Introduction to Computer Graphics

2016 Spring

National Cheng Kung University

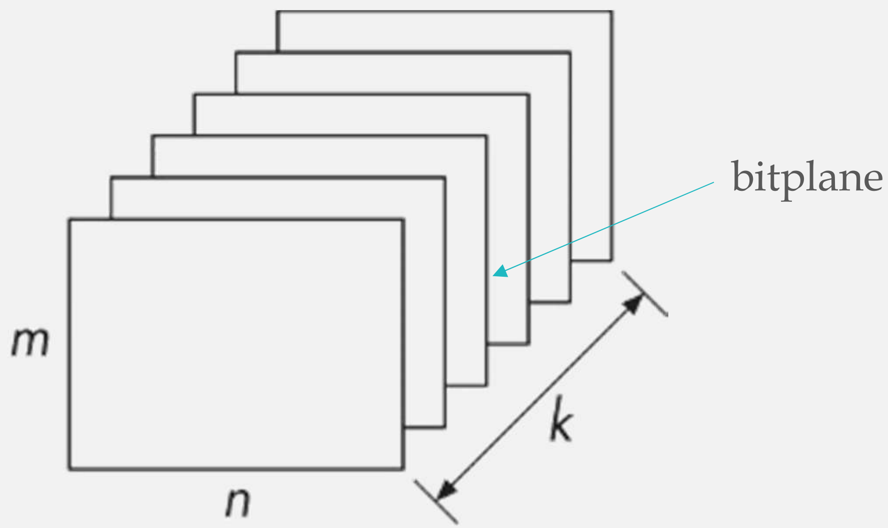
Instructors: Min-Chun Hu 胡敏君

Shih-Chin Weng 翁士欽 (西基電腦動畫)

Buffers and Mapping Techniques

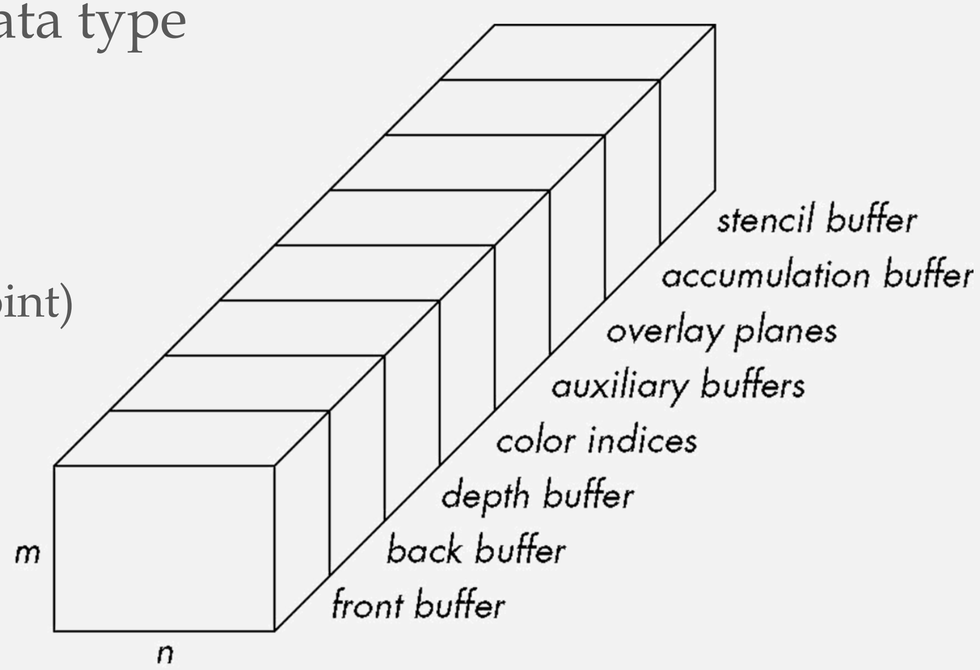
Buffer

- Define a buffer as a block of memory with n × m elements in spatial resolution and k elements in depth/precision.
  - k : the number of bits used to represent each pixel



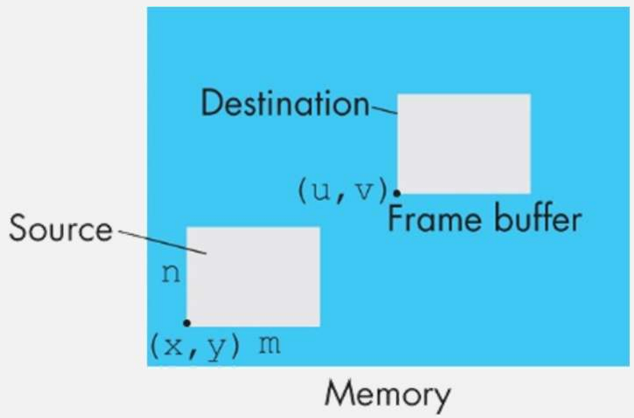
OpenGL Frame Buffers

- Each frame buffer can have different depth (k) and can be represented in different data type
  - 64 bits RGBA Color Buffers
    - 32 bits Front Buffer (byte)
    - 32 bits Back Buffer (byte)
  - 32 bits Depth Buffer (integer or floating point)
  - …



Writing in Buffers

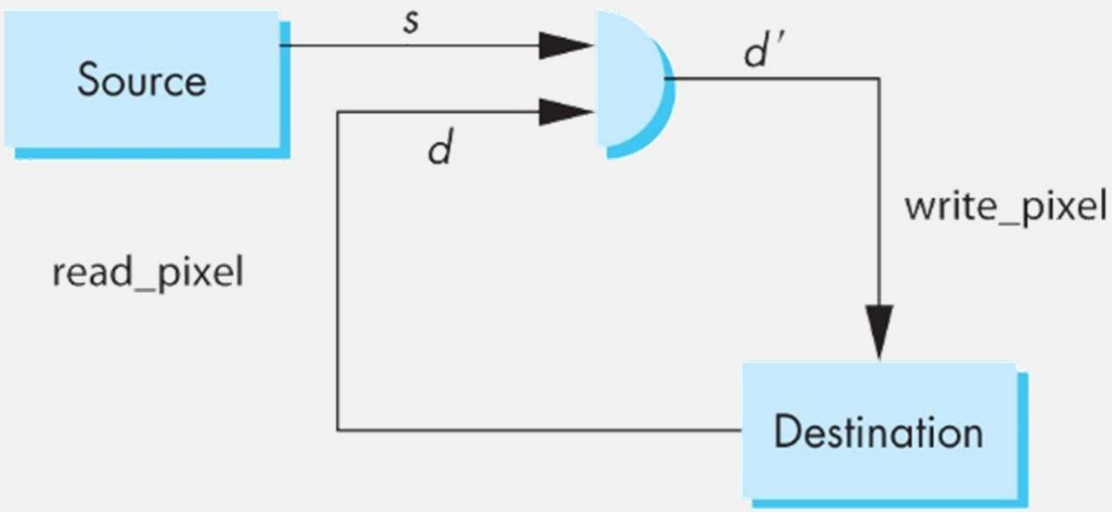
- Conceptually, we can consider all of memory as a large two-dimensional array of pixels
- We only occasionally read and write a rectangular block of pixels (i.e. frame buffer) inside the memory
  - Bit block transfer (bitblt) operations or raster operations (raster-ops)



write_block(source, n, m , x, y, destination, u, v)

Writing Model

- Read destination pixel before writing source

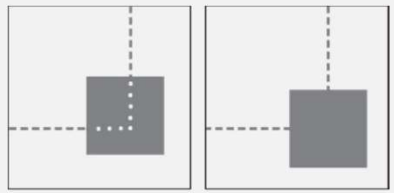


Bit Writing Modes

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

clear (0) **replace** (3) **XOR** (6) **OR** (7) **clear** (15)

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


Mode 3 Mode 7

```
glLogicOp(mode);
glEnable(GL_COLOR_LOGIC_OP);
```

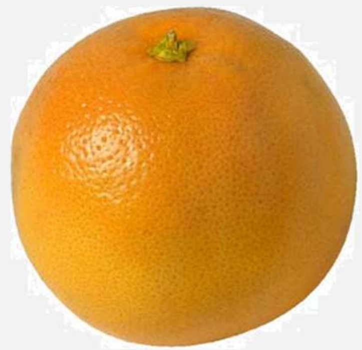
The Limits of Geometric Modeling

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

Image from "Final Fantasy"movie

Modeling an Orange

- Consider the problem of modeling an orange



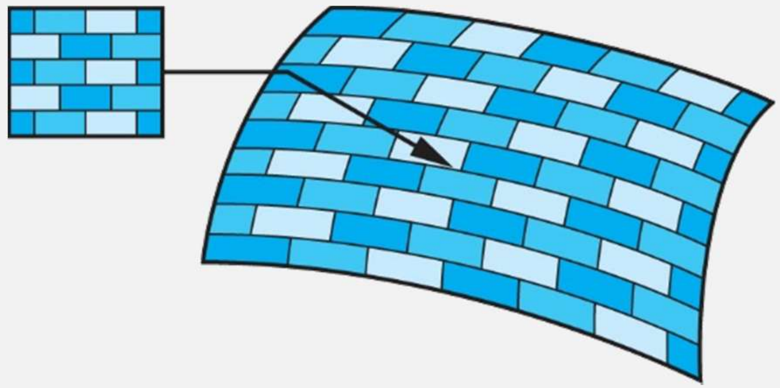
- Start with an orange-colored sphere
  - Too simple
- Replace sphere with a more complex shape
  - Does not capture surface characteristics (small dimples)
  - Takes too many polygons to model all the dimples

Mapping

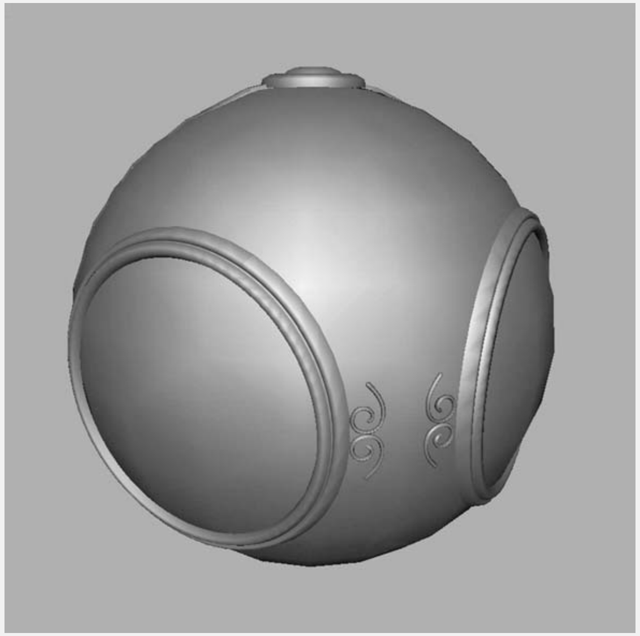
- Instead, we use mapping method that build a simple model and add details as part of the rendering process.
- Three major mapping methods:
  - Texture Mapping
  - Environment (Reflection) Mapping
  - Bump Mapping

Texture Mapping

- Use an image (or texture) to fill inside of polygons
  - Take a picture of a real orange, scan it, and "paste" onto simple geometric model
- Still might not be sufficient because resulting surface will be smooth
  - Need to change local shape
  - Bump mapping



Texture Mapping



geometric model

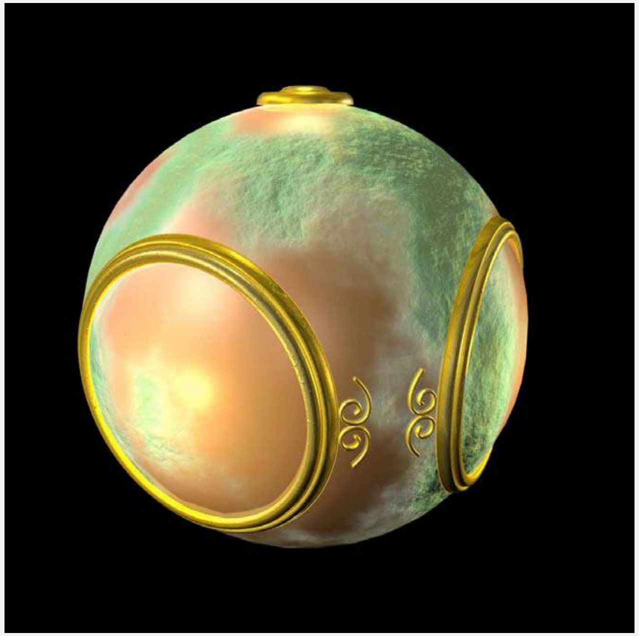
texture mapped

Environment (Reflection) Mapping

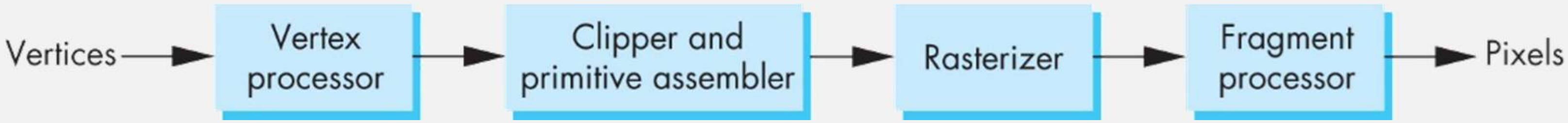
- Allows simulation of highly specular surfaces
- An image of the environment is painted onto the surface as that surface is being rendered
  - Create images that have the appearance of reflected materials without our having to trace reflected rays

Bump Mapping

- Distort the normal vectors during the rendering process to make the surface appear to have small variations in shapes

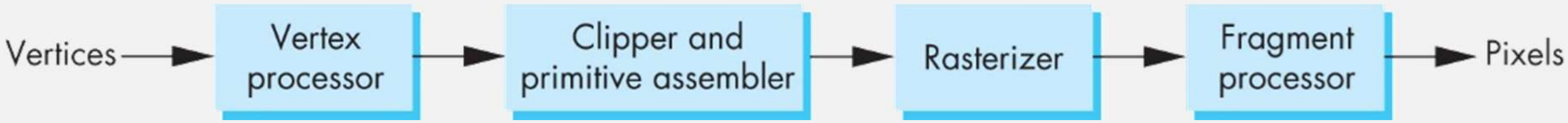


The Graphics Pipeline



- The graphics pipeline or rendering pipeline refers to the sequence of steps used to create a 2D raster representation of a 3D scene/model.
- Vertex processing
  - Each vertex is processed independently.
  - To carry out coordinate transformations.
    - Each change of the camera coordinate can be represented by a matrix.
  - To compute a color for each vertex.
- Clipper and Primitive Assembly
  - Efficient clipping must be done on a primitive-by-primitive basis rather than on a vertex-by-vertex basis.

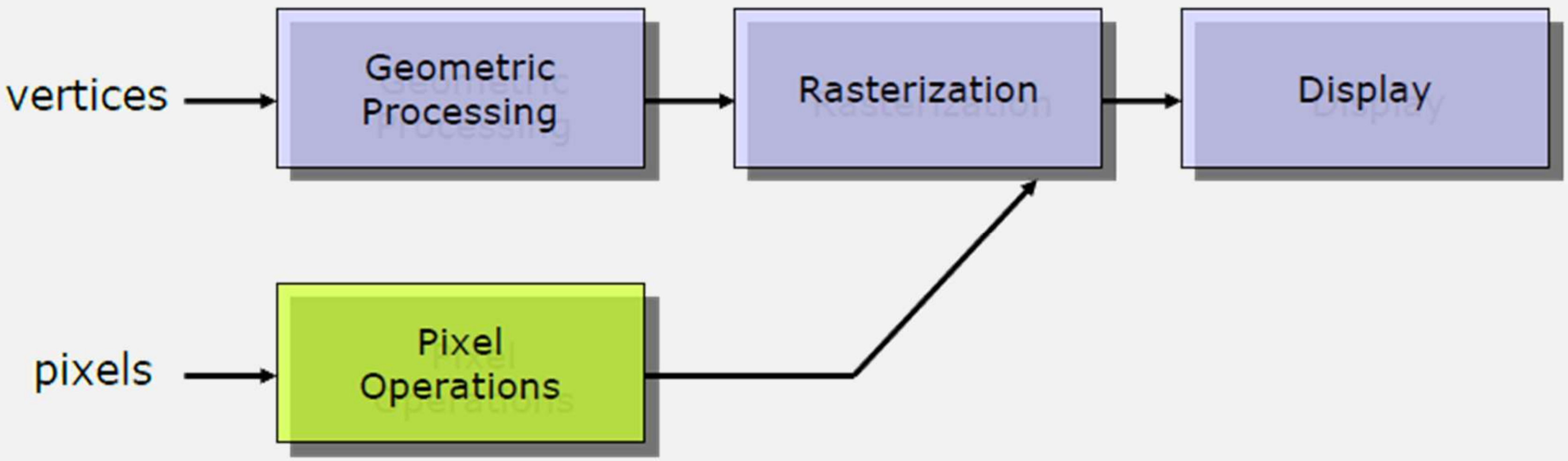
The Graphics Pipeline (Cont.)



