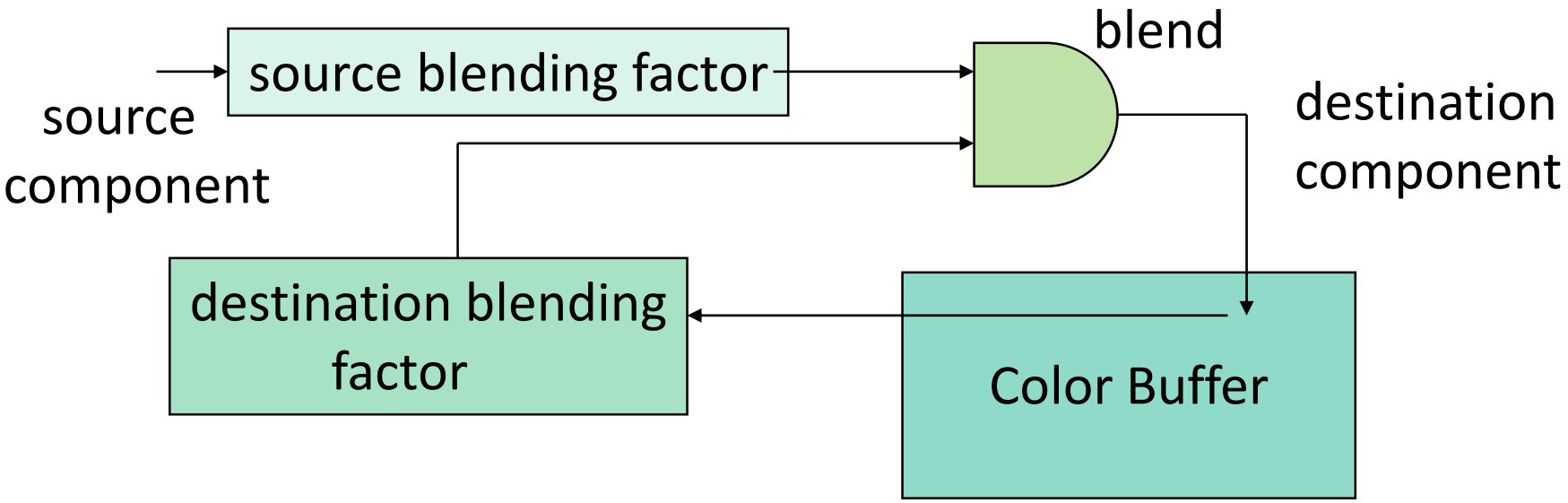
Physical Models

- Dealing with translucency in a physically correct manner is difficult due to
  - the complexity of the internal interactions of light and matter
  - Using a pipeline renderer



Writing Model

- Use A component of RGBA (or RGBα) color to store opacity
- During rendering we can expand our writing model to use RGBA values



Blending Equation (general)

- We can define source and destination blending color for each RGBA component
  - $S = [s_r, s_g, s_b, s_\alpha]$
  - $D = [d_r, d_g, d_b, d_\alpha]$

- Suppose that the source and destination blending factors are
  - $b = [b_r, b_g, b_b, b_\alpha]$
  - $c = [c_r, c_g, c_b, c_\alpha]$

- Blend as
  - $c' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g , b_b s_b + c_b d_b , b_\alpha s_\alpha + c_\alpha d_\alpha]$

Blending in practice

- glEnable(GL_BLEND);
  glBlendFunc(source_factor, destination_factor)
- Only certain factors supported:
  GL_ZERO, GL_ONE,
  GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA,
  GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA

While we the source **α** as the source blending factor and **1− α** for the destination factor

$$(R_d', G_d', B_d', \alpha_d') = (\alpha_s R_s + (1-\alpha_s)R_d, \alpha_s G + (1-\alpha_s)G_d, \alpha_s B_s + (1-\alpha_s)B_d, \alpha_s\alpha_s + (1-\alpha_s)\alpha_d).$$