- Rasterization (Scan conversion)
  - Primitives emerging from the clipper are still represented in terms of their vertices and must be converted to pixels in the frame buffer.
  - Determine which pixels in the frame buffer are inside the polygon.
  - Output of rasterization is a set of fragments (potential pixels with color, location, and depth information) for each primitive.
- Fragment Processing
  - Update the pixels in the frame buffer according to the processed fragments. (Some surfaces may not be visible because of occlusion)
  - The color of pixels in each fragment can be altered by texture mapping or bump mapping.

Where Does Mapping Take Place?

- Mapping techniques are implemented at the end of the rendering pipeline

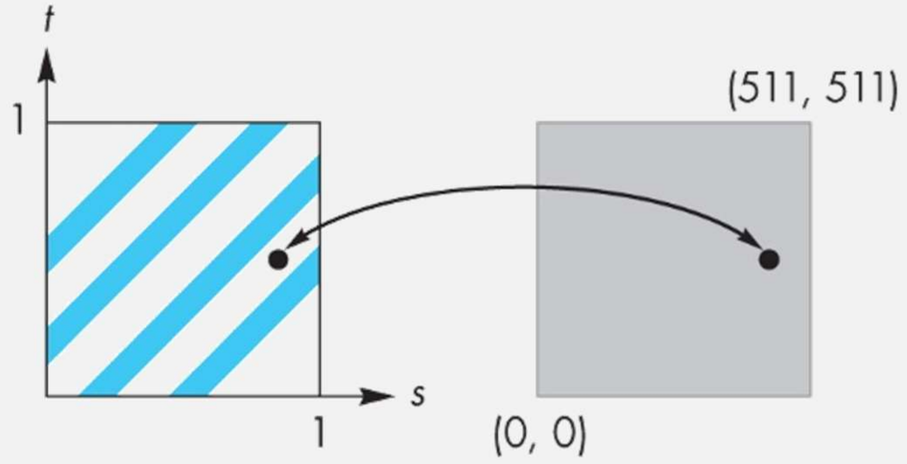


Two-Dimensional Texture Mapping in OpenGL

- Three basic steps:
  - 1st step: form a texture image and place it in texture memory on the GPU
    - glTexImage2D(Glenum target, Glint level, Glint iformat, Glsizei width, Glsizei height, Glint border, Glenum format, Glenum type, Glvoid *tarray); //specify a two-dimensional texture
      - target : choose a single image, set up a cube map, or test if there is sufficient texture memory
      - level: used for mipmapping, where 0 denotes the highest level or we are not using mimapping
      - iformat: specifies how we would like the texture stored in texture memory
      - width/height: specify the size of the image in the memory
      - format/type: describe how pixels in the image in processor memory are stored, so that OpenGL can read those pixels and store them in texture memory
  - 2nd step: assign texture coordinates to each fragment
  - 3rd step: apply the texture to each fragment
- Multiple ways to accomplish each step
  - Controlled by many parameters

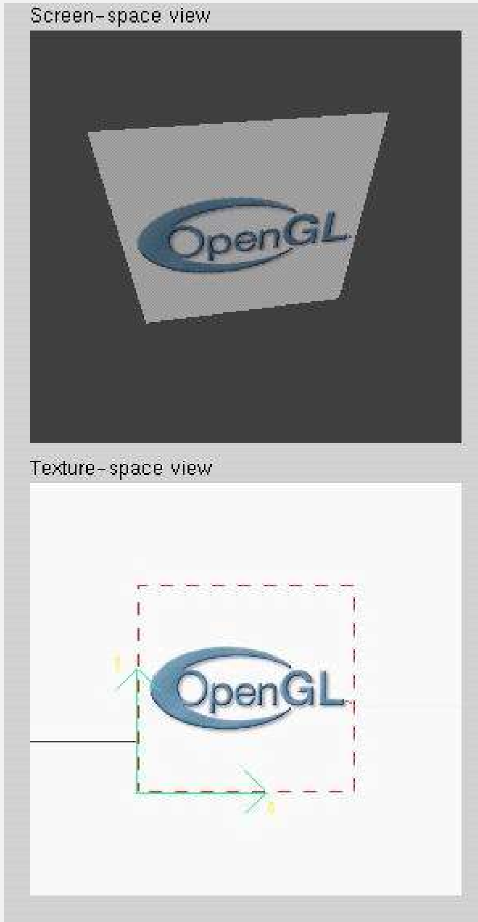
Two-Dimensional Texture Mapping in OpenGL (Cont.)

- Use two floating-point texture coordinates, s and t, rather than using integer texel locations that depend on the dimension of texture image



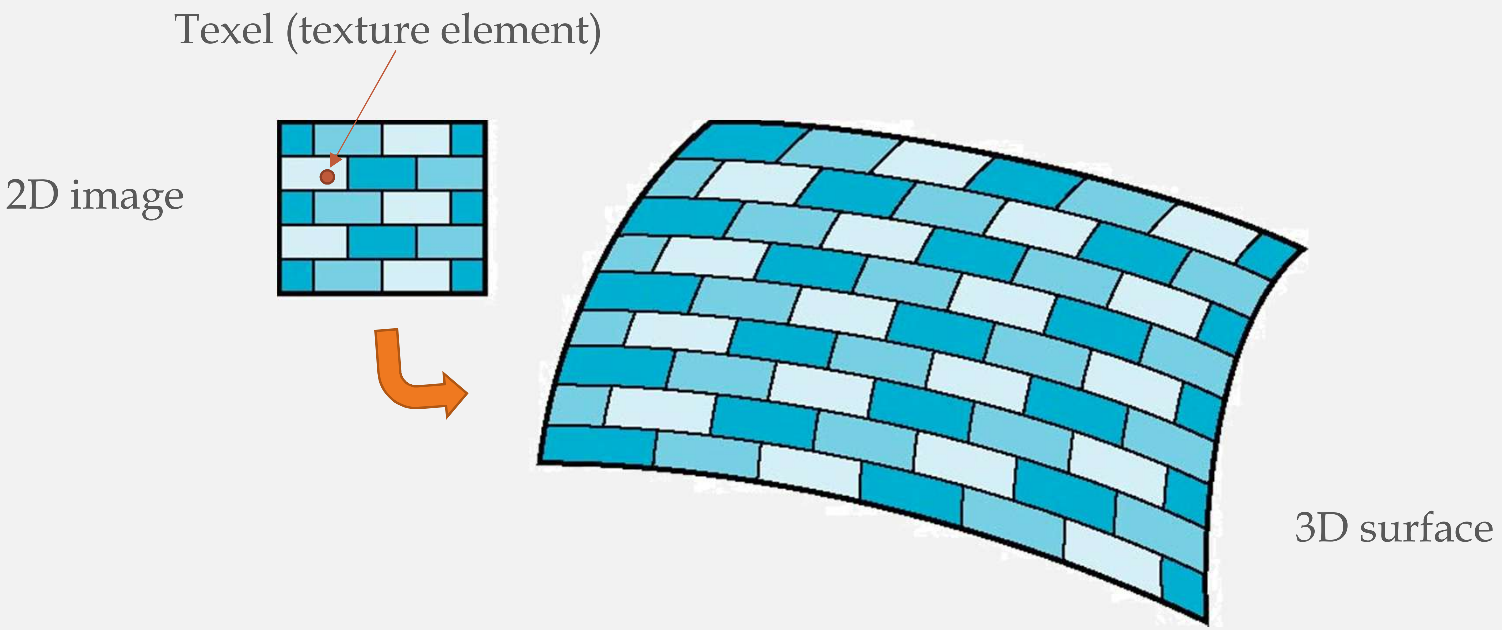
Texture Mapping Example

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



Is it Simple?

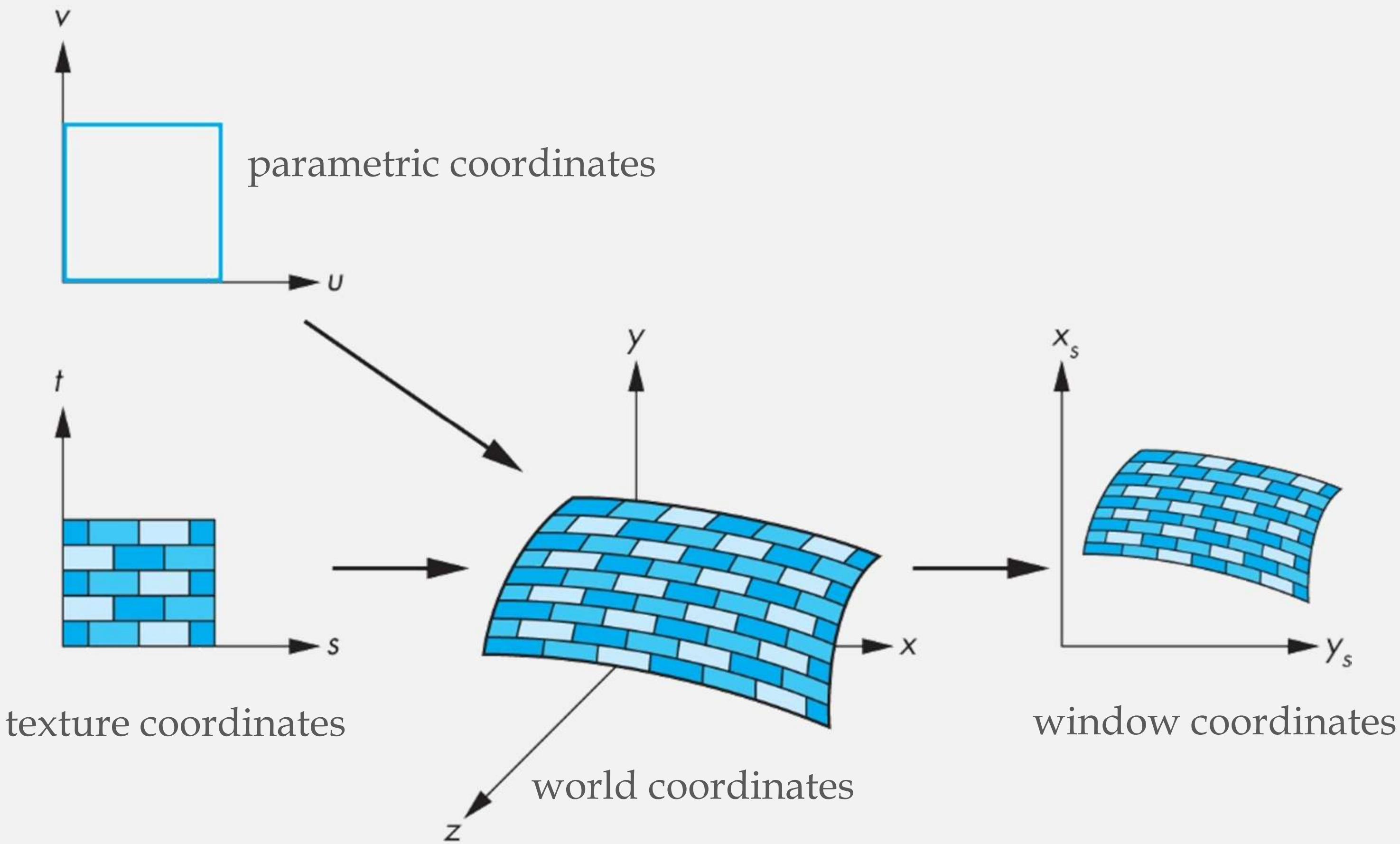
- Although the idea is simple
  - there are 3 or 4 coordinate systems involved in mapping an image to a surface



Coordinate Systems

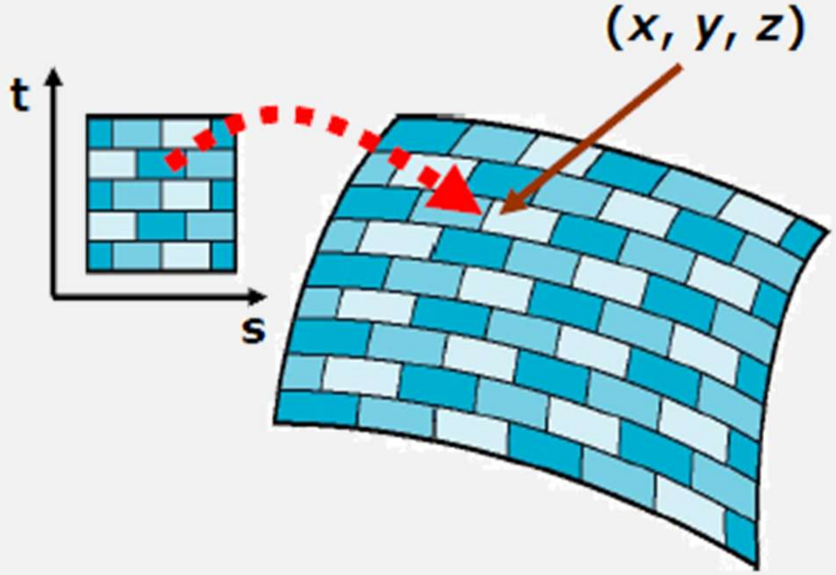
- Parametric coordinates
  - Used to model curves and surfaces
- Object or World Coordinates
  - Conceptually, where the mapping takes place
- Texture coordinates
  - Used to identify points in the image to be mapped
- Window Coordinates
  - Where the final image is really produced

Texture Mapping Involving 3 Mappings



Mapping Functions

- 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)
  - w=w(s,t)

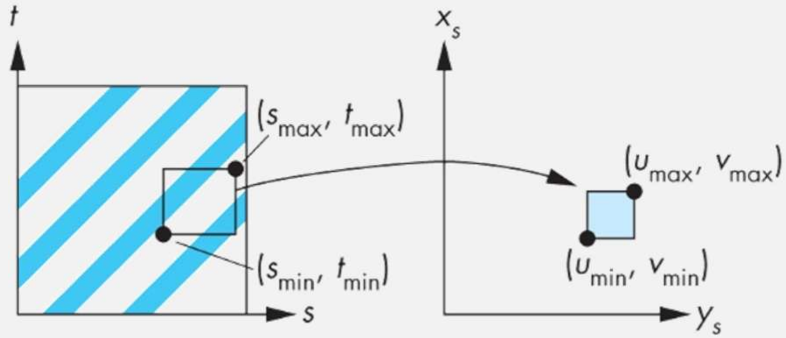


Backward Mapping

- Given a point (x, y, z) or (x, y, z, w) on an object, find the corresponding texture coordinates, i.e. the texel T(s,t)
  - s= s(x, y, z, w)
  - t= t(x, y, z, w)
- Such functions are difficult to find in general

Linear Map

- Map a point in the texture map T(s,t) to a point on the surface p(u,v) by a linear map



$$p(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$$

$$u = as + bt + c$$
$$v = ds + et + f$$

$$u = u_{min} + \frac{s - s_{min}}{s_{max} - s_{min}}(u_{max} - u_{min})$$

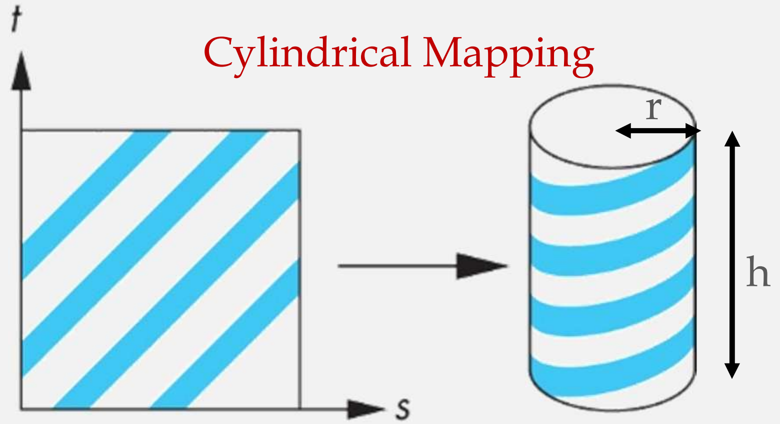
$$v = v_{min} + \frac{s - s_{min}}{s_{max} - s_{min}}(v_{max} - v_{min})$$

- Does not take into account the curvature of the surface

Two-part Mapping

- 1^{st} step: Map the texture to a simple intermediate surface
  - Example: map to cylinder, sphere, or cube

s = u
t = v



parametric cylinder:

$x = r \cos 2\pi u$

$y = r \sin 2\pi u$

$z = v/h$

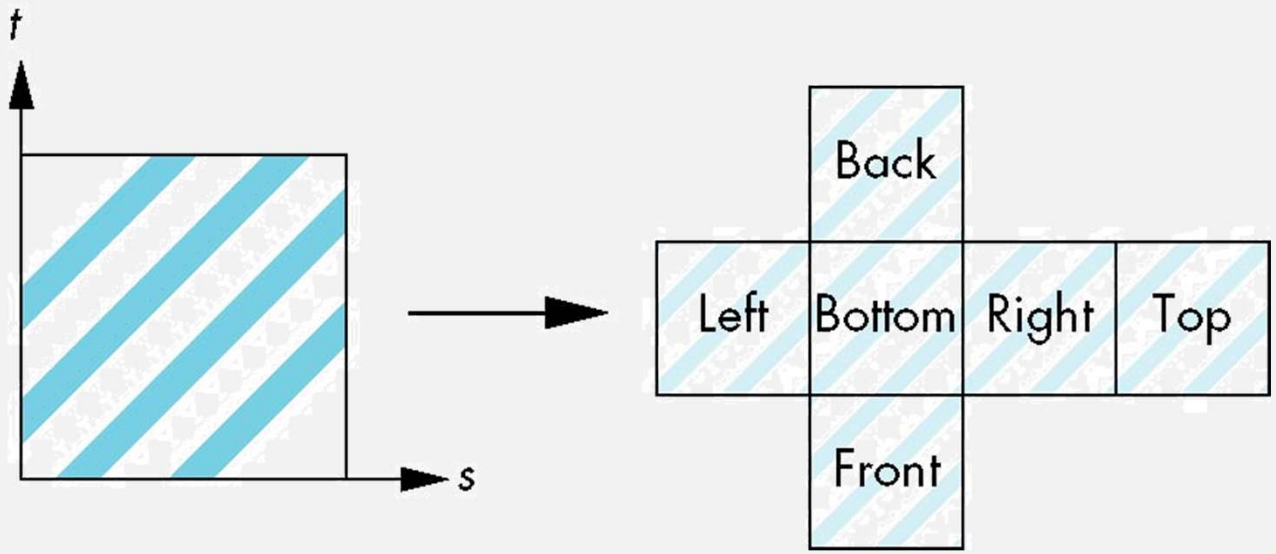
- 2^{nd} step: Map the intermediate surface to the surface being rendered

Spherical Map

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

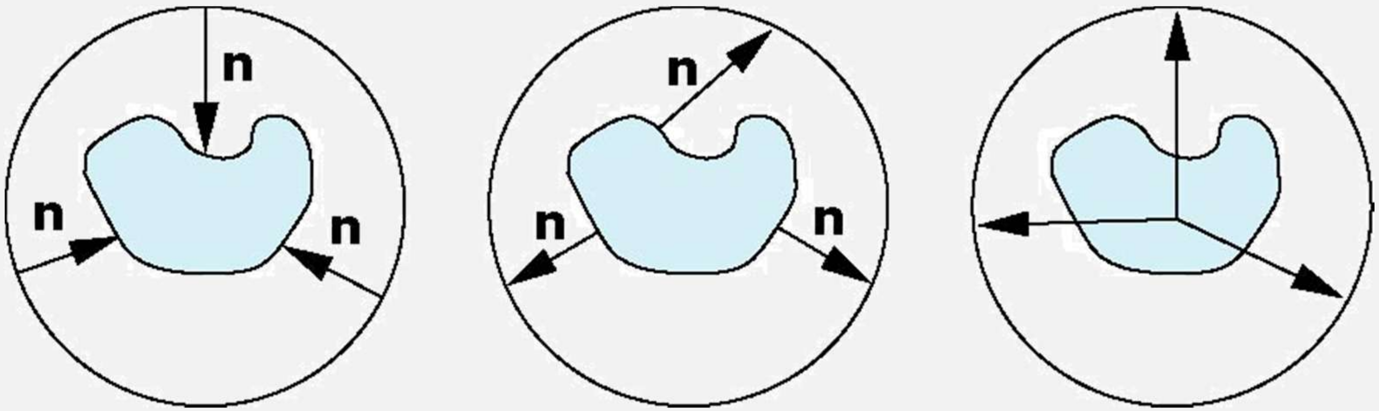
Box Mapping

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

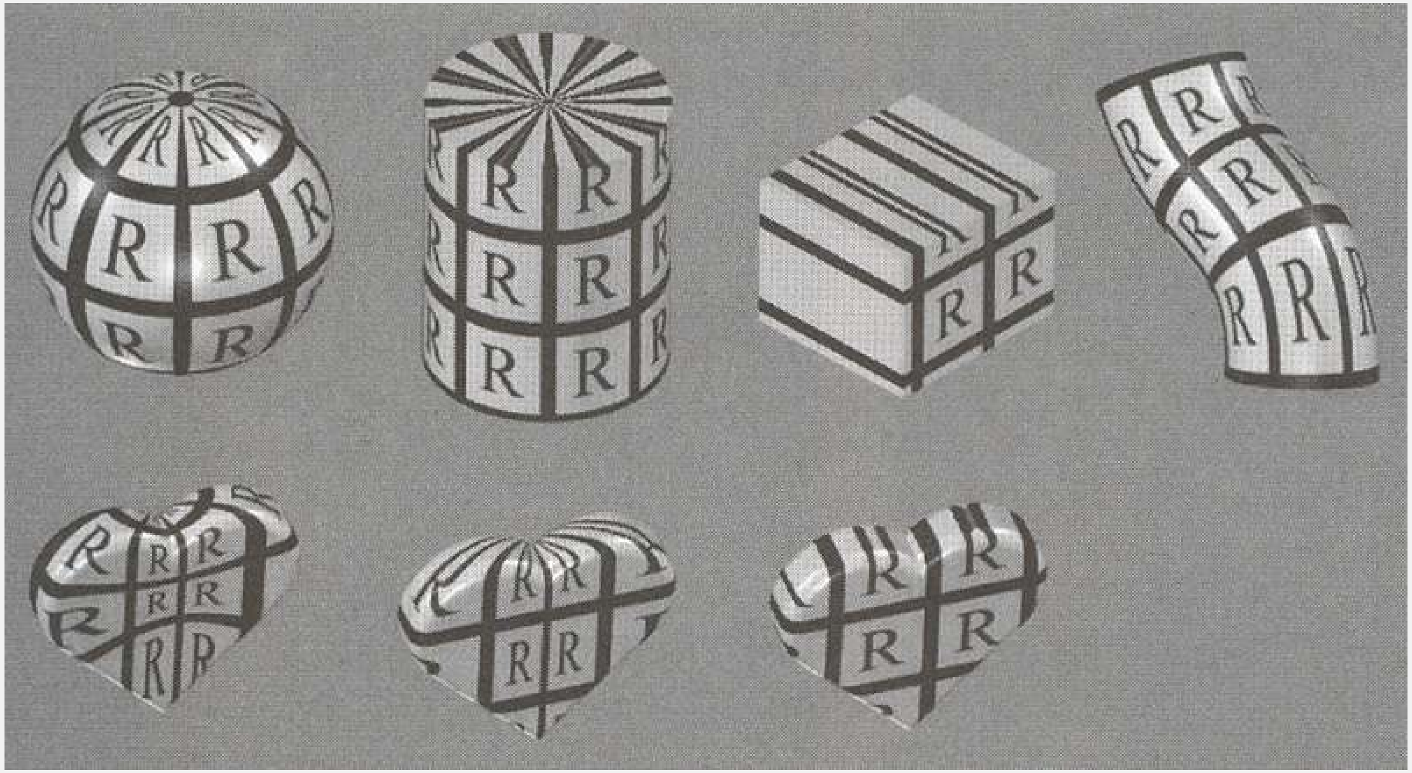


Second Mapping

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

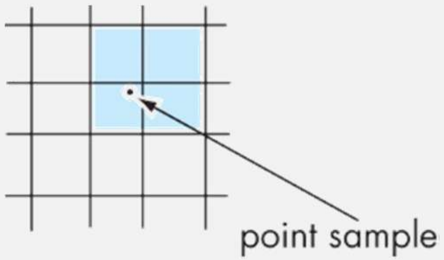


Two-part Mapping



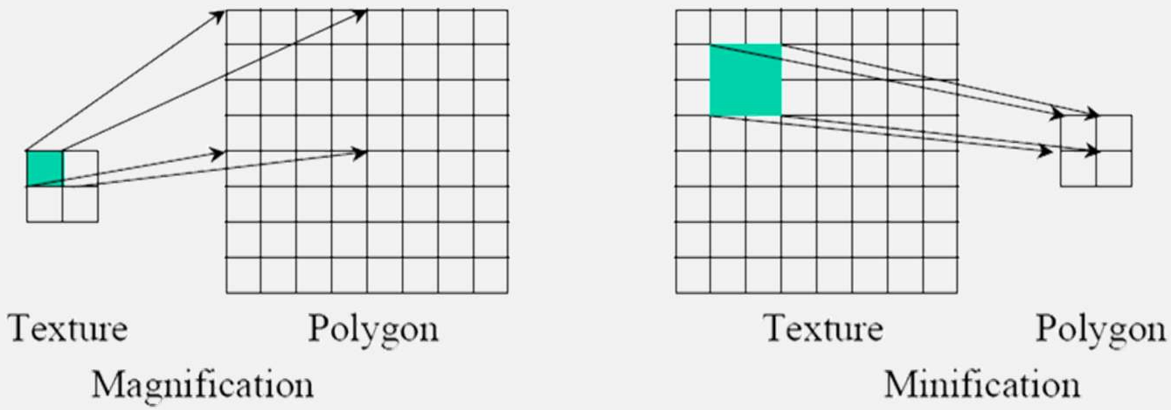
Texture Sampling

- How to assign a texture value to a pixel ?
  - Point Sampling: use the value of the texel that is closest to the texture coordinate output by the rasterizer
  - Linear Filtering: weighted averaging the neighborhood texels of the texel determined by point sampling
    - A better strategy
    - More difficult to implement (how to deal with the boundary of the texel array)
    - Still imperfect due to the limited resolution of both the frame buffer and the texture map



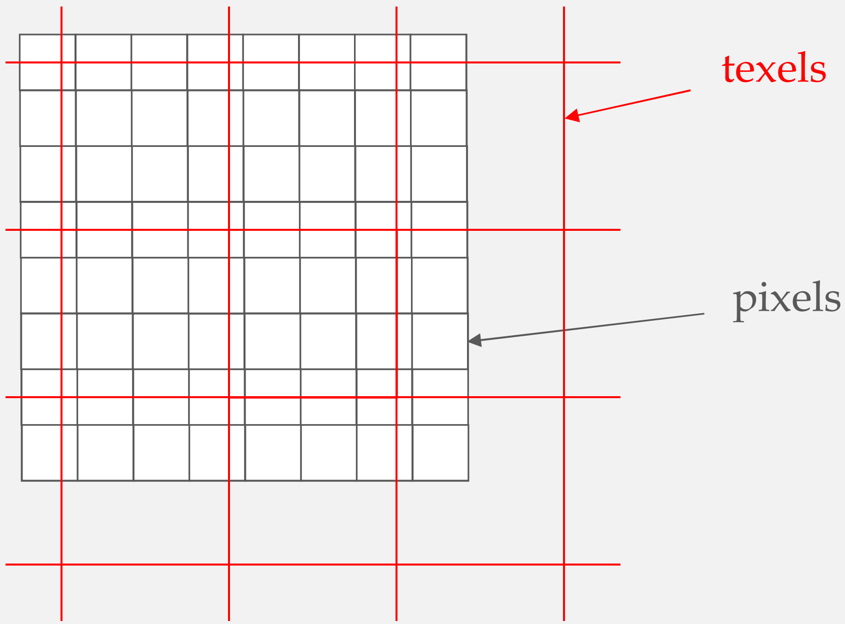
Magnification and Minification

- The size of the pixel on the screen may be smaller or larger than one texel
  - Magnification
    - The texel covers multiple pixels
  - Minification
    - The pixel covers multiple texels



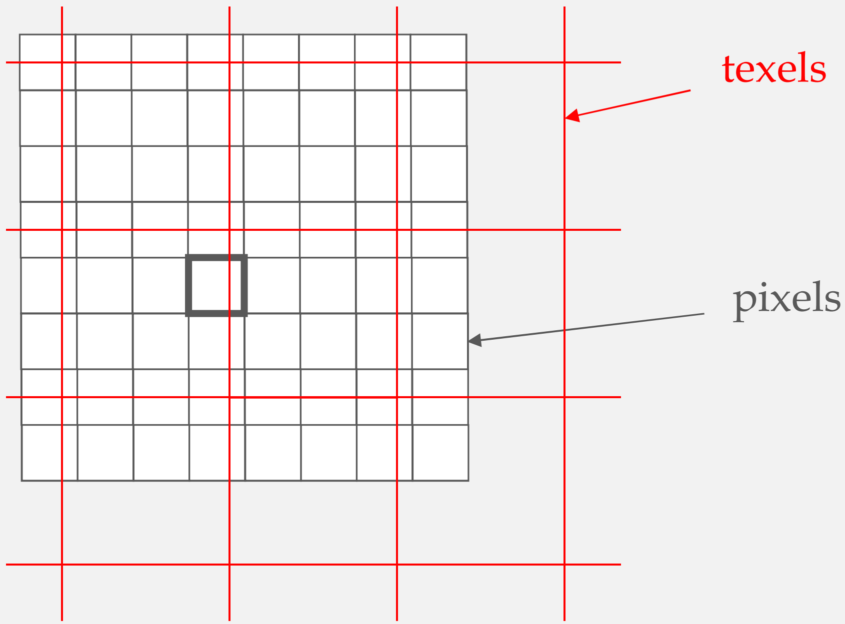
Magnification

- The alignment is probably not exact.