It ensures that neither colors nor opacities can saturate, but …

Alpha Blending for Objects with Fine Borders

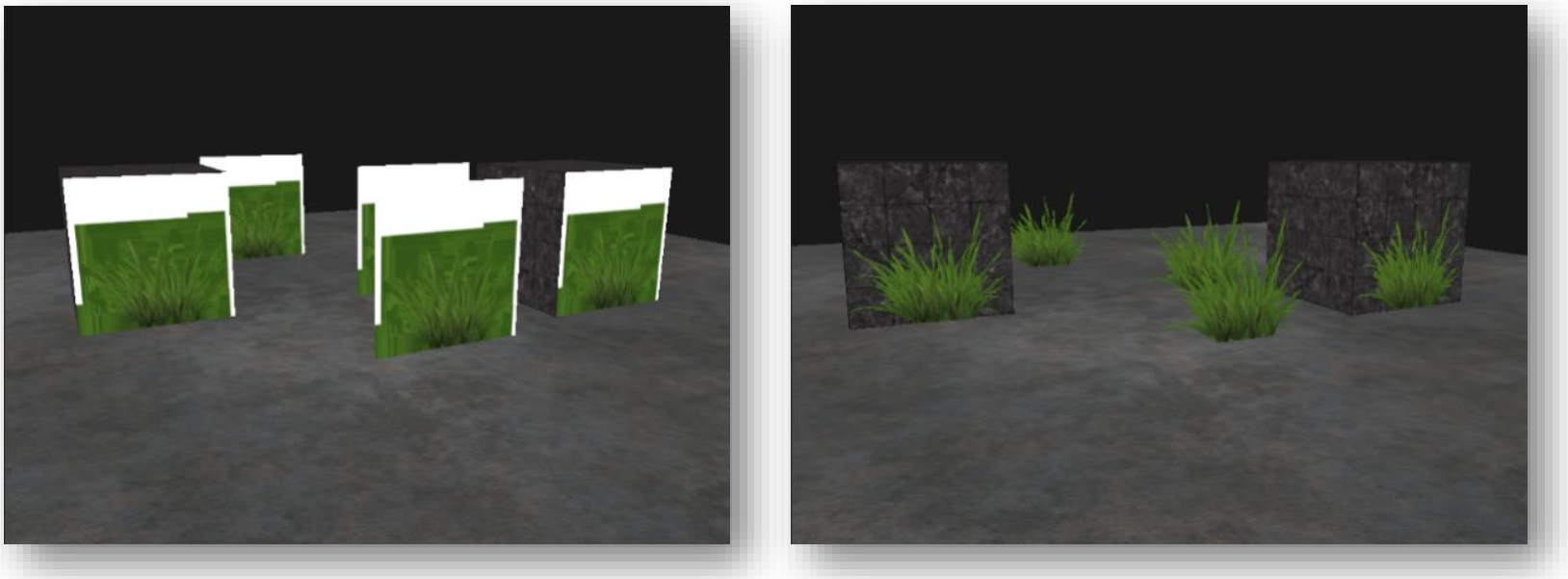
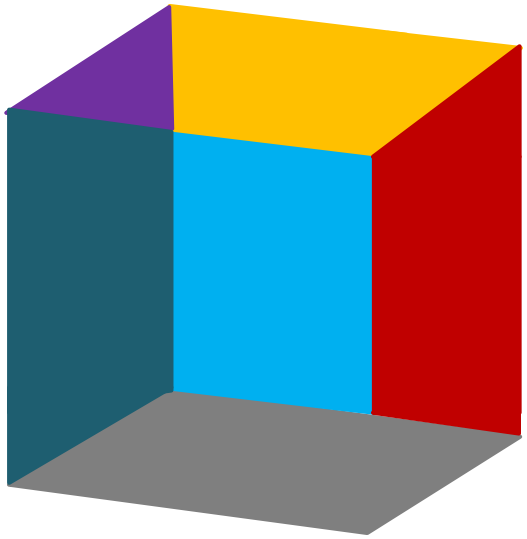