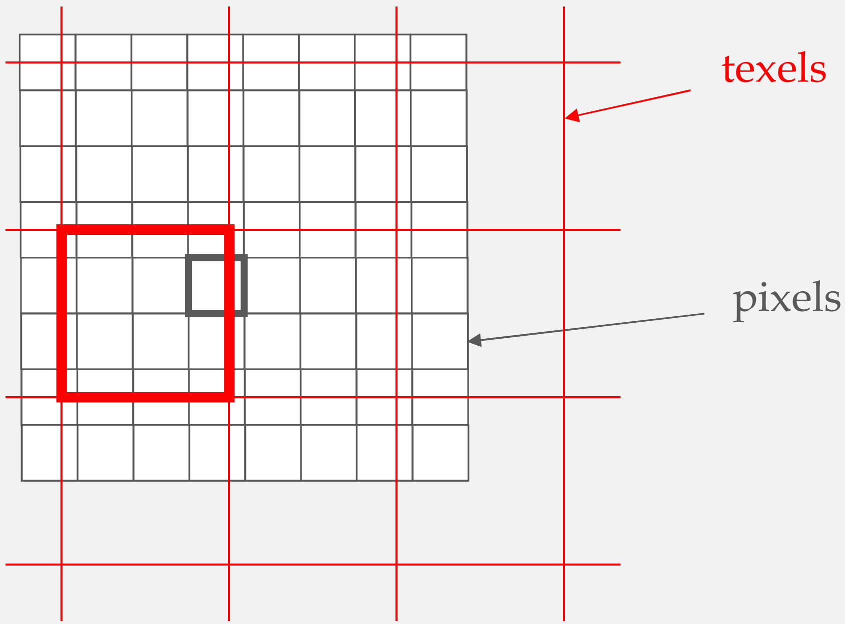
Nearest Texel

- Find the nearest texel.



Nearest Texel

- Find the nearest texel.

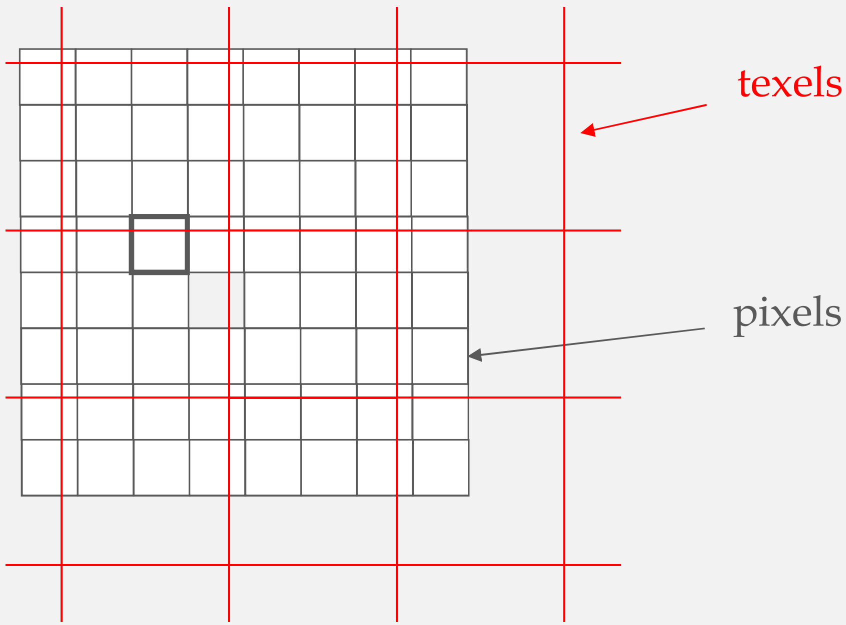


Linear Interpolation

- OpenGL may also interpolate the colors of the nearest four texels.

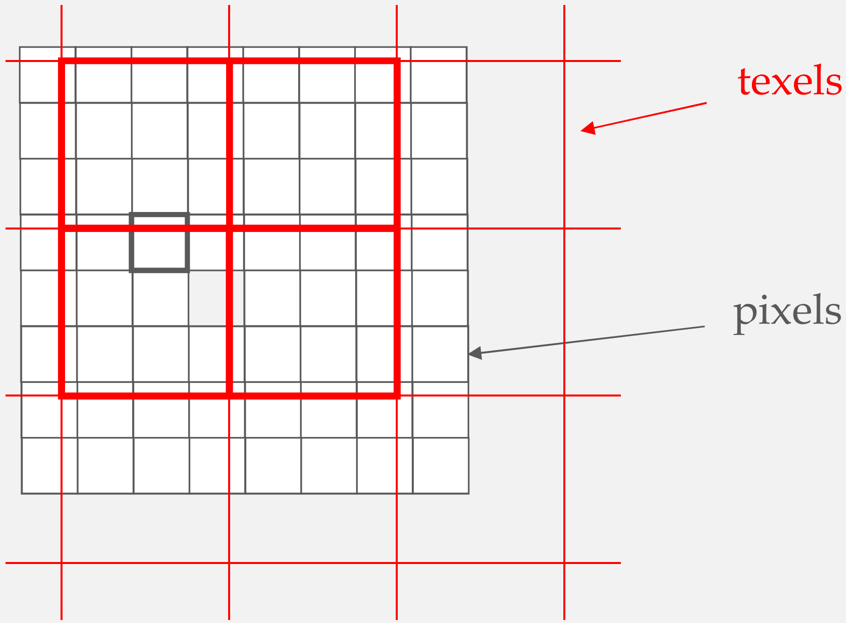
Linear Interpolation

- Find the nearest four texels.



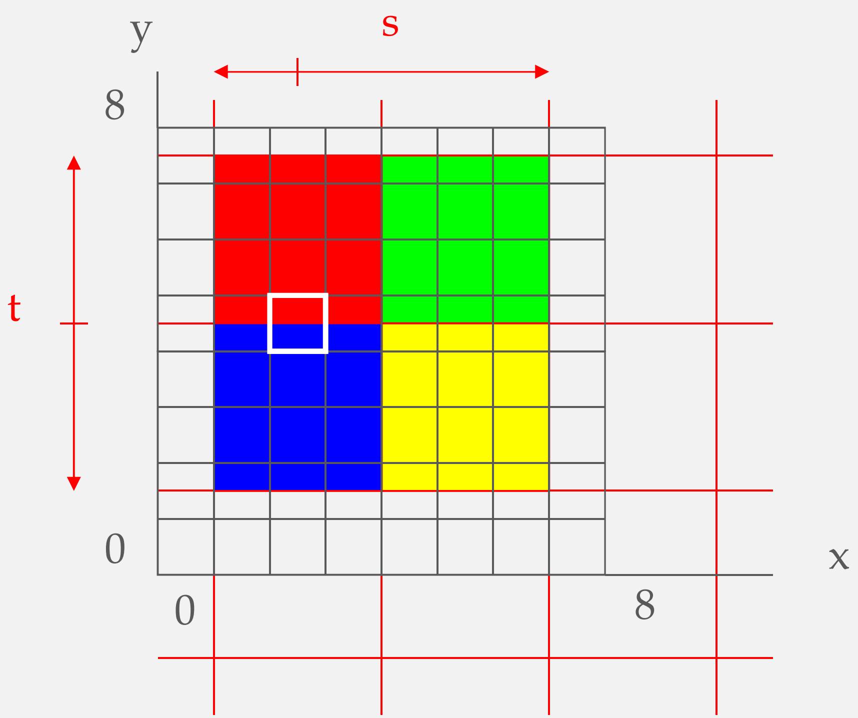
Linear Interpolation

- Find the nearest four texels.



Example: Interpolation

- Using the nearest texel, color the pixels.



Example: Interpolation

- Compute the color of the pixel (2, 4).
- Assume the texture is 2×2.
- The center of the pixel is
  - 25% of the way across the group of texels.
  - Therefore, $s = 0.25$.
  - 50% of the way up the group of texels.
  - Therefore, $t = 0.50$.

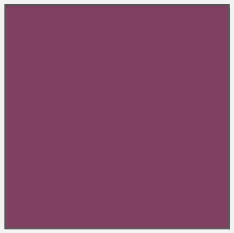
Example: Interpolation

- Interpolate horizontally:
  - Top edge:

  0.75(1, 0, 0) + 0.25(0, 1, 0) = (0.75, 0.25, 0).
  - Bottom edge:

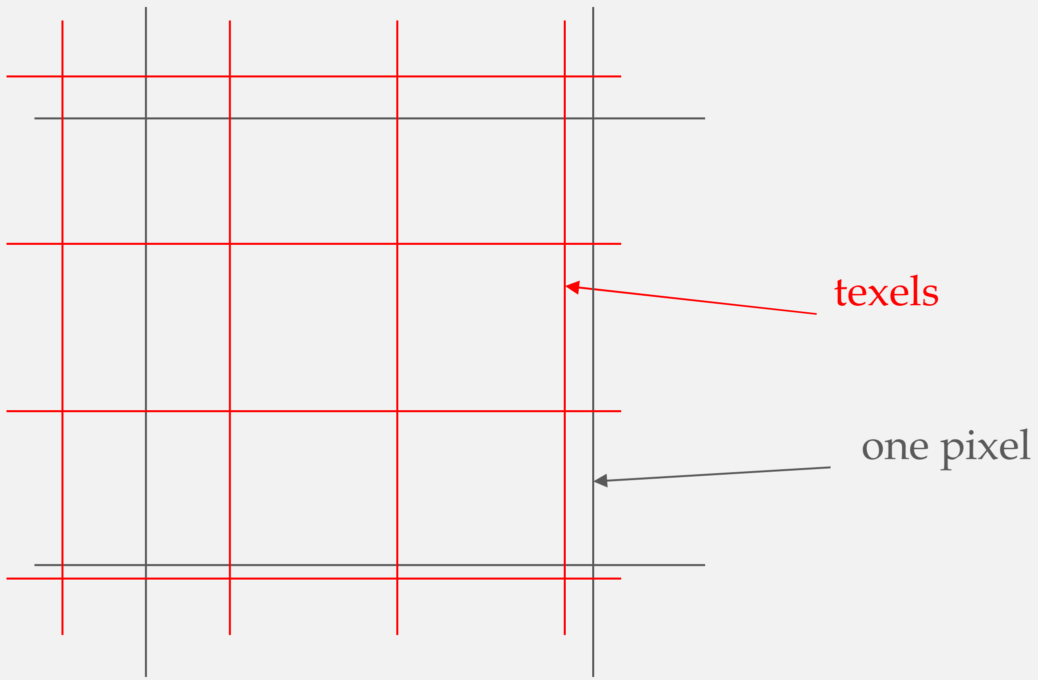
  0.75(0, 0, 1) + 0.25(1, 1, 0) = (0.25, 0.25, 0.75).
- Now interpolate those values vertically:
  - 0.5(0.75, 0.25, 0) + 0.5(0.25, 0.25, 0.75)

  = (0.5, 0.25, 0.375).



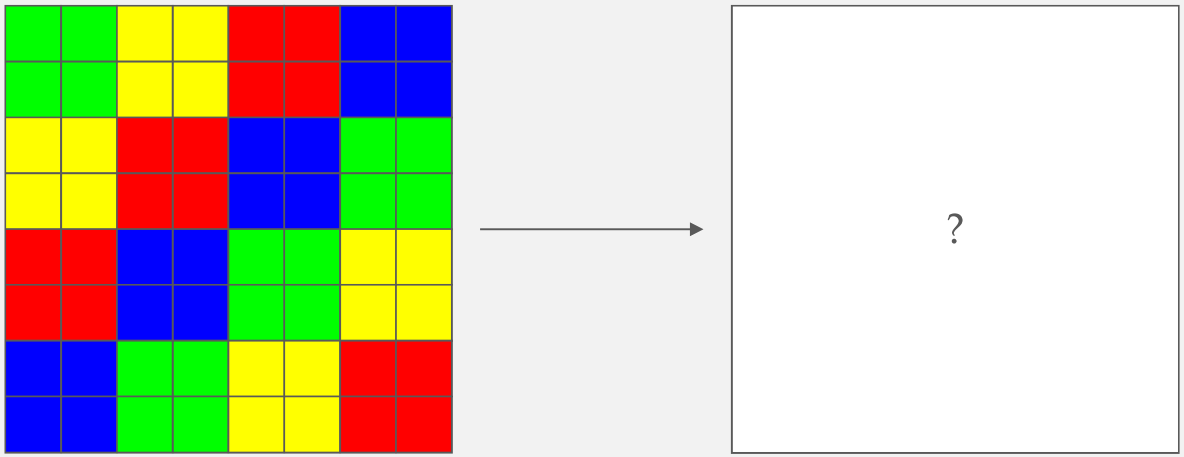
Minification

- Again, the alignment is not exact.



Minification

- If 64 texels all map to the single pixel, what color should the pixel be?

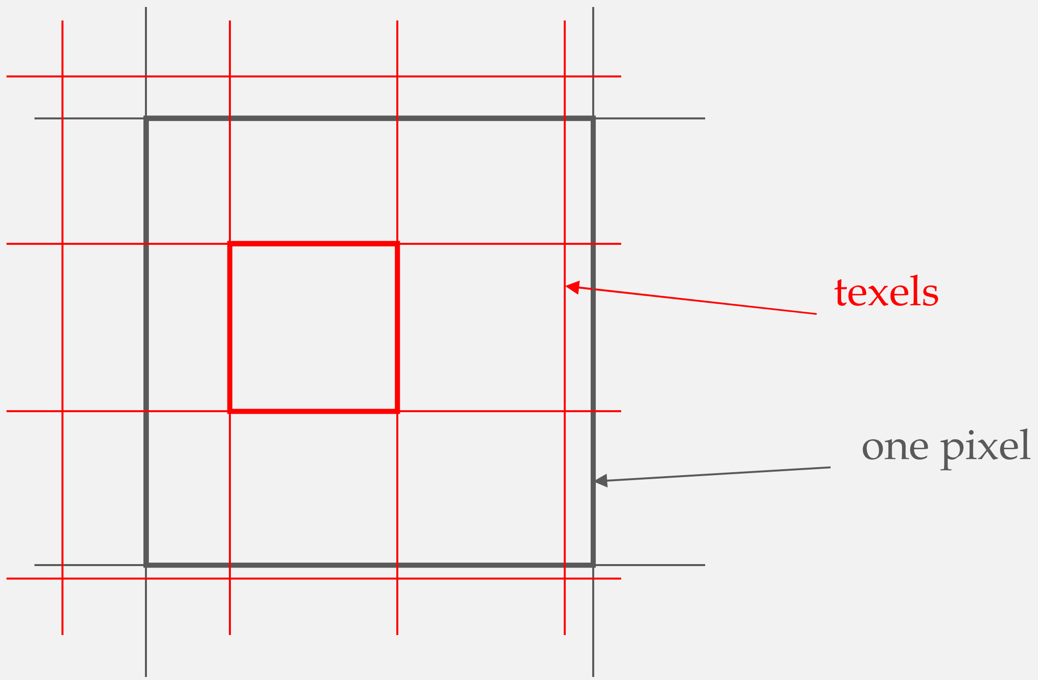


Minification

- Again, we may choose between the nearest texel and interpolating among the nearest four texels.

Minification

- Choose the nearest texel.