Fig. from learnopengl.com/Advanced-OpenGL/Blending

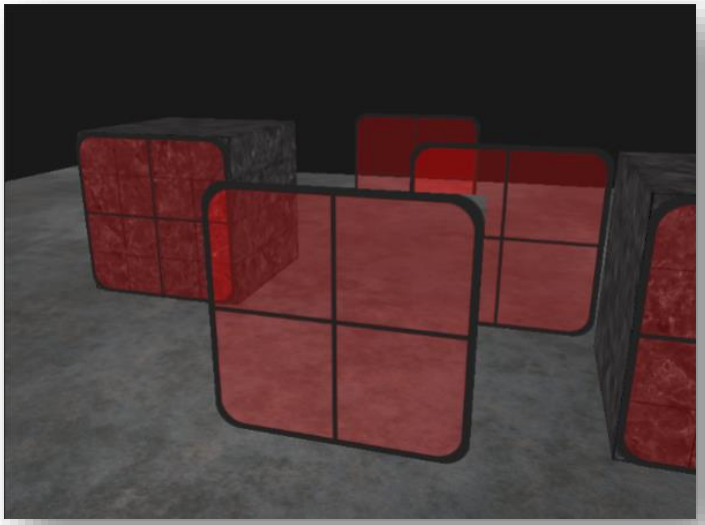
Order Dependency

- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent

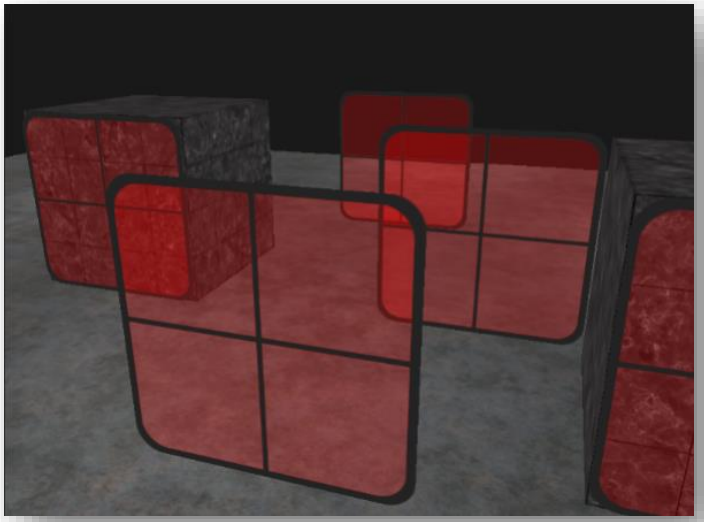


Ordering in Alpha Blending

Something wrong?



The most front window occludes the window behind due to depth buffer



When dealing with transparent windows

- Render patches **from the furthest to nearest**
- Do the depth test **without updating the depth** values.

Examples from https://learnopengl.com/Advanced-OpenGL/Blending

Opaque and Translucent Polygons

- Suppose that we have a group of polygons some of which are opaque and some translucent
- Opaque polygons block all polygons behind them and affect the depth buffer