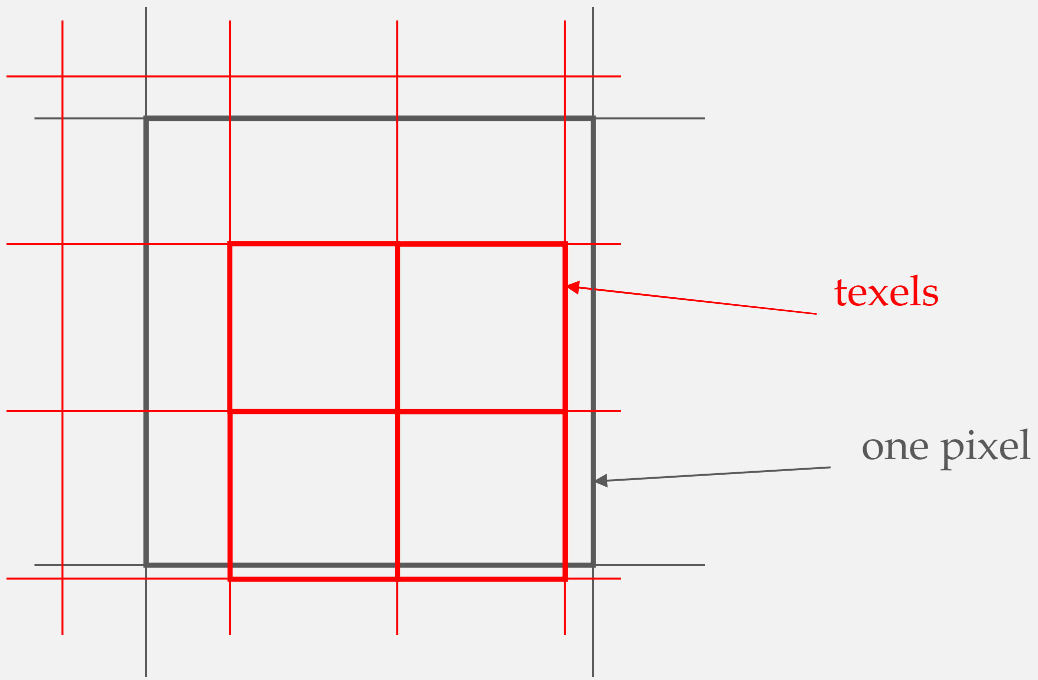


Minification

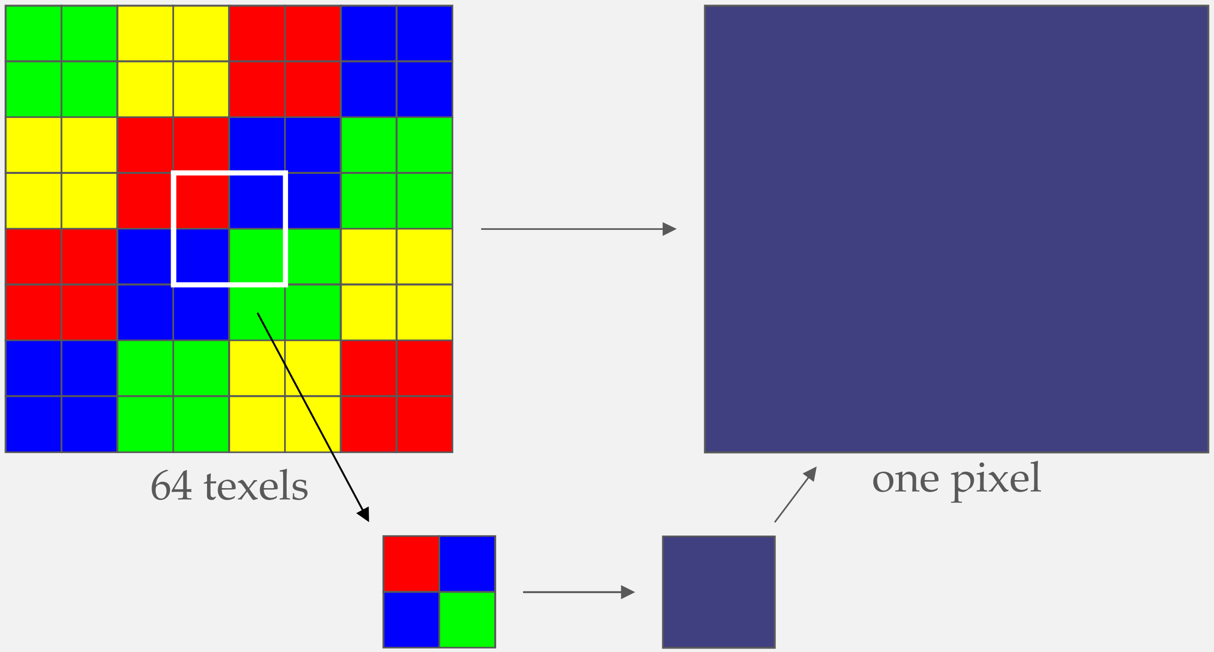
- If we choose to interpolate, then OpenGL will compute the average of the four texels that whose centers are *nearest* to the center of the pixel.
- This will reduce, but not eliminate, the aliasing or effect.

Minification

- Choose the four nearest texels.

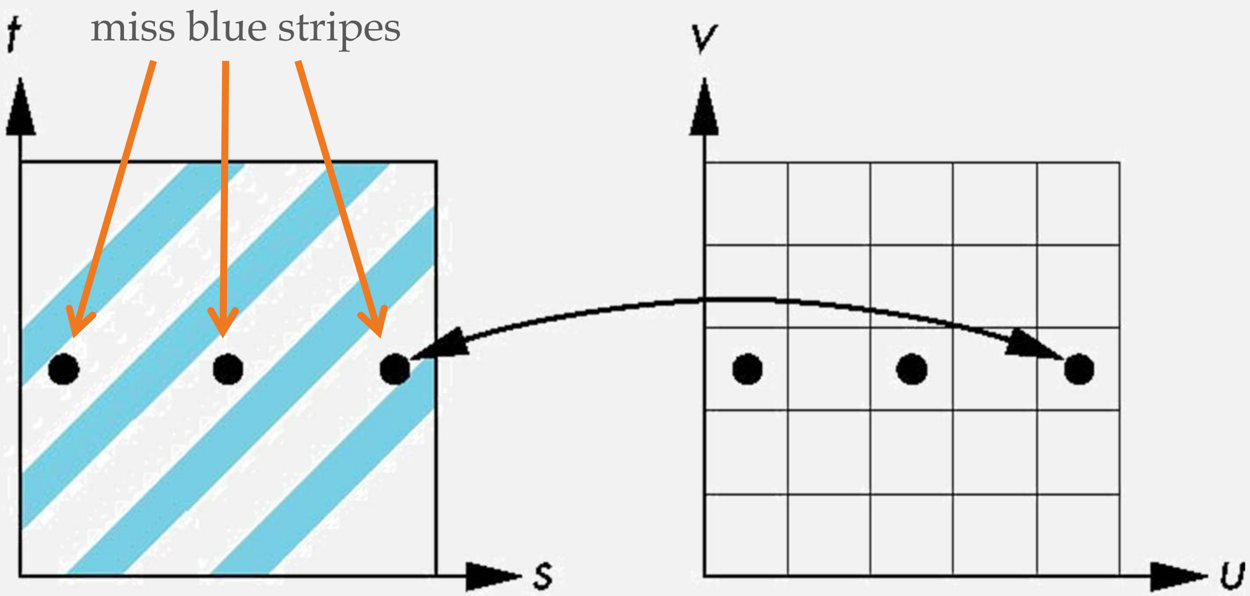


Minification



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

Aliasing Example

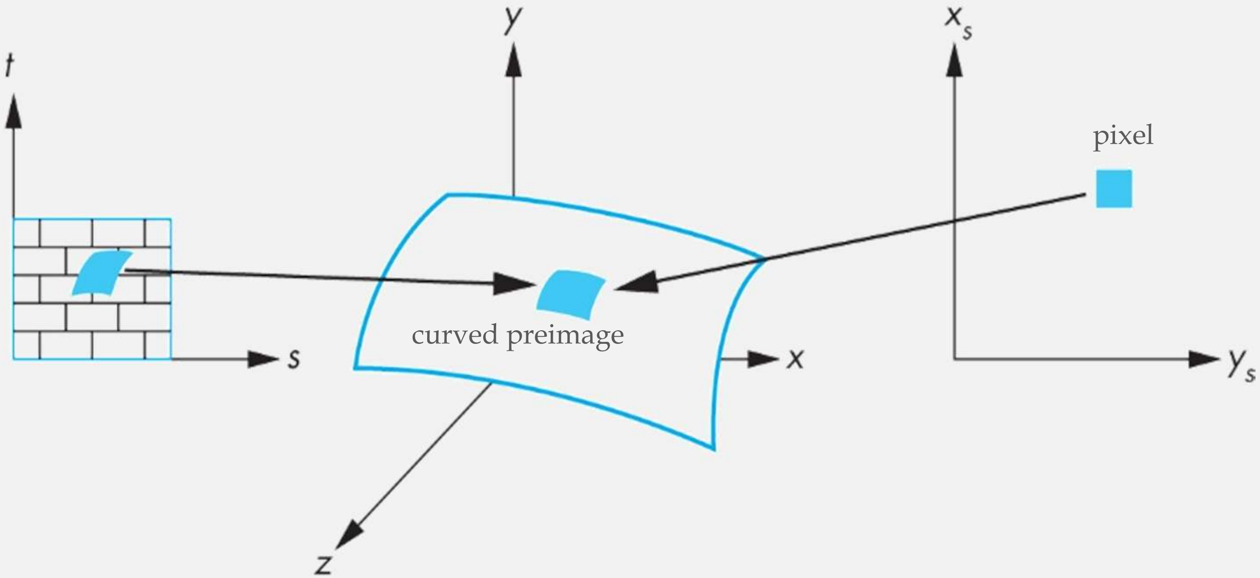
Original image

Sample one for each 5x5 pixels

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

Area Averaging

- A better but slower option is to use area averaging



Area Averaging

Original image

Sampling every 5x5 pixels

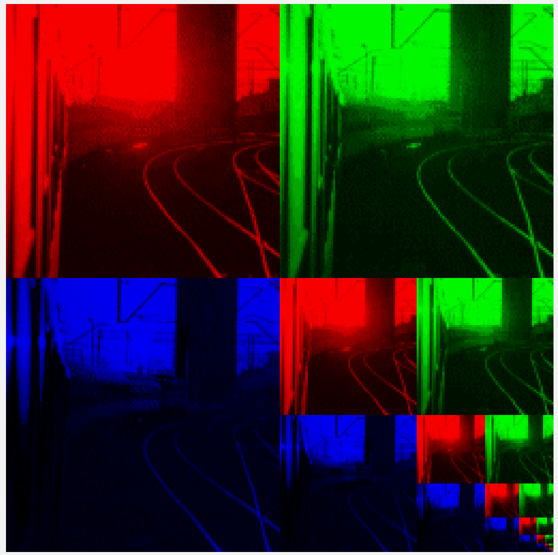
Applying a 5x5 box filter

Sampling every 5x5 pixels

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

Mipmapped Textures

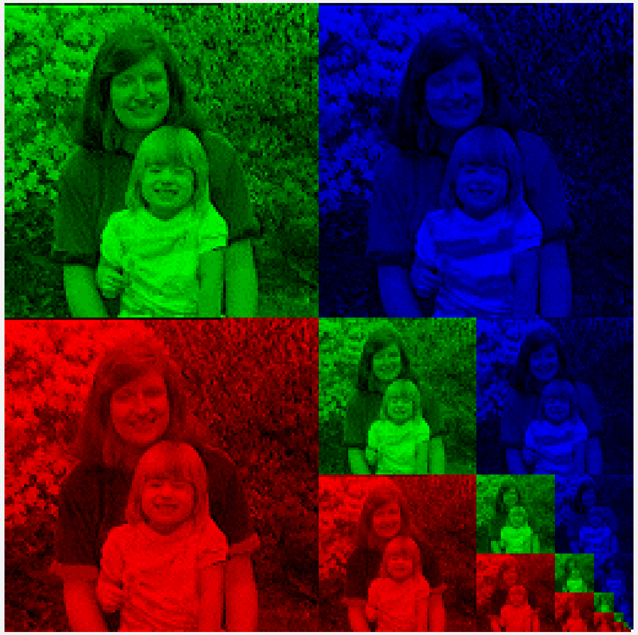
- OpenGL has another way to deal with the minification problem:
  - Mipmapping: create a series of texure arrays at reduced sizes
    - glGenerateMipmap(GL_TEXTURE_2D);

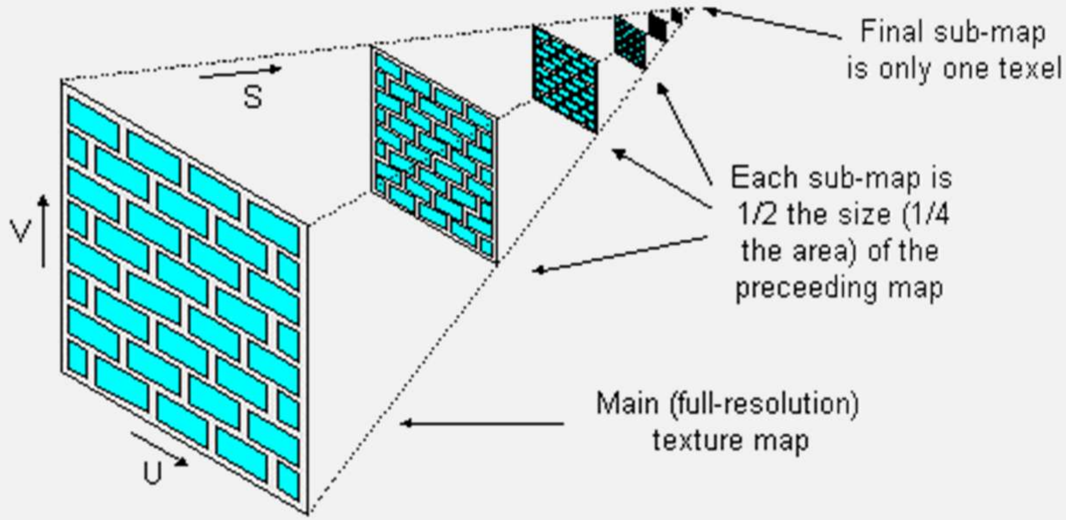


storage

Mipmapping

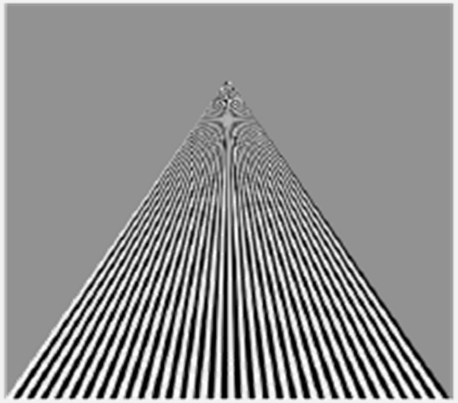
- 1/3 overhead of maintaining the MIP map.