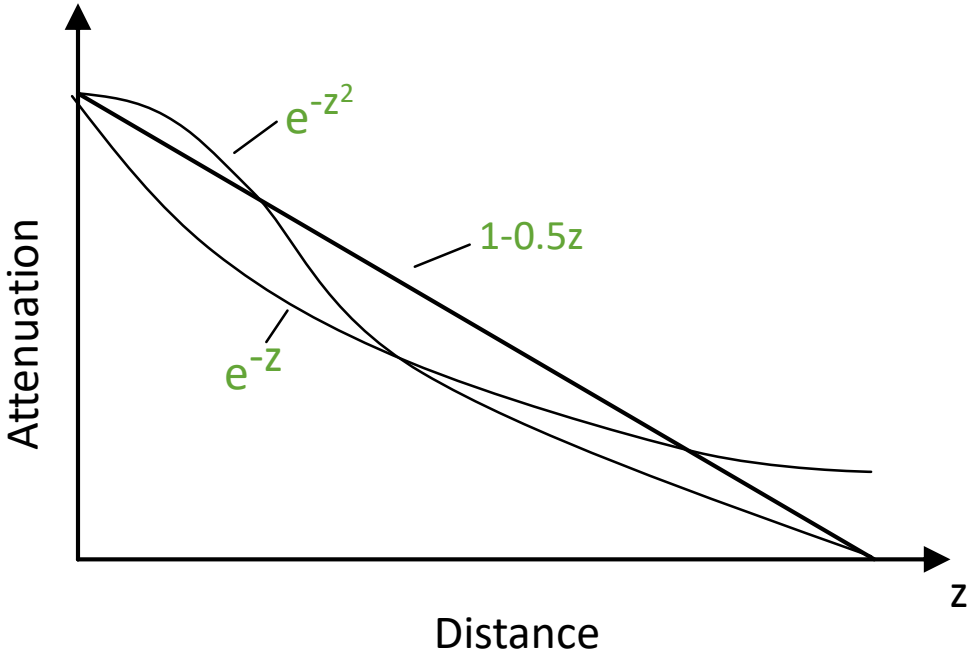
- Translucent polygons should not affect depth buffer
  - Render with **`glDepthMask(GL_FALSE)`** which makes depth buffer read-only
- Sort polygons first to remove order dependency

Fog

- We can composite with a fixed color and have the blending factors depend on depth
  - Simulates a fog effect
  - Blend source color Cs and fog color C_f by
  - $C_s' = f\, C_s + (1-f)\, C_f$