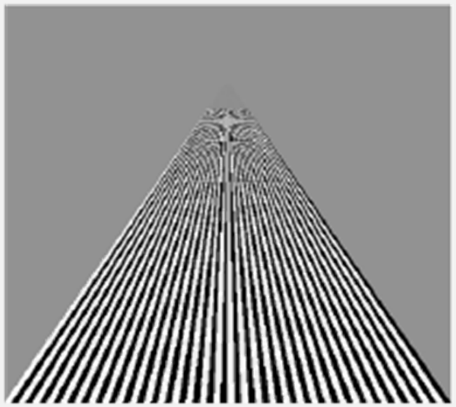
Example

point sampling



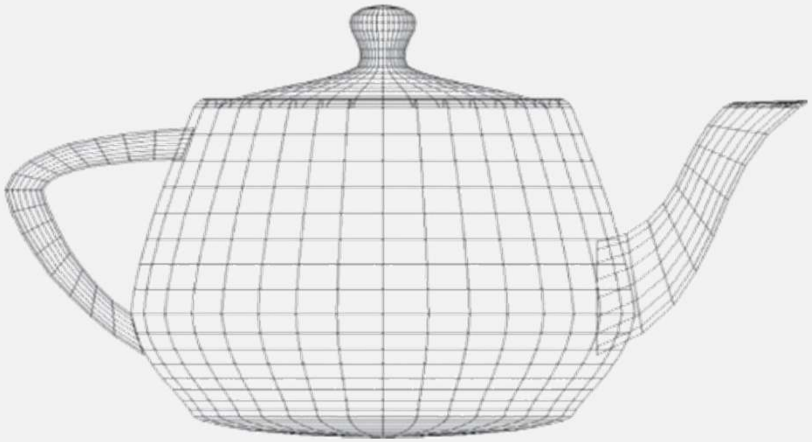
linear filtering

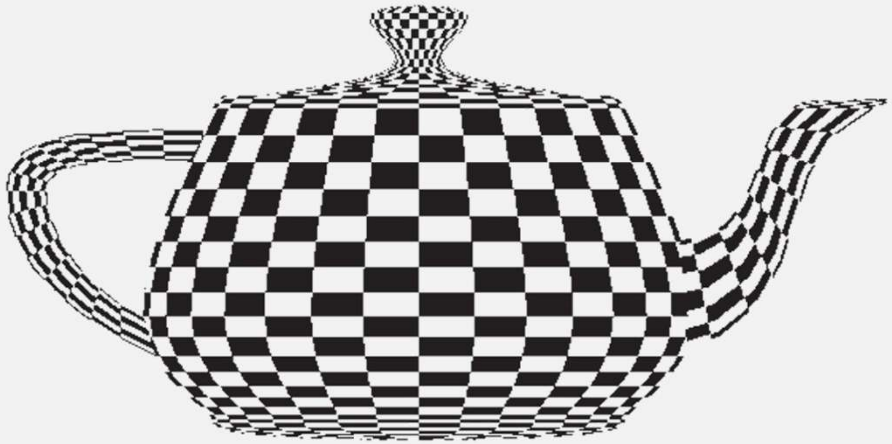
mipmapped
point sampling



mipmapped
linear filtering

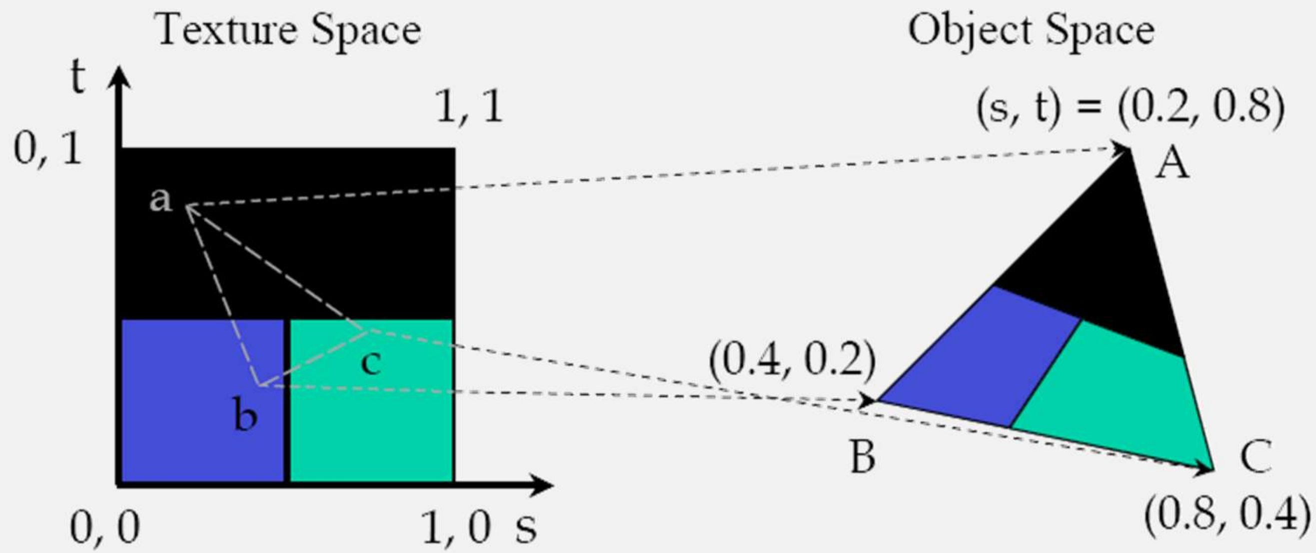
Mesh Size Problem





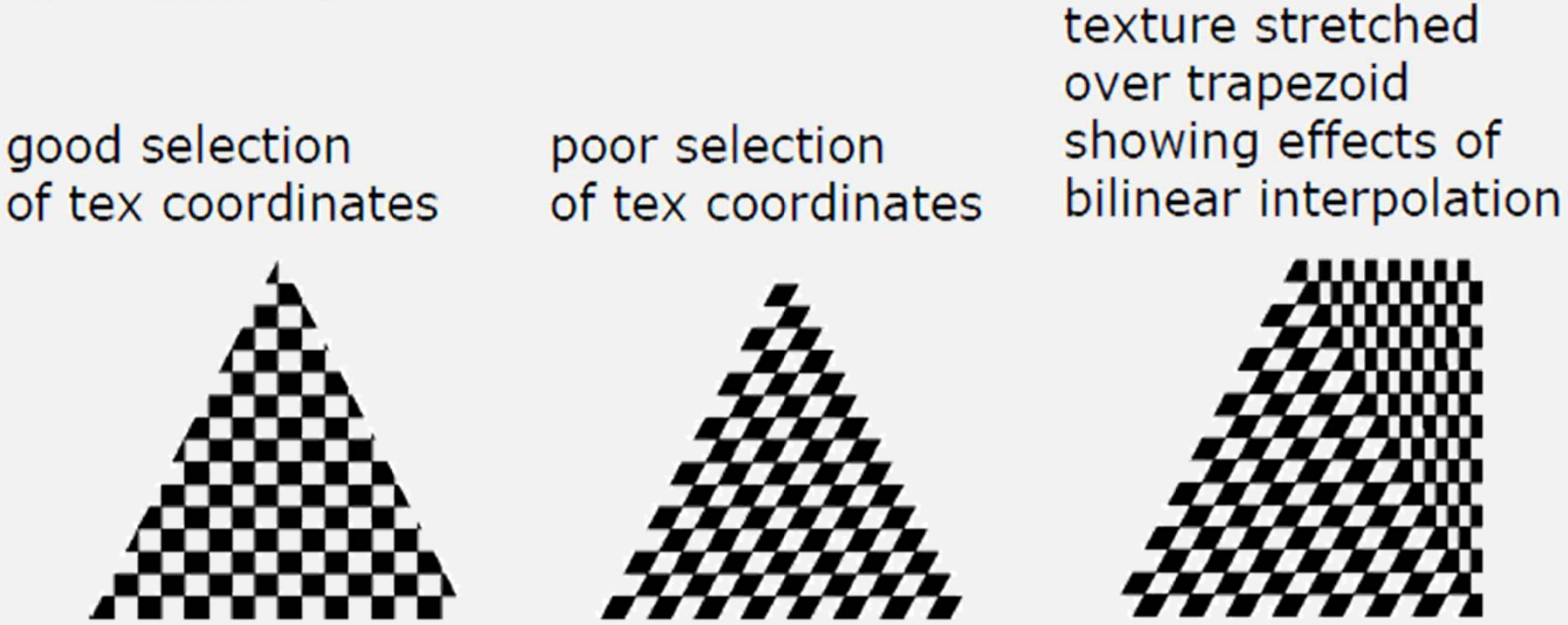
Texture Mapping for Polygons in OpenGL

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



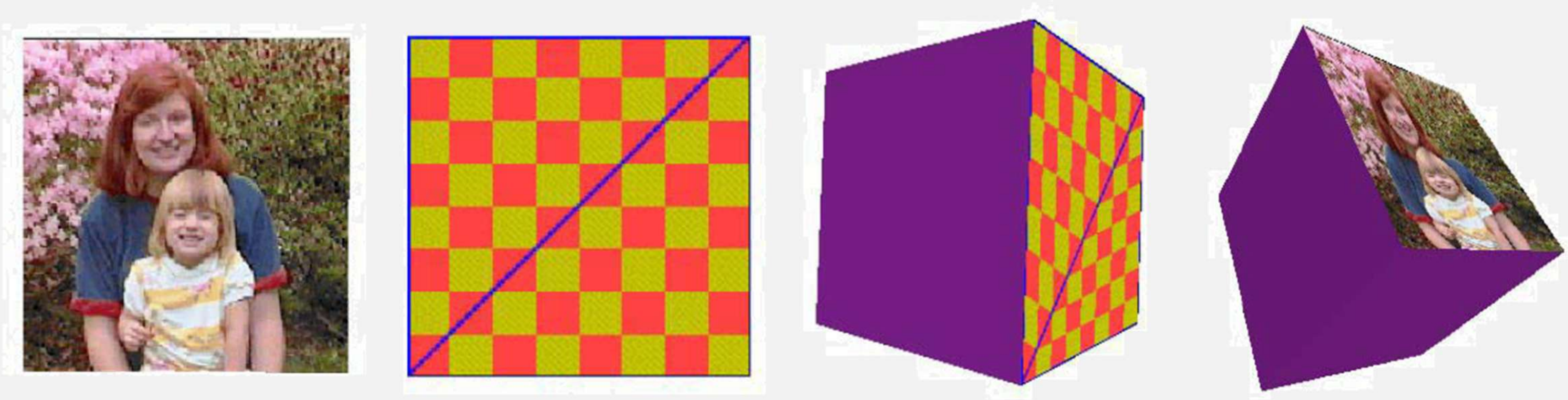
Interpolation

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



Interpolation (Cont.)

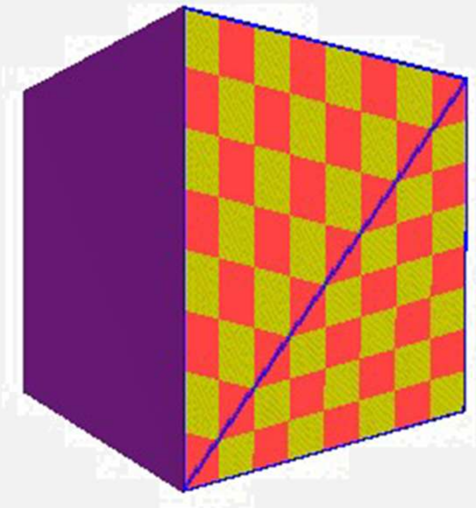
- Can we just use Linear interpolation in screen space?

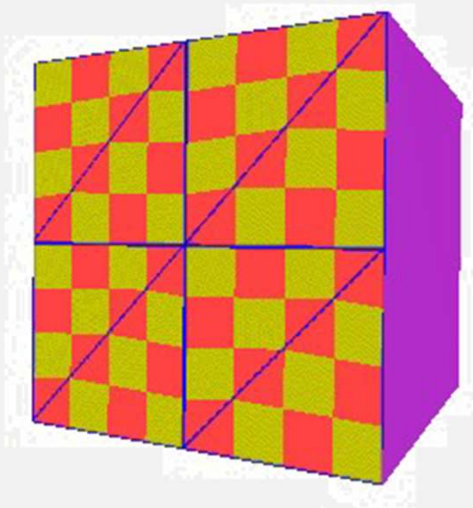


Pictures from lecture notes of “Computer Graphics”, UNC

Reduction of the flaws

- Subdivide the texture-mapped triangles into smaller triangles.



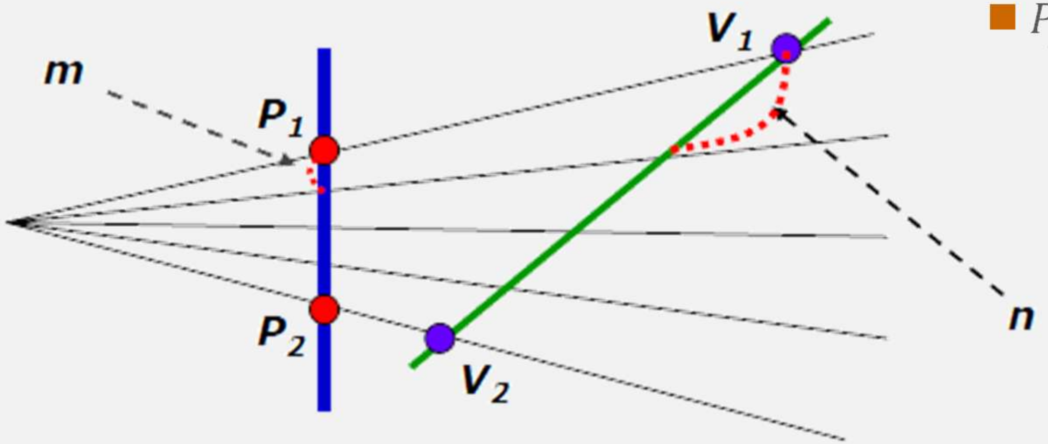


- Is it correct?

Mapping from Screen Space to 3D Space

- Interpolation in screen space
  - $P(m) = P_1 + m(P_2 - P_1)$

- Interpolation in 3D space
  - $V(n) = V_1 + n(V_2 - V_1)$
  - $P_y(n) = V_y(n) \ / \ V_z(n)$



$$P_y(m) = \frac{y_1}{z_1} + m\left(\frac{y_2}{z_2} - \frac{y_1}{z_1}\right) = \frac{y_1+n(y_2-y_1)}{z_1+n(z_2-z_1)} \Rightarrow n = \frac{mz_1}{z_2 + m(z_1 - z_2)}$$

Perspective Correct Interpolation

- $T(n) = T_1 + n(T_2 - T_1)$
- Assume $w_1 = \frac{1}{z_1}, w_2 = \frac{1}{z_2}$ (for the graphics pipeline)

$$I = I_1 + \frac{mz_1}{z_2 + m(z_1 - z_2)}(I_2 - I_1)$$

$$= I_1 + \frac{mw_2}{w_1 + m(w_2 - w_1)}(I_2 - I_1)$$

$$= \frac{I_1 w_1 + m(I_2 w_2 - I_1 w_1)}{w_1 + m(w_2 - w_1)}$$

How to Handle Highly Specular Surfaces?

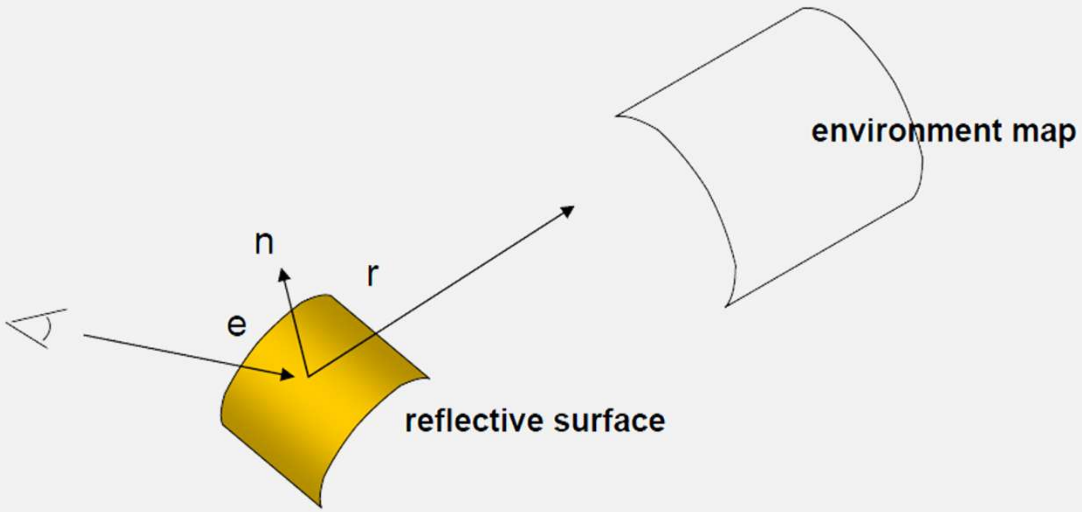
- How to render a flat mirror?
- How to render a mirror-like object in a virtual scene?

Environment Mapping

- Also known as reflection mapping
- First proposed by Blinnand Newell.
- An efficient way to create reflections on curved surfaces
  - can be implemented using texture mapping supported by graphics hardware

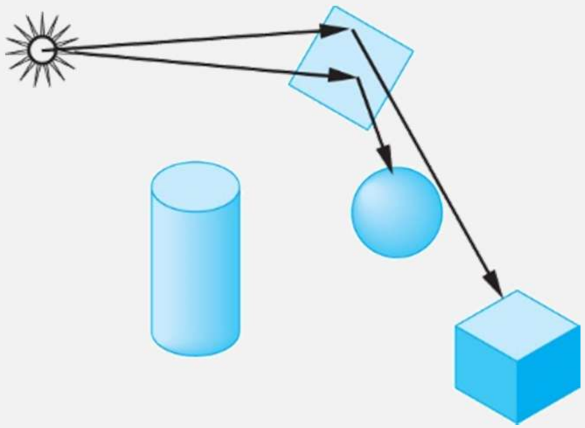
Environment Mapping (Cont.)

- 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.