- *f* is the *fog factor*
  - Exponential
  - Gaussian
  - Linear

Fog Functions



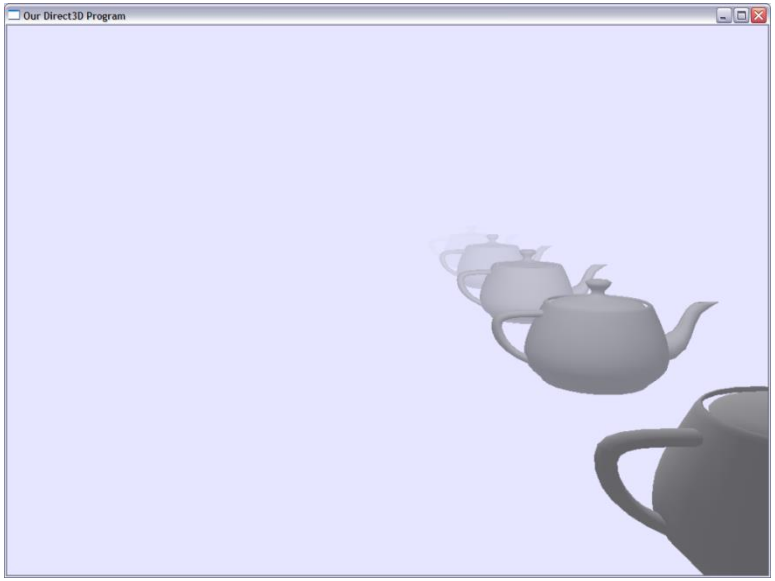


Figure from: http://www.directxtutorial.com

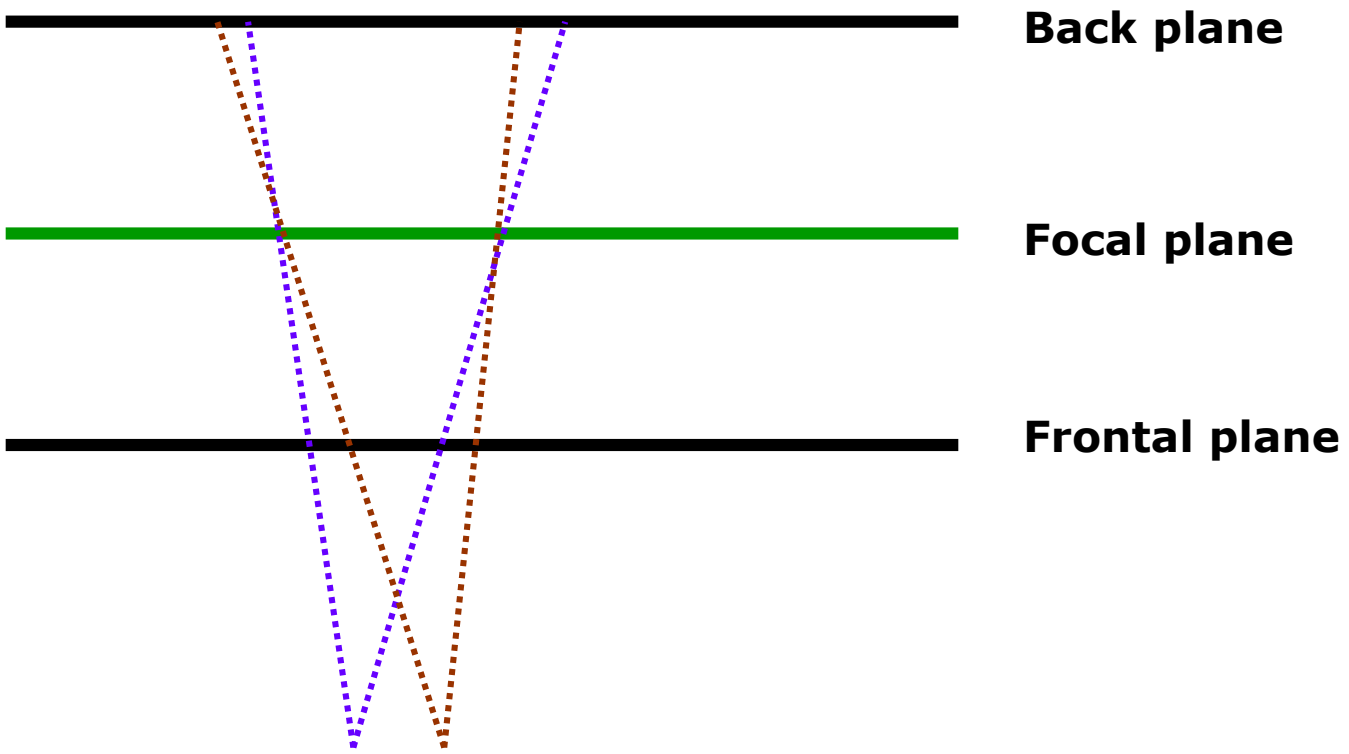
Accumulation Buffer

- Compositing and blending are limited by resolution of the frame buffer
  - Typically 8 bits per color component
- The accumulation buffer is a high resolution buffer
  - 16 or more bits per component
  - Write into it or read from it with a scale factor
- Slower than direct compositing into the frame buffer
- Now deprecated but its operations can be done with floating point frame buffers

Applications

- Compositing
- Image Filtering
- Motion effects
- Full screen antialiasing
- …..

Depth of Focus

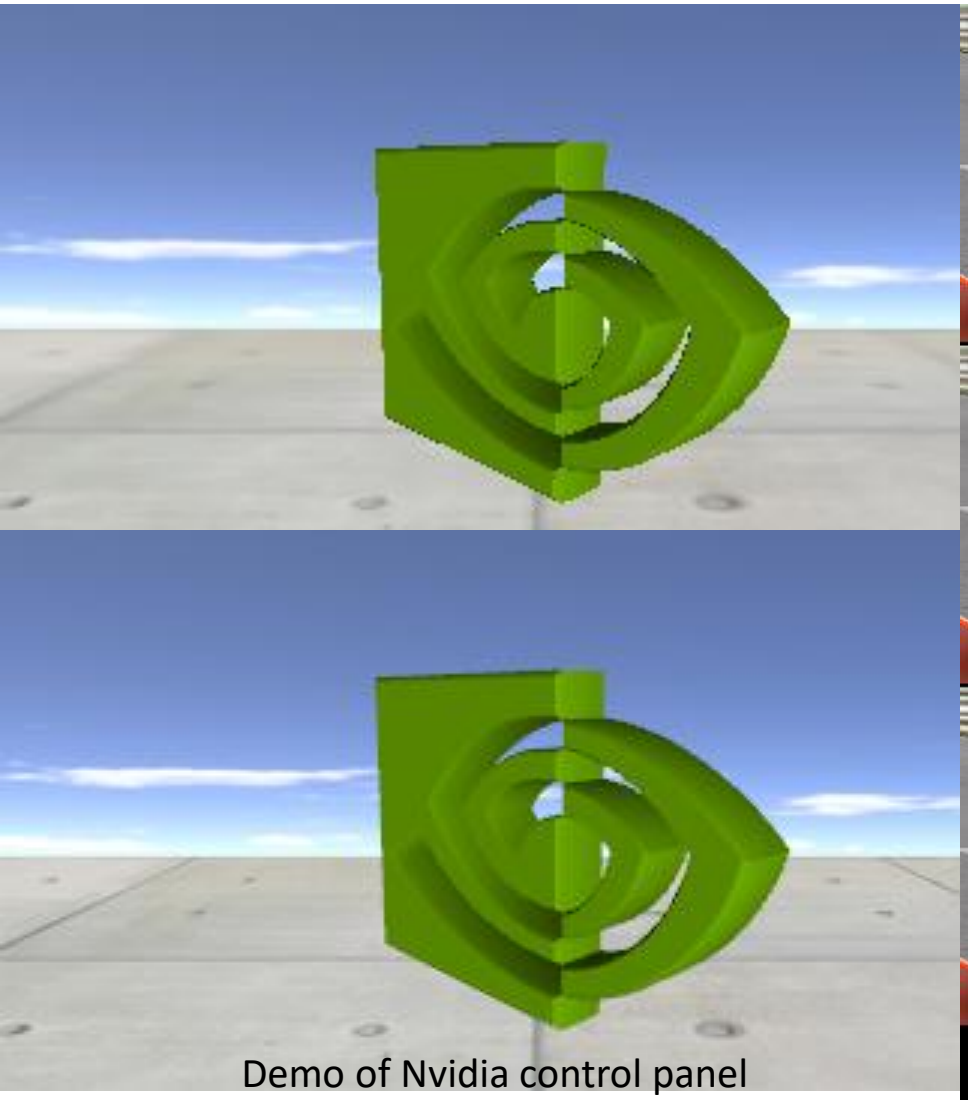


Note: there are multiple ways to mimic focusing.

Motion blur

http://www.eml.hiroshima-u.ac.jp/gallery/ComputerGraphics/motion_blur/

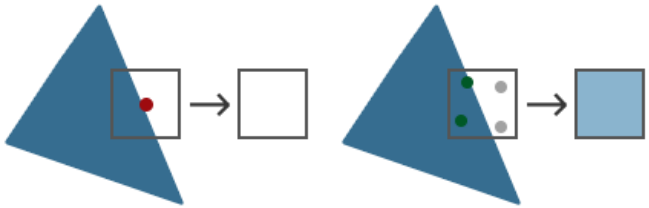
Anti-aliasing (Full screen and Multiple samples)

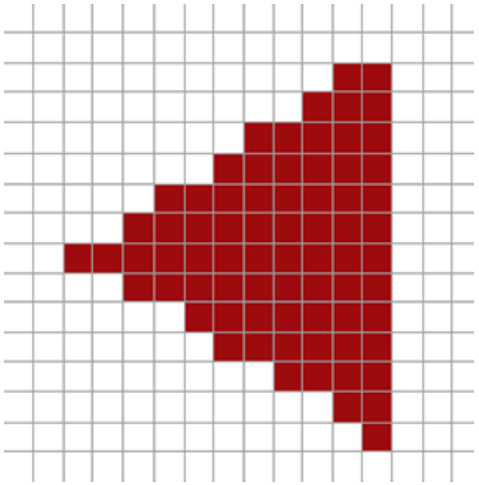


Demo of Nvidia control panel

Multisampling Anti-aliasing (MSAA)

- The fragment shader still runs once per pixel for each primitive.
- MSAA then uses a larger depth/stencil buffer to determine subsample coverage.