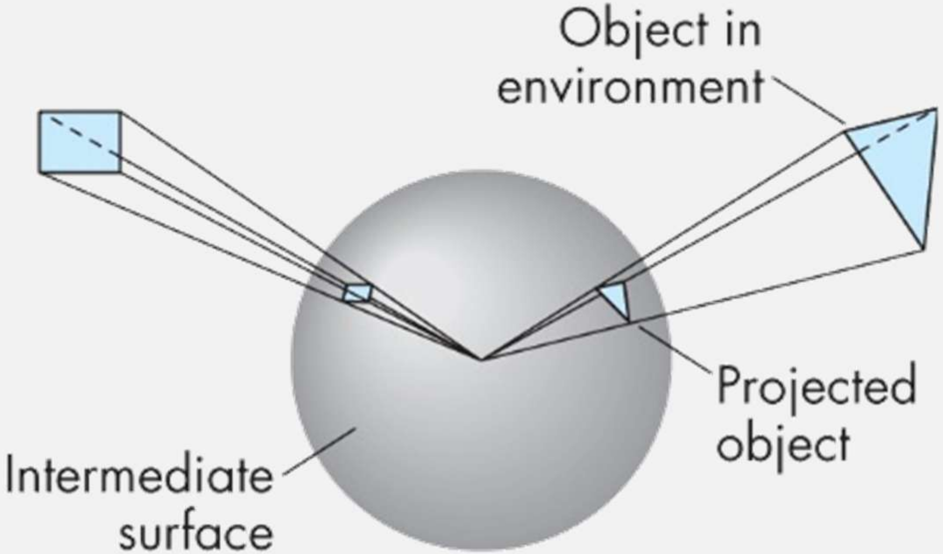
Environment Mapping (Cont.)

- We can obtained an approximately correct value of the shade as part of a two-step rendering pass
  - 1st step: render the scene without mirror polygon, with the camera placed at the center of the mirror pointed in the direction of the normal of the mirror
    - Thus, we obtained an image of objects seen by the mirror
  - 2nd step: render the scene with the mirror polygon, and use the obtained image as the shade (texture) to place on the mirror polygon

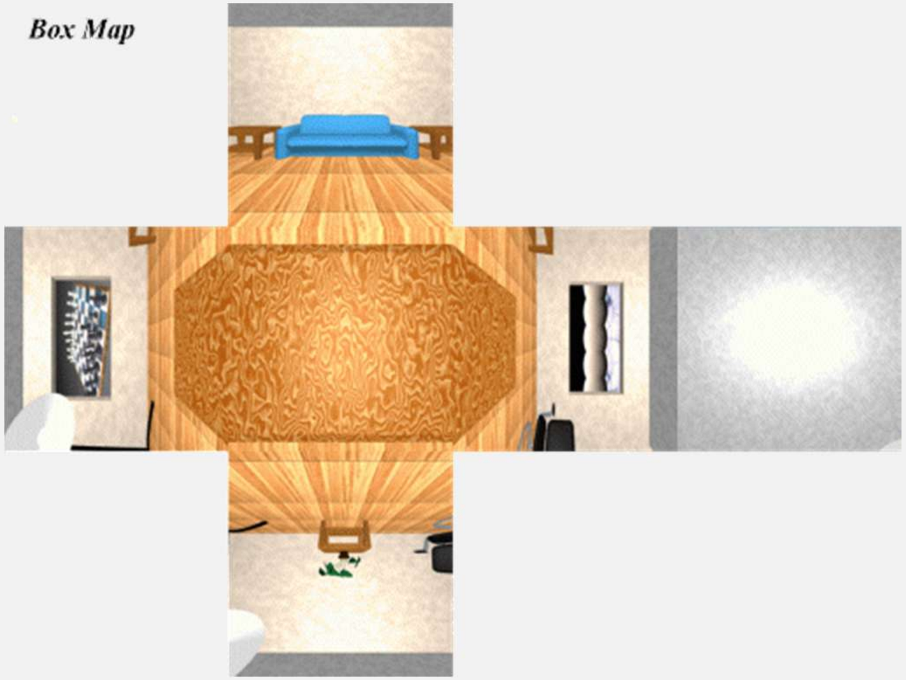


Environment Mapping (Cont.)

- Project the environment onto a sphere centered at the center of projection



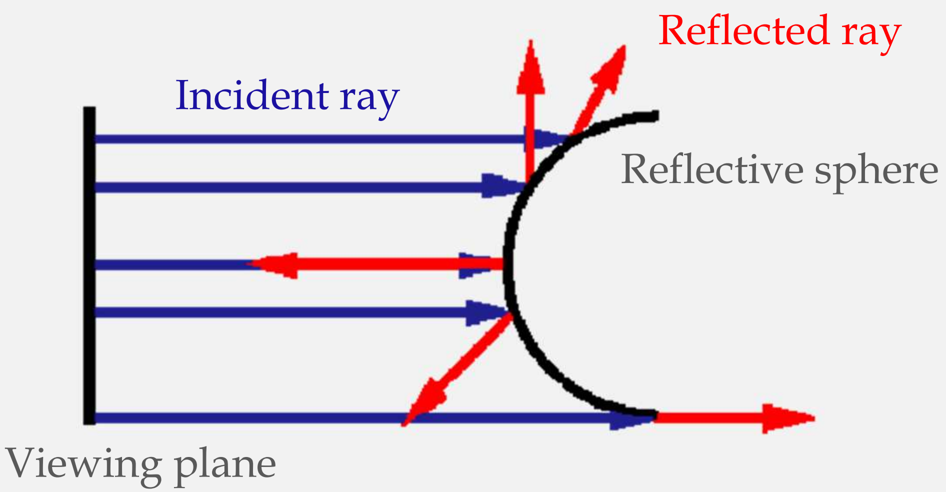
Environment Mapping Pictures



Pictures from lecture notes of Computer Graphics course, UNC

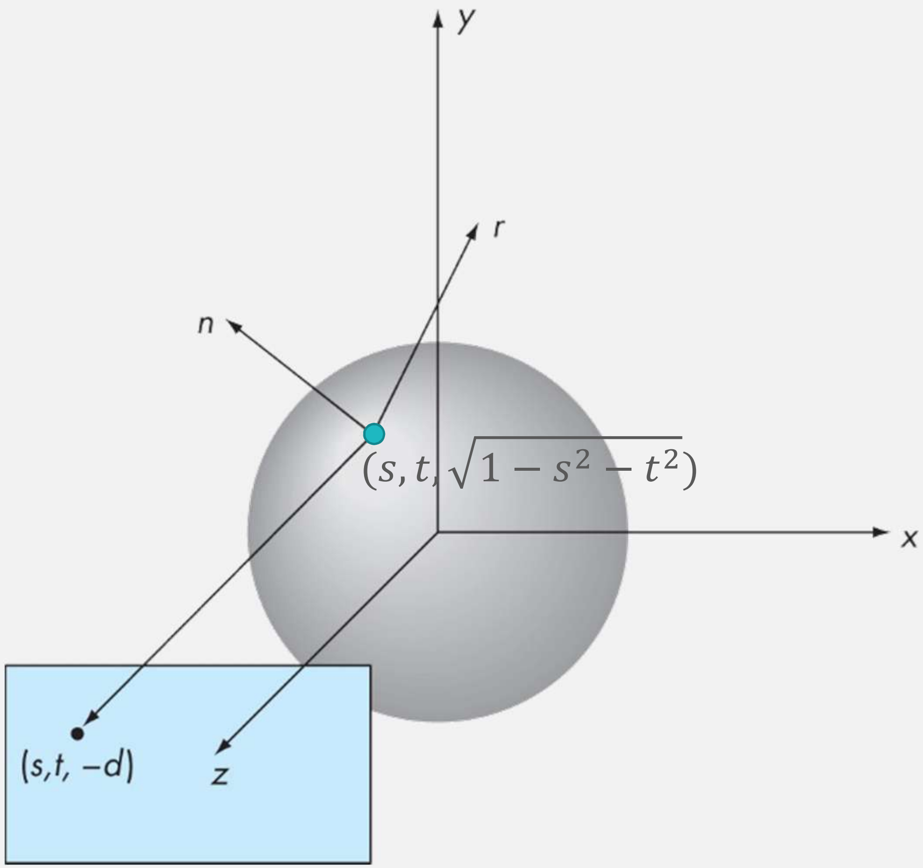
Sphere Mapping

- The image texture is taken from a perfectly reflective sphere.
- Assume the size of the sphere $\rightarrow 0$.
  - Map the rays to the environment

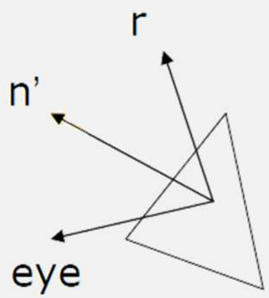


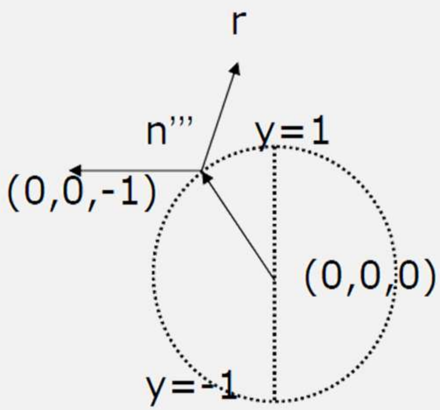
Pictures from OpenGL tutorial. http://www.opengl.org

Sphere Mapping (Cont.)



Sphere Mapping (Cont.)





- To access the sphere map texture
  - Compute the reflection vector on the object surface as usual

    $r = \left(r_x, r_y, r_z\right) = e' - 2(n' \cdot e')n'$

  - Now, compute the sphere normal in the local space

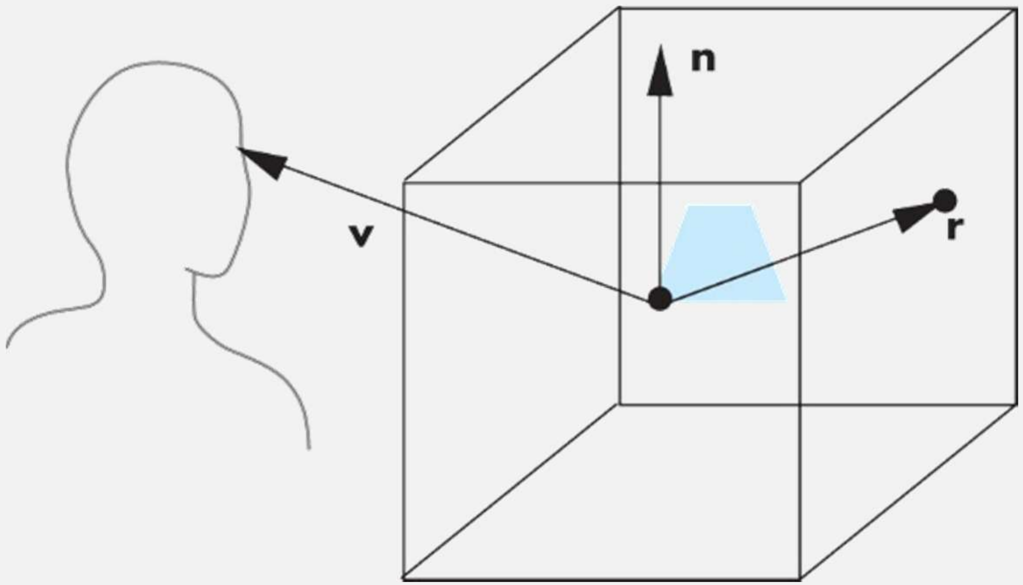
    n'' = (r_x, r_y, r_z) + (0,0,-1)

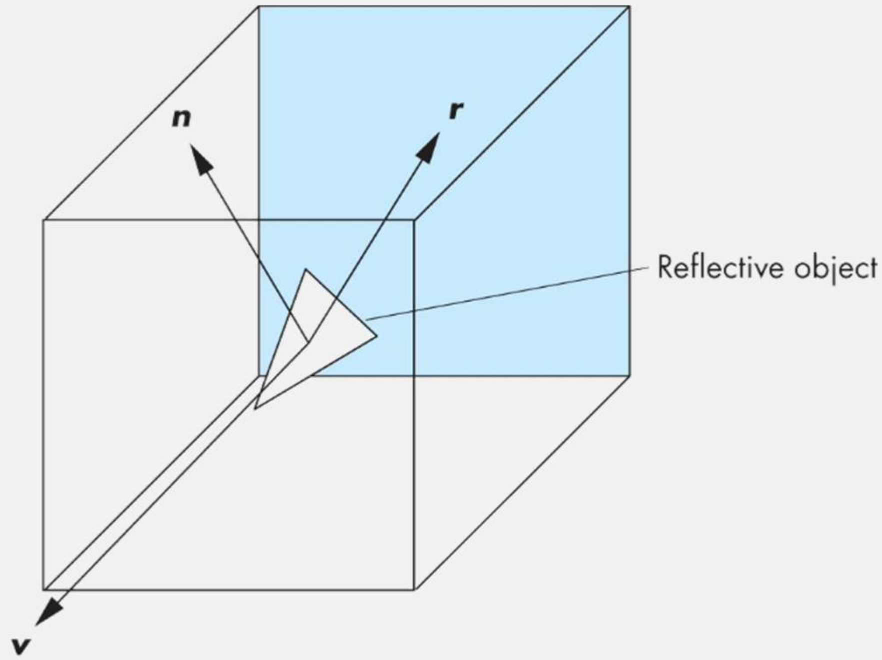
    $n''' = \left(\frac{r_x}{m}, \frac{r_y}{m}, \frac{r_z - 1}{m}\right)$ $\quad m = \sqrt{{r_x}^2 + {r_y}^2 + (r_z - 1)^2}$

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

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

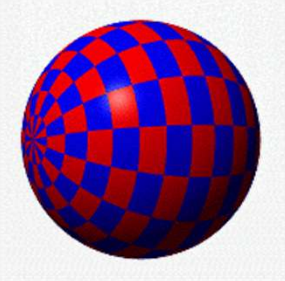
Reflective Cube Map

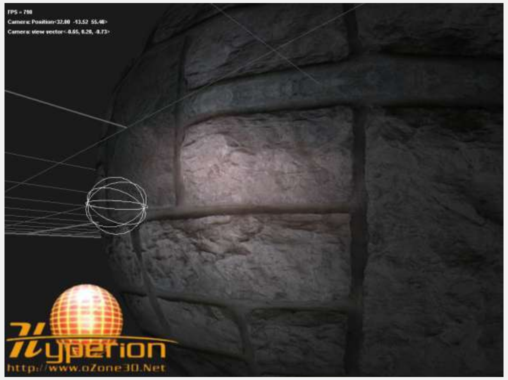


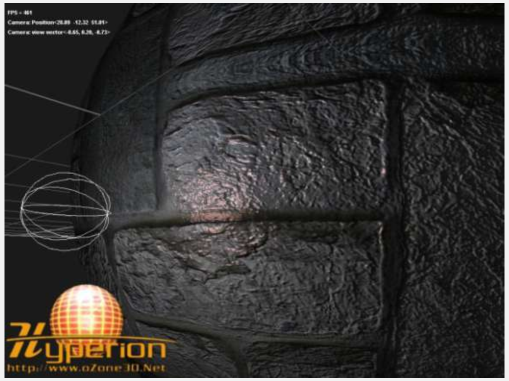


Bump and Normal Mapping

- Represent surface details and avoid heavy geometric computation.