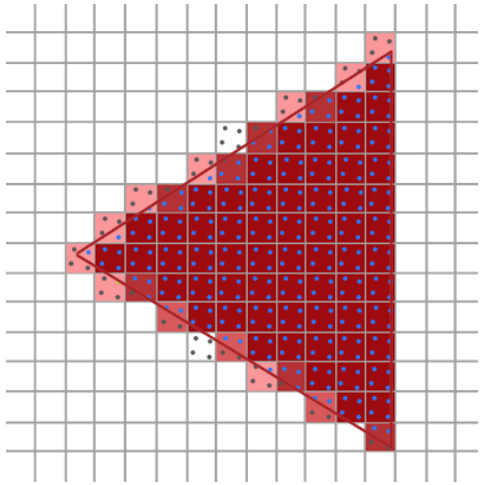


Fig. from learnopengl.com

Deep learning super sampling (DLSS) 2.0

- Convolutional autoencoder takes the *low resolution current frame* (the aliased image and motion vectors), and the *high resolution previous frame*, to generate a higher quality current frame.

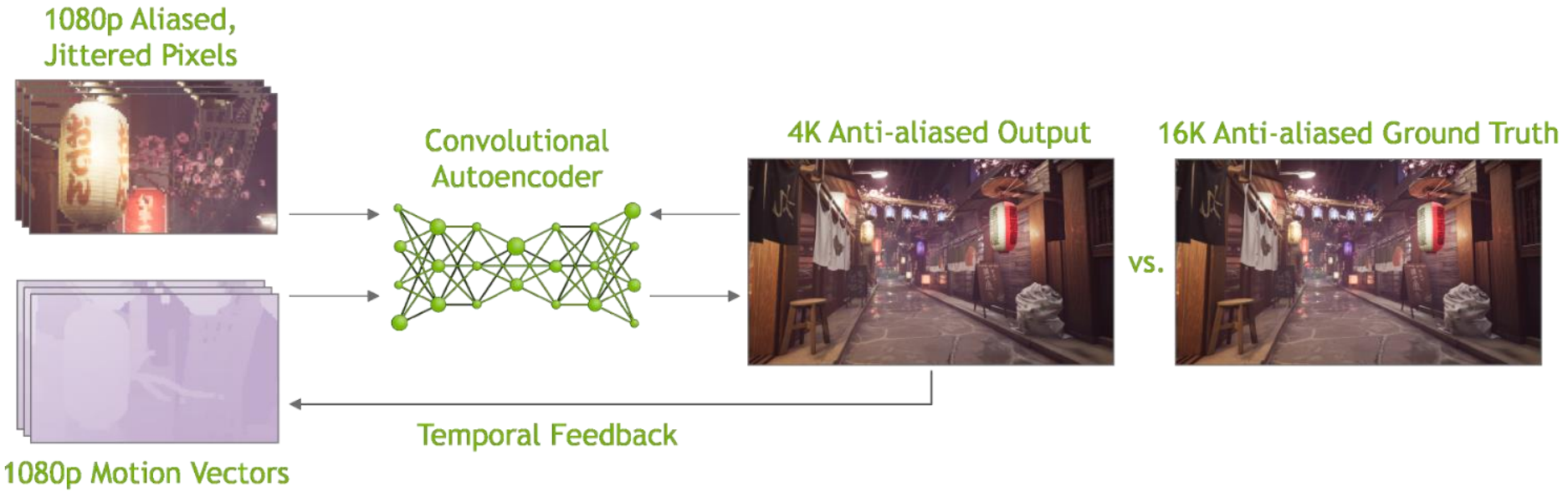


Fig. from NVIDIA DLSS 2.0

DLSS 3 further enhances real-time ray tracing

The End of Chapter