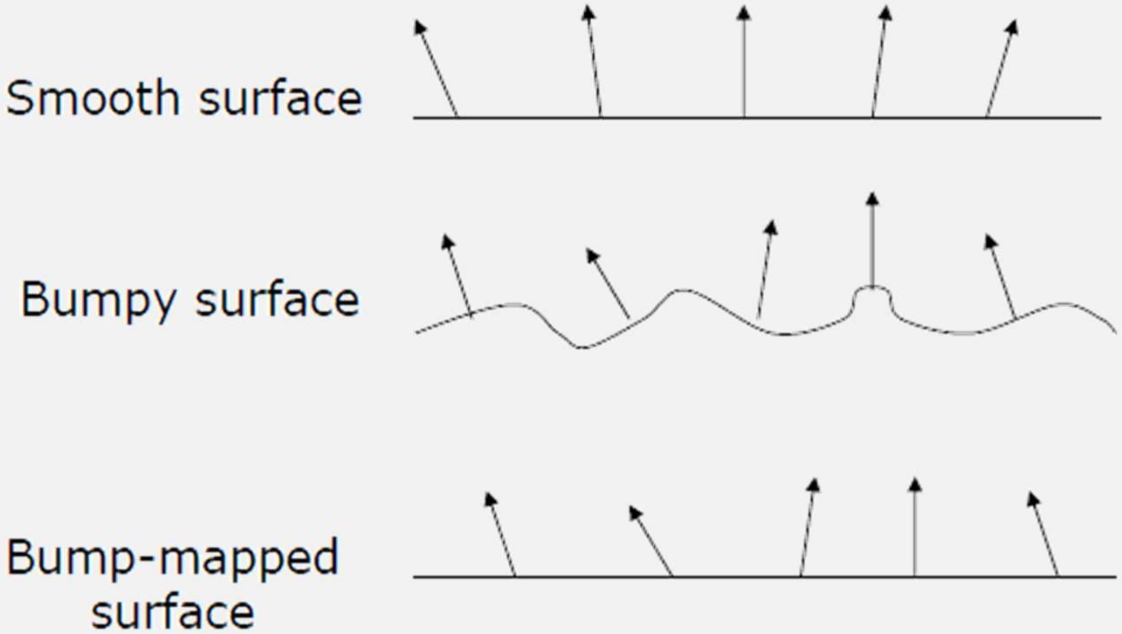




http://www.ozone3d.net/tutorials/bump_mapping.php

Bump and Normal Mapping (Cont.)

- Calculate reflection (Phong Shading) with a normal map.
- Or with a height map.



Bump Mapping

- Let **p**= **p**(u,v) be a smooth parametric surface, with normals **n**= **n**(u,v).

$$\mathbf{p}_u = \begin{bmatrix} \frac{\partial x}{\partial u} \\ \frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial u} \end{bmatrix} \qquad \mathbf{P}_v = \begin{bmatrix} \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial v} \end{bmatrix} \qquad \mathbf{n} = \mathbf{p}_\mathrm{u} \times \mathbf{p}_v$$

- Apply a bump map b= b(u,v):

$$\mathbf{p}' = \mathbf{p} + d(u,v)\mathbf{n}$$ Displaced surface

$$\mathbf{n}' = \mathbf{p}_u{}' \times \mathbf{p}_v{}'$$ Normal at p′

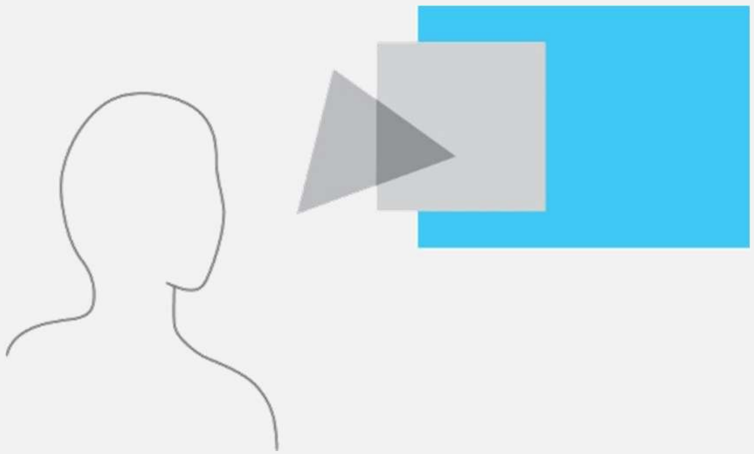
$$\mathbf{p}_u{}' = \frac{\partial}{\partial u}(\mathbf{p} + d(u,v)\mathbf{n}) = \mathbf{p}_u + d_u\mathbf{n} + d(u,v)\mathbf{n}_u$$

$$\mathbf{p}_v{}' = \frac{\partial}{\partial v}(\mathbf{p} + d(u,v)\mathbf{n}) = \mathbf{p}_v + d_v\mathbf{n} + d(u,v)\mathbf{n}_v$$

$$\mathbf{n}' \approx \mathbf{n} + d_u(\mathbf{n} \times \mathbf{p}_v) + d_v(\mathbf{n} \times \mathbf{p}_u)$$

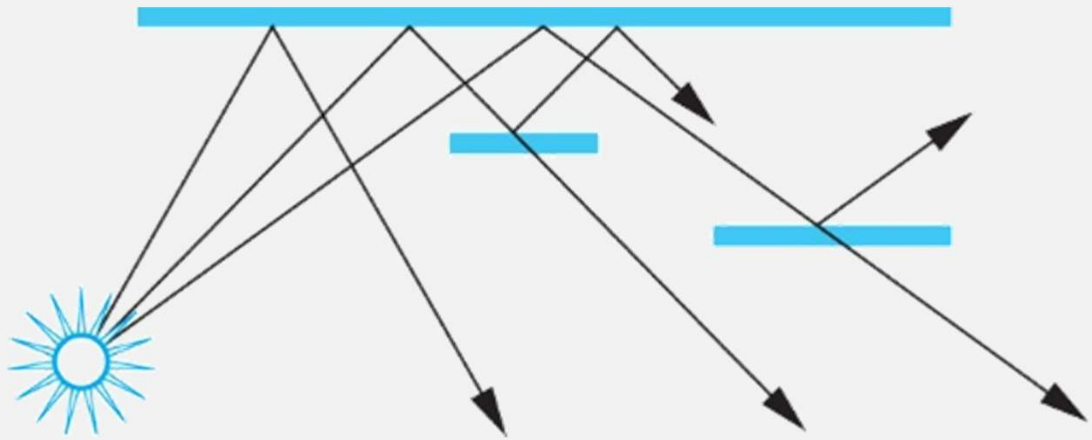
Opacity and Transparency

- 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

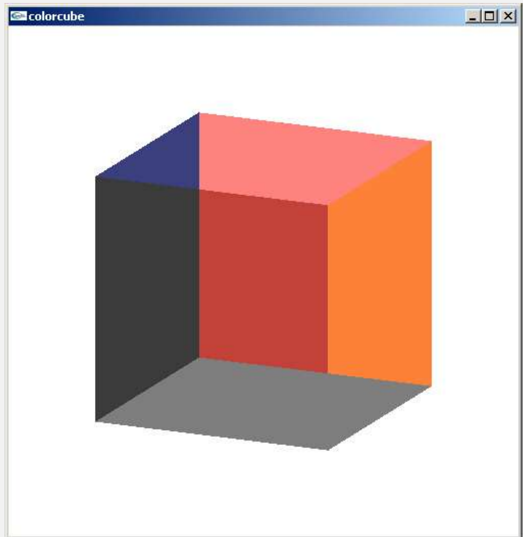


Blending Equation

- We can define source and destination blending factors 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 colors are
  - b=$[b_r, b_g, b_b, b_\alpha]$
  - c=$[c_r, c_g, c_b, c_\alpha]$
- Blend as
  - c′=$[b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_\alpha s_\alpha + c_\alpha d_\alpha]$

Order Dependency

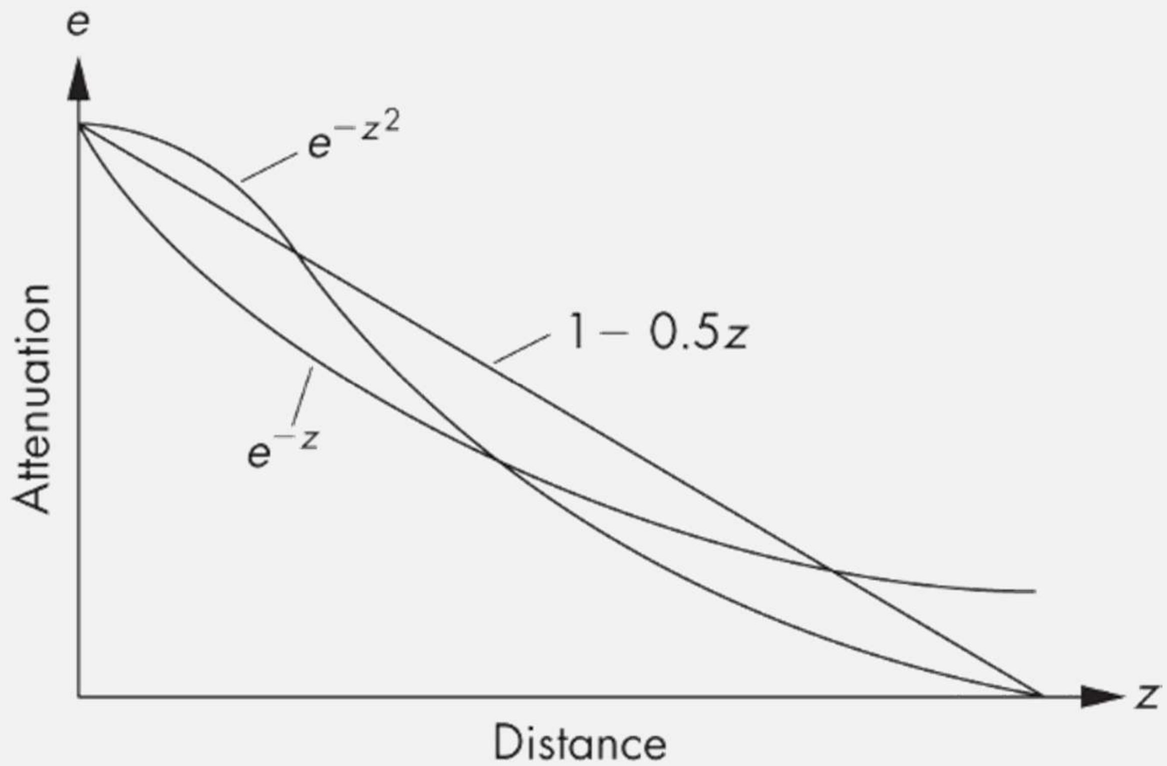
- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent



Fog

- We can composite with a fixed color and have the blending factors depend on depth
  - Simulates a fog effect
  - Blend source color C_s 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



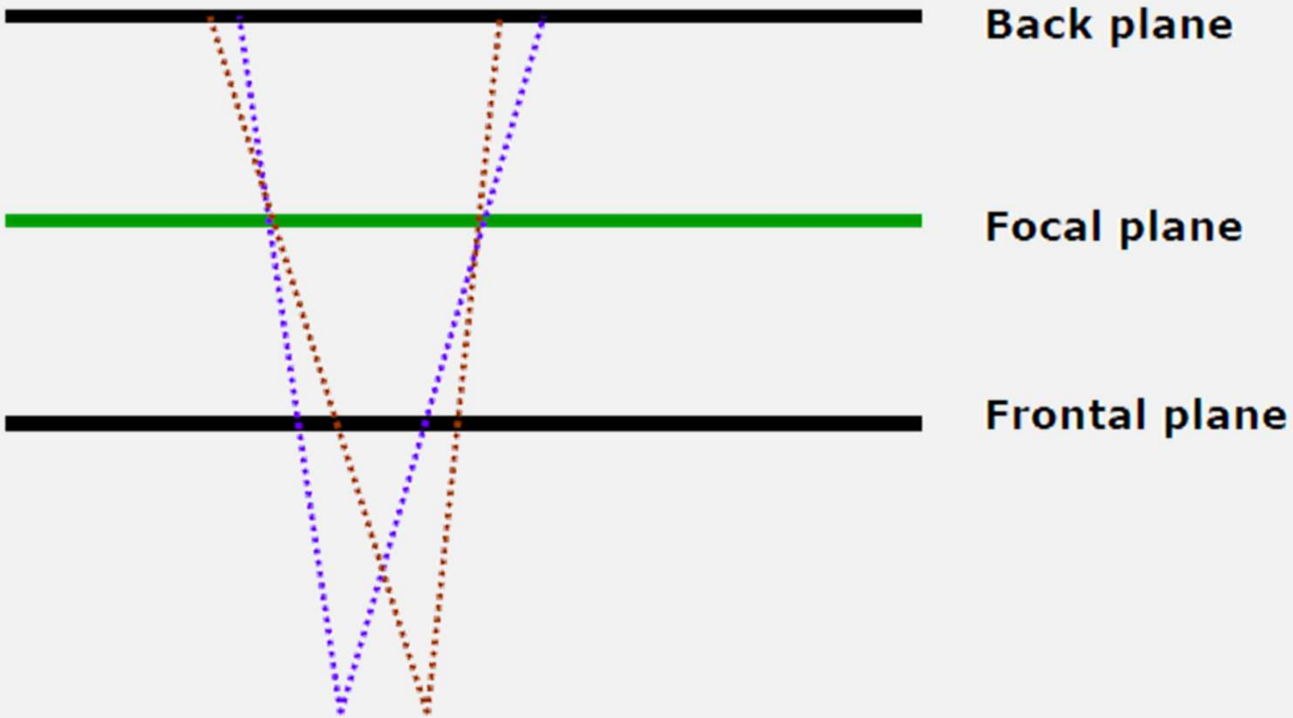
Accumulation Buffer

- 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

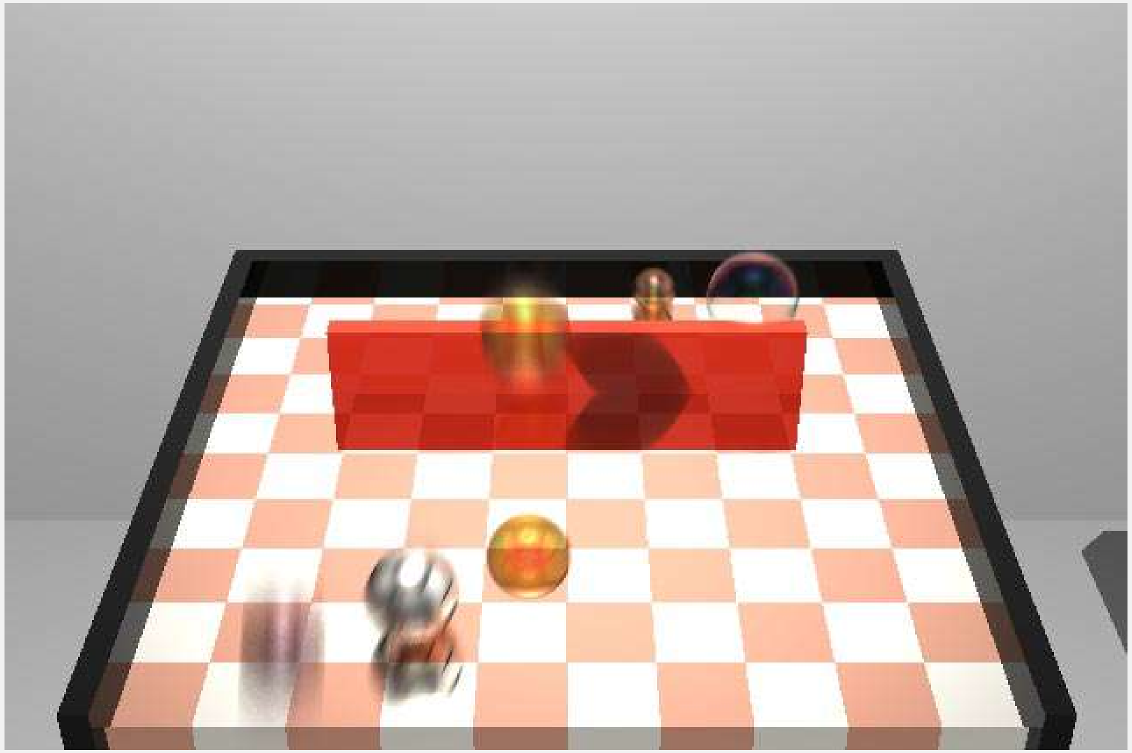
Applications of Composition Techniques

- Compositing
- Image Filtering
- Whole scene antialiasing
- Motion effects
- … …

Depth of Focus



Motion blur



